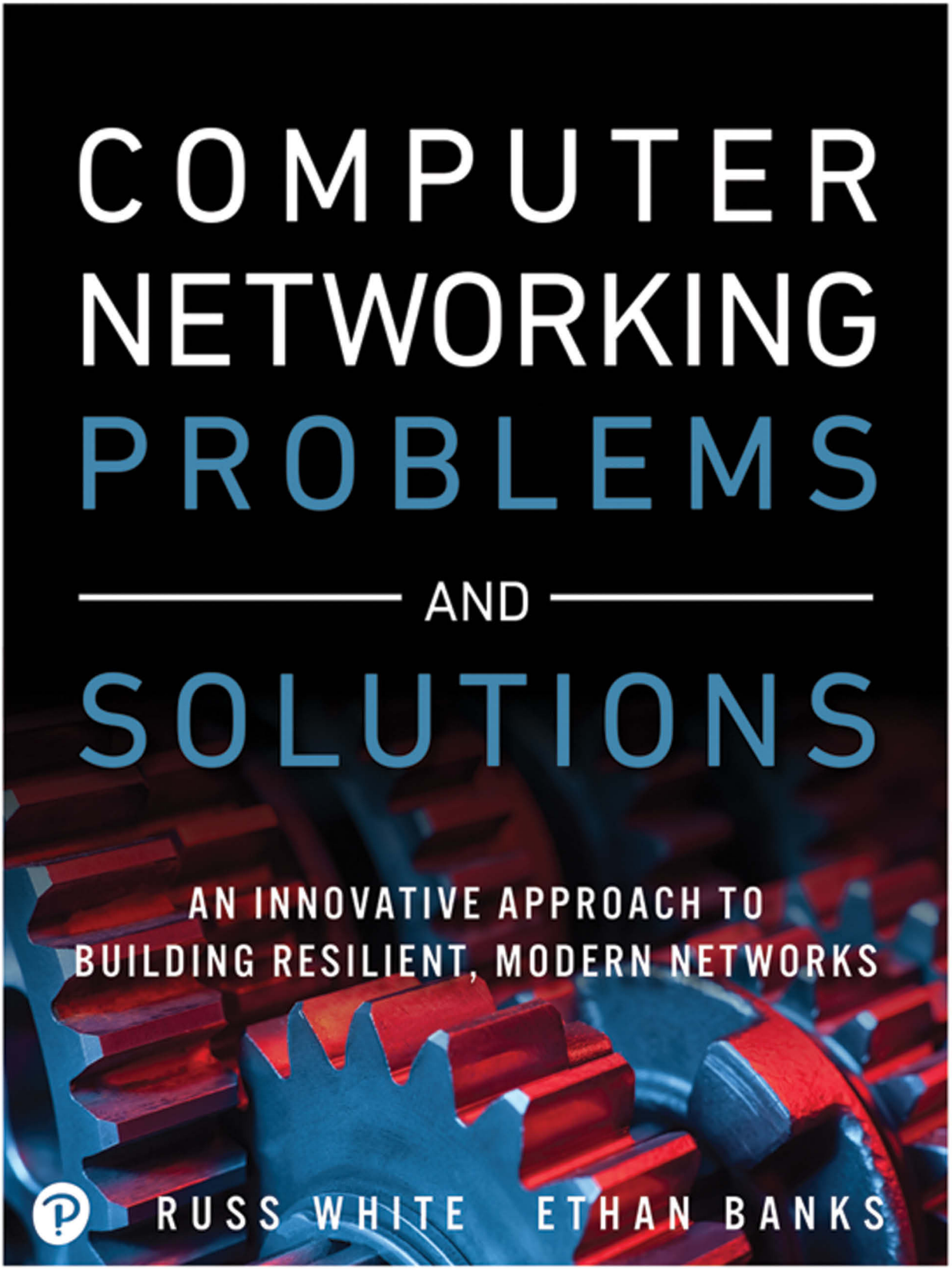
COMPUTER NETWORKING PROBLEMS AND SOLUTIONS



AN INNOVATIVE APPROACH TO BUILDING RESILIENT, MODERN NETWORKS

RUSS WHITE ETHAN BANKS

Computer Networking Problems and Solutions

This page intentionally left blank

Computer Networking Problems and Solutions

An innovative approach to building resilient, modern networks

Russ White and Ethan Banks

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco
Amsterdam • Cape Town • Dubai • London • Madrid • Milan
Munich • Paris • Montreal • Toronto • Delhi • Mexico City • São Paulo
Sidney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2017958319

ISBN-13: 978-1-58714-504-9
ISBN-10: 1-58714-504-9

1 17

Editor-in-Chief
Mark Taub

Product Line Manager
Brett Bartow

Development Editor
Christopher Cleveland

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Chuck Hutchinson

Indexer
Ken Johnson

Proofreader
Abigail Manheim

Technical Reviewers
Peter Welcher, Jordan Martin

Publishing Coordinator
Vanessa Evans

Cover Designer
Chuti Prasertsith

Compositor
codeMantra

To Lori, my beautiful wife of 20 years.

To Bruce Little and Doug Bookman; for challenging me to think.

To Brett Bartow, Eyvonne Sharp, Phil Gervasi, and Jordan Martin; for inspiring me.

May God bless each of you for the blessings you have brought into my life.

—Russ White

To Summerset; for enabling me to pursue the things I must chase.

To Drew Conry-Murray; for commiseration, advice, and encouragement.

To Robin Young and Greg Ferro; for freedom to write and moral support.

To Jordan Martin; for not saying no.

To the Packet Pushers community; for their multiplied voices, both frustrated and victorious.

—Ethan Banks

Contents

Acknowledgments

To begin, this book would not have been written if the need had not been recognized by Radia Perlman, hence planting the seed of the idea this book grew in to. Beyond the seed, however, a book does not represent the work of two authors; many people are actually involved in the process of creating and publishing the kind of high-quality content you now have access to. Below is a (hopefully complete) list of those who have participated in the creation of this content.

Ignas Bagdonas is an architect at Equinix, where he focuses on large-scale design of interconnection fabrics and network automation. Ignas has implemented BGP as part of his work at Routing System, Ltd.

Chris Kane is currently a systems engineer for Arista Networks, where he works on designing and deploying large-scale networks and is a founding member of the Ohio Networking User Group. Chris has been in the networking industry for over 25 years now, having worked in various verticals including Service Provider, Financial, Retail, and Consulting.

Kim Pedersen, CCIE 29189, CCDE 2017:0021, is a network engineer at Lytzen IT A/S, where he focuses on network design and the maintenance and development of international MPLS networks. He has a passion for learning new technical topics and is an avid reader of all things networking. He lives in Denmark with his wife and enjoys traveling!

Nick Russo, CCIE 42518, CCDE 2016:0041, is a network engineer at Cisco Systems in the Aderdeen, Maryland area, where he focuses on service provider, large-scale MPLS, and mobility design, as well as network automation. Nick is the author of the *CCIE Service Provider Version 4 Written and Lab Exam Comprehensive Guide,* available on LeanPub.

Maria Urlea, CCDP, CCDA, CCNP, CCNA, is a systems engineer at Cisco Systems in Ontario, Canada. Maria has received several master's scholarships and student research awards, and focuses on network design and architecture for several large network operators.

Chris Cleveland is one of the finest development editors in the network engineering space; he has worked with Russ on 13 projects in conjunction with Pearson since 1997.

About the Authors

Russ White, CCIE No. 2635, CCDE 2007::1, CCAr, has more than 30 years of experience in designing, deploying, breaking, and troubleshooting large-scale networks. In that time, he has co-authored more than 40 software patents, spoken at venues throughout the world, participated in the development of several Internet standards, helped develop the CCDE and the CCAr, and worked in Internet governance with the Internet Society. Russ is currently a member of the architecture team at LinkedIn, where he works on next-generation data center designs, complexity, security, and privacy. He is also currently on the routing area directorate at the IETF and co-chairs the IETF I2RS and BABEL working groups. His most recent books are *The Art of Network Architecture* and *Navigating Network Complexity*.

Russ holds an MSIT from Capella University, a MACM from Shepherds Theological Seminary, and a PhD in progress from Southeastern Theological Seminary.

Ethan Banks, CCIE No. 20655, Routing & Switching, has been in IT since 1995, working early in his career as a systems engineer for Novell, Windows, and Linux environments. He later became an Internet services engineer working with DNS, SMTP, HTTP, and related applications at a regional ISP. He predominantly has been a network engineer and architect for enterprises in verticals including higher education, state government, consulting, finance, and technology. He has held titles such as senior network engineer, network operations manager, technical services manager, network architecture manager, and senior network architect.

In 2010, Ethan co-founded Packet Pushers Interactive, a media company whose premier product is a weekly podcast listened to by more than 10,000 network engineers all over the world.

Ethan is a writer whose content can be found in *Network World*, *Network Computing*, *InformationWeek*, *Modern Infrastructure*, and *TechTarget*, among other outlets. Ethan also maintains his own blog where he writes about technology at ethancbanks.com. Ethan has written and/or edited whitepapers for SolarWinds, Nuage Networks, CloudGenix, and NetBrain Technologies. He is currently the Future of Networking co-chair for Interop.

Ethan holds a Bachelor of Science degree in Computer Science & Business Administration from Pensacola Christian College in Pensacola, Florida where he graduated Summa Cum Laude in 1993. In the past, Ethan was certified as a Certified Netware Engineer, Microsoft Certified Systems Engineer, Cisco Certified Network Professional, Certified Ethical Hacker, and Cisco Certified Security Professional, among other titles.

Introduction

There are many ways to approach teaching (or understanding) the fundamentals of computer network operation. For instance, one rather traditional way is to begin by examining the operation of a control plane in total, from building neighbor adjacencies to carrying information to building routes. Another common method is to start with a model, such as the Open Systems Interconnect (OSI) model, and describe the operation of the protocols from within the model. These methods have obviously been useful in teaching engineers and engineering students about how computer networks work, as they have been used to teach thousands, perhaps hundreds of thousands, of network engineers over the last 30 years.

But—in the view of the authors writing here—they have not been as effective as they could be. There are still many engineers who do not understand the basics of how a computer network actually works, in spite of many hours spent in labs, reading technical material, and even configuring and deploying network equipment. There is still a large gap in the fundamental mental skills of a large number of network engineers and engineering students that needs to be filled.

This book aims to fill that gap—not only for existing engineers, but also for all students who are trying to learn how computer networks work, even if network engineering is not their ultimate career goal. If you are a computer science student, a network engineer with 20 years of experience, someone just trying to learn network engineering, or even a business manager in charge of "the network," this book has something to offer you.

How This Book Is Organized

This book was born of more than 50 years of combined experience in the field of network engineering, split between two authors who have, over those years, taken in everything from forwarding devices to control planes to storage to compute. The authors (and reviewers!) have spent thousands of hours teaching the many different crafts involved in network engineering in formal and informal training, across a wide range of formats and venues. The organization of this book is the result of numerous hours spent considering how best to approach the many aspects of computer

network technologies. What works and (more importantly) what does not work were considered in detail, until a plan finally emerged that the authors believe will be helpful to the largest possible set of people in and around the computer networking field.

The organization of this book begins with a seed laid out in the Internet Engineering Task Force (IETF) Request for Comments (RFC) 1925, *The Twelve Networking Truths*. Rule 11 states:

> Every old idea will be proposed again with a different name and a different presentation, regardless of whether it works.

While this is clearly humorous, humor would not be funny without at least a grain of truth. In the case of rule 11, there is more than a grain of truth: buried in rule 11, there is an entire way of looking at technology, and the pace of technological change, that can revolutionize the way engineers learn technology. If it is true that every idea will be proposed again, then it is also true that every idea has been proposed before. If it were possible to learn the basic concepts behind an idea the first time it is proposed, it should be possible to understand every new proposal grounded in the same ideas in the future.

This observation—the grounding ideas behind the technologies that make computer networks work do not really change—is what drives the teaching method used in this book. Instead of focusing on models or protocols, this book follows a distinct pattern.

The thesis of this book is, then: To truly understand computer networks, you need to ask and answer three questions: What is the problem? What are the possible solutions? What do these solutions look like when they are implemented?

What Is the Problem?

This book is divided into three major parts covering data transport, the control plane, and specific design (or rather technology) situations. Within each of these parts, there are sets of chapters that begin by asking a basic question: what is the problem? Describing the problem set in a meaningful way will often involve a good bit more theory work, so these chapters may not, at first, seem to be very practical.

These chapters, however, are extremely practical; without a solid understanding of the problem, it is almost impossible to really understand any proposed or implemented solution in the correct context. Understanding the fundamental problems allows you to do two things:

- Relate problems you are facing right now, problems that might appear to be new, or unique, to a common body of problems solved in network engineering in the past.
- See and understand the component problems within a larger system clearly, and hence have a solid chance at applying a full range of solutions to each problem in a way that builds a complete and coherent system.

Asking this question is, in reality, the most important step you can take in truly understanding the technologies used to solve network engineering problems.

What Is the Solution?

Once the problem is laid bare, this book will then consider a range of possible solutions. The set of solutions will not (necessarily) be restricted to the most common solutions, or to implemented solutions. Rather, the solutions chosen for inclusion will (hopefully) provide you with a good overview of the types of solutions available. Again, this part will tend to be theoretical, specifically in describing point solutions designed to solve point problems. Again, the appearance of impracticality will be wrong—each solution is a "tool" you can add to the set of mental tools you can use to solve a wide array of problems. Combining problems and solutions in this way thus builds a solid set of mental skills useful for engineers of any type.

How Has This Been Implemented?

Finally, once a set of problems and a range of solutions for each problem have been considered, the problems and solutions will be drawn together into a set of implementation examples. This part is where you will see the connection between theory and practice: how each protocol sets out to solve a common set of problems and then selects among a range of solutions to solve those problems. The authors have striven to choose a wide range of protocols and systems for these parts, so you are not only carried through the solution space, but also (as much as possible within the confines of a work of this type) the history of computer network engineering.

What This Book Does Not Cover

Any book written in this field could be endless in scope—but such an endless book would not be constrained enough to be useful. To manage the scope and scale of this book, then, several choices have been made about what to cover and what not to cover.

Packet switched networks are covered; circuit switched networks are not. In packet switched networks, information is carried in packets, each of which contains enough information to route the packet through the network, from end to end. There is no "fixed" line of communications between the sender and receiver; just an underlying set of packet forwarding devices that, acting as a complete system, deliver these packets on a best-effort basis. Circuit switched networks can break up information in a way that does not require each packet to carry all the information needed to

forward the information, and there are agreed-on paths and resources tied to each particular information flow.

The data and control planes are covered, but not the management plane. It is often difficult to determine where the data plane ends and the control plane begins. Likewise, it is often difficult to determine where the control plane ends and the management system begins. The authors have, based on their extensive experience, attempted to include just those topics related to building and managing paths available for forwarding packets through a network, while leaving out topics that appear to be more network management focused.

These omissions are not a statement about the importance of the topics in question; rather, a book such as this must be scoped in some way if it is to be writable by any set of humans in anything like a reasonable amount of time.

On Reading Flow

In many ways, understanding how the authors intend a book to be read is just as important of a guide to understanding how to use the material as understanding how the information is structured, or what question the book is trying to answer. This book is designed to reach a broad audience, from the "average" network engineer, to people trying to learn network engineering without any formal training, to college classrooms.

To reach across this scope, the authors have taken several specific steps:

- The material presented in the main text, while of varying depth (as required by the specific topic), will strive to maintain an introductory feel. The main flow of text will strive to use as few "big words" and "heavy symbols" as possible.
- More technical material, historical asides, and other material that the authors believe will be useful to those trying to learn network engineering will be placed into sidebars.
- Footnotes will only be provided to give credit to specific works that originated ideas or the works of specific individuals known for originating specific ideas. Explanations that would normally be placed in a footnote in other contexts will be placed in a sidebar.
- More deeply technical papers and resources will be listed at the end of each chapter for those who would like to investigate a specific topic more deeply. These items will have some information about which specific topic they are related to where possible.

A Beginning

A great deal of time and effort have gone into researching, writing, editing, and producing this book. The authors and editors who have worked on this represent some of the broadest, and often deepest, experience in every aspect of network engineering—protocol design and specification, protocol implementation, network design, network implementation, troubleshooting, and many others. Hopefully, this book will provide you with a deep and broad foundation from which to truly understand how computer networks work, and hence lay the groundwork you need to design, implement, and manage protocols and networks that will solve real-world problems for many years to come.

Reader Services

Register your copy of *Computer Networking Problems and Solutions* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account*. Enter the product ISBN (9781587145049) and click Submit. When the process is complete, you will find any available bonus content under Registered Products.

*Be sure to check the box that you would like to hear from us to receive exclusive discounts on future editions of this product.

Part I

The Data Plane

To begin, the primary job of a network is to carry data from one attached host to another. This might appear to be simple at first glance, but it is actually fraught with problems. An illustration might be helpful here; Figure PI-1 is used to illustrate the complexity.

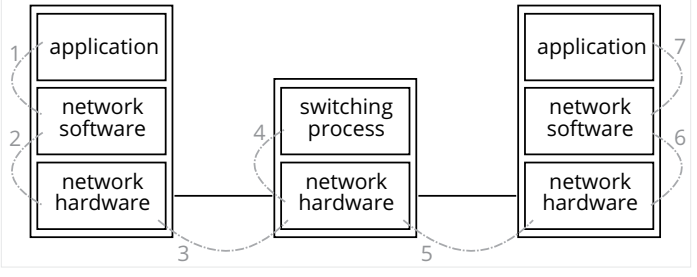


Figure PI-1 *The Data Plane Problem Space*

Beginning at the upper-left corner of the illustration:

1. The application generates some data. This data must be formatted in a way that allows the receiving application to understand what has been transmitted—the data must be marshalled. The mechanism used to marshal the data must be efficient in many ways, including fast and easy to encode, fast and easy to decode, flexible enough to allow for changes in encoding without breaking too many things, and adding the smallest amount of overhead possible during data transmission.

2. The network software needs to encapsulate the data, and get it ready to actually be transmitted. Somehow the network software needs to know the address of the destination host. The network that connects the source and destination is a *shared resource*, and hence some form of multiplexing must be available so the source can direct the information at the correct destination. Generally this will involve some form of addressing.
3. The data must be moved out of memory at the source and onto the network proper—the actual wire (or optical cable, or wireless link) that will carry the information between network-connected devices.
4. Network devices must have some way to discover the ultimate destination of the information—a second form of the multiplexing problem—and determine if there is any other processing that needs to be done on the information while it is in transit between the source and destination.
5. The information, after passing through the network device, must once again be encoded and moved out of memory onto the wire. At every point where information is moved from memory to some form of physical media, the information will need to be queued; there will often be more data to transmit than can be placed onto any particular physical media at any given time. This is where quality of service comes into play.
6. The information, as carried through the network, must now be copied off the physical media and back into memory. It must be checked for errors—this is error control—and there must be some way for the receiver to tell the transmitter it is running out of memory in which to store the incoming information—this is flow control.

The network device in the middle of the diagram is of particular interest. A network device—such as a router, switch, or middle box—connects two physical media together to build an actual network. Perhaps the simplest question to begin with is this: why are these devices required in the first place? Routers and switches are obviously complex devices, with their own internal architecture (which will be covered in this chapter at a high level); why add this complexity to a network? There are two fundamental reasons.

The original reason for building these devices was to connect different kinds of physical media together. For instance, within a building it might be practical to run ARCnet or thicknet Ethernet (to use examples from the time when network devices were first invented). The distance these media can traverse, however, is very short—on the order of hundreds of meters. Somehow these networks must be extended between buildings, between campuses, between cities, and eventually between

continents, using some sort of multiplexed (or inverse multiplexed) telephone circuit like a T1 or DS3. These two different media types use different kinds of signaling; there must be some sort of device that translates one kind of signaling into another.

The second reason is this: scale quickly became an issue. The nature of the physical world is such that you have two choices when it comes to putting data on a wire:

- The wire can connect precisely two computers; in this case, every pair of computers needs to be physically connected to every other computer it needs to communicate with.
- The wire can be shared among many computers (the wire can be a shared media).

To solve the problem the first way, you need a lot of wire. Solving the problem the second way seems like the obvious solution, but it presents another set of problems—specifically, how is the bandwidth available on the wire shared among all the devices? At some point, if there are enough devices on a single shared media, any sort of scheme used to enable resource sharing will, itself, consume as much or more bandwidth as any individual device connected to the wire. At some point, even a 100G link, shared among enough hosts, will leave each individual host with very little available resources.

The solution to this situation is the network device—the router or switch—that separates two shared media, only passing traffic between the two as needed. With some logical planning, devices that need to talk to each other more often can be placed closer together (in terms of network topology), conserving bandwidth in other places. Routing and switching has moved far beyond these humble beginnings, of course, but these are the root problems engineers solved by injecting network devices into networks.

There are other difficult problems to solve in this space beyond the bare carrying of information from a source to a destination; many times it is useful to be able to virtualize the network, which generally means creating a tunnel between two devices in the network.

The series of chapters in Part I consider the sometimes incredibly difficult problems in simply carrying data from one end of a network to the other, along with a range of possible solutions for each of these problems. Along the way, various chapters also explore the concept of layering in data transport protocols, and its importance to breaking this complex domain into more solvable chunks. Layering, however, brings its own set of problems into the transport world, so Part I also needs to consider how to resolve the problems caused by the introduction of layering—specifically, interlayer discovery.

The chapters in this part include:

- **Chapter 1: Fundamental Concepts**, which discusses business drivers, circuit switching, packet switching, and network complexity
- **Chapter 2: Data Transport Problems and Solutions**, which discusses marshaling data, dictionaries, grammars, metadata, error detection, error correction, addressing, multiplexing, multicast, anycast, and flow control
- **Chapter 3: Modeling Network Transport**, which discusses the value of modeling, the Department of Defense (DoD) model, the Open Systems Interconnect (OSI) model, the Recursive Internet Architecture (RINA) model, connection-oriented and connectionless transport mechanisms
- **Chapter 4: Lower Layer Transports**, which discusses Ethernet and 802.11 Wireless
- **Chapter 5: Higher Layer Data Transports**, which discusses the Internet Protocol (IP), the Transmission Control Protocol (TCP), QUIC, and the Internet Control Message Protocol (ICMP)
- **Chapter 6: Interlayer Discovery**, which discusses mapping identifiers and services between layers, the Domain Name System (DNS), the Address Resolution Protocol (ARP), Neighbor Discovery (ND), Stateless Address Autoconfiguration (SLAAC), and the concept of the default gateway
- **Chapter 7: Packet Switching**, which discusses the process of copying a packet off of the physical media, processing the packet, moving a packet through the network device, and finally copying a packet onto the physical medium
- **Chapter 8: Quality of Service**, which discusses why Quality of Service (QoS) is needed, traffic classification, Class of Service, Type of Service, QoS trust boundaries, jitter, and fairness in queueing
- **Chapter 9: Network Virtualization**, which discusses use cases for network virtualization, tunneling, switching tunneled packets, the problems network virtualization must solve, Segment Routing (SR), Software-Defined Wide Area Networks (SD-WAN), virtualization tradeoffs, and shared fate
- **Chapter 10: Transport Security**, which discusses data exhaust, asymmetric and symmetric encryption, key exchange, hiding user information, man-in-the-middle (MitM) attacks, and Transport Level Security (TLS)

Chapter 1

Fundamental Concepts

Learning Objectives

After reading this chapter, you should be able to:

- Understand the relationship between business drivers and network engineering
- Understand the difference between circuit and packet switching
- Understand the advantages and disadvantages of circuit and packet switching
- Understand the basic concept of network complexity and complexity tradeoffs

Networks were always designed to do one thing: carry information from one attached system to another. The discussion (or perhaps argument) over the best way to do this seemingly simple task has been long-lived, sometimes accompanied by more heat than light, and often intertwined with people and opinions of a rather absolute kind. This history can be roughly broken into multiple, and often overlapping, stages, each of which asked a different question:

- Should networks be circuit switched or packet switched?
- Should packet switched networks use fixed- or variable-sized frames?
- What is the best way to calculate a set of shortest paths through a network?
- How should packet switched networks interact with Quality of Service (QoS)?
- Should the control plane be centralized or decentralized?

Some of these questions have been long since answered, most often by blending the more extreme elements of each position into a sometimes messy, but generally always useful, solution. Some of these questions are, on the other hand, still active, particularly the last one. Perhaps, in twenty years' time, readers will be able to look on this last question as being answered, as well.

This chapter will describe the basic terms and concepts used in this book from within the framework of these historical movements or stages within the world of network engineering. By the end of this chapter, you will be ready to tackle the first two parts of this book—the forwarding plane and the control plane. The third part, an overview of network design, builds on the first two parts. The final part of this book looks at a few specific technologies and movements likely to shape the future—not only of network engineering, but even of the production and processing of information overall.

Art or Engineering?

One question that must be asked, up front, is whether network engineering is an art, or truly engineering. Many engineering fields begin as more of an art. For instance, in the early 1970s, working on and around electronics—tubes, "coils," and transformers—was largely considered an art. By the mid-1980s, electronics had become ubiquitous, and this began a commoditization process harsher than any standardization. Electronics then was considered more engineering than art. By the 2010s, electronics became "just the stuff that makes up computers." There is still some art in the designing and troubleshooting of electronics, but, by and large, their creation became more focused on engineering principles. The problems have moved from "how do you do that," to "what is the cheapest way to do that," or "what is the smallest way to do that," or some other problem that would have been considered second order in the earlier days. Perhaps one way to phrase the movement in electronics is in ratios. Perhaps (and these are very rough estimates), electronics started at around 80% art and 20% engineering, and has now moved to 80% engineering and 20% art.

What about network engineering? Will it pass through the same phases, eventually moving into the 80% engineering, 20% art range? This seems doubtful for several reasons. Network engineering works in a largely virtual space; although there are wires and devices, the protocols, data, and functionality are all laid on top of the physical infrastructure, rather than *being* the physical infrastructure. Unlike electronics, where you can point to a physical object and say, "this is the product," a network is not a physical thing. To put this another way, the network is a conceptual "thing" built using a wide array of individual components connected together through

protocols and data models. This means design choices are almost infinitely variable and malleable. Each problem can be approached, assessed, and designed much more specifically than in electronics. So long as there are new problems to solve, there will be new solutions developed, deployed, and—eventually (finally) removed from networks. Perhaps a useful comparison is between applications and the various kinds of computing devices; no matter how standardized computing devices become, there is still an almost infinite selection of software applications to run on top.

Figure 1-1 will be useful in illustrating this "fit" between the network and the business from one perspective.

In Figure 1-1, the solid gray curved line is *business growth*. The dashed black line running vertical and horizontal is *network capacity.* There are many times when the network is overprovisioned, costing the business money to maintain unused capacity; these are shown in the gray line-shaded regions. There are other times when the network is under strain. In these darker gray solid-shaded regions, the business could grow faster, but the network is holding it back. One of the many objectives of network architecture and design (this is more of an architecture issue than strictly a design issue; see *The Art of Network Architecture)* is to bring these lines closer together. Accomplishing this part of the work requires creativity and future thinking problem-solving skills. The engineer must ask questions like "How can the

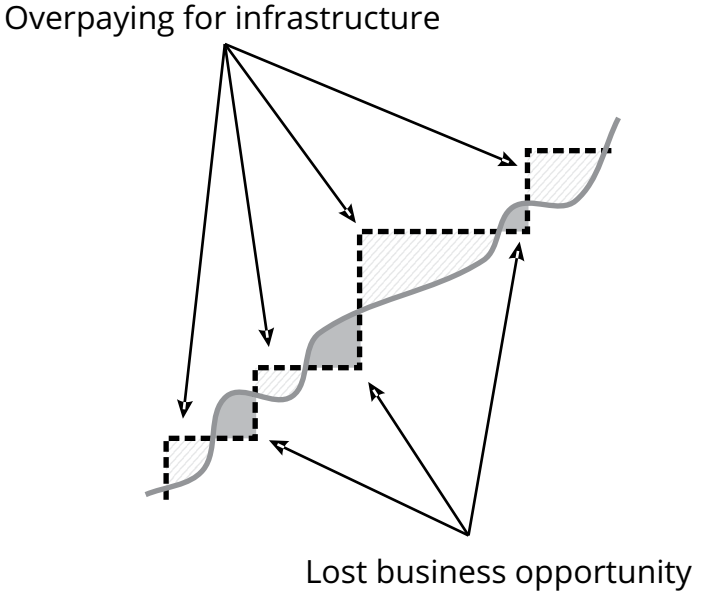


Figure 1-1 *Business to Technology Fit, First Perspective*

network be built so it scales larger and smaller to move with the business's requirements?" This is more than just scale and size; however, it is possible the nature of the business may even change over time, driving changes in applications, operational procedures, and operational pace. The network must have an architecture capable of changing as needed without introducing *ossification,* or the hardening of systems and processes that will eventually cause the network to fail in a catastrophic way. This part of the working on networks is often considered more *art* than *engineering,* and it is not likely to change until the entire business world changes in some way.

Figure 1-2 illustrates another way in which businesses drive network engineering as an art.

In Figure 1-2, time runs from the left to the right, and feature count from the bottom to the top. What the chart expresses is the additional features added to a product over time. Network operator A will start out needing a somewhat small feature set, but the feature set required will increase over time; the same will hold true of the other three networks. The feature sets required to run any of these networks will always overlap to some degree, *and they will also always be different to some degree*. If a vendor wants to be able to sell a single product (or product line) and cater to all four networks, it will need to implement every unique feature required by each network. The entire set of features is depicted by the peak of the chart on the right side. For each of the networks, some percentage of the features available in any product will be unnecessary—also known as *code bloat*.

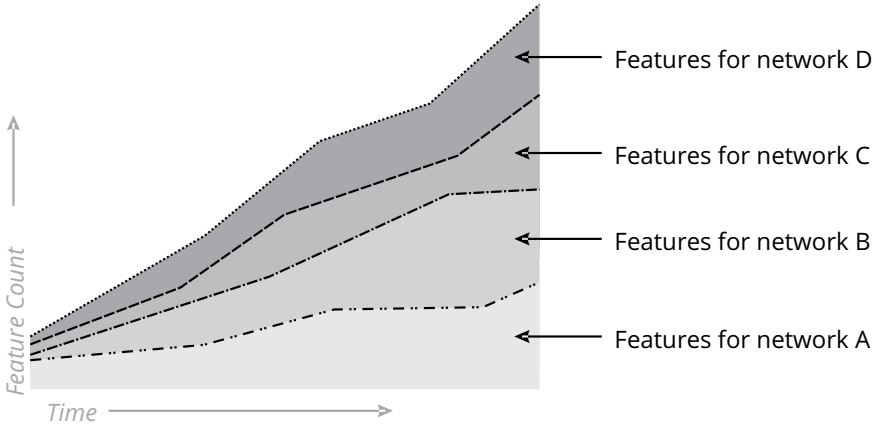


Figure 1-2 *Features versus Usage in Networking Products*

Even though these features are not being used, each one will still represent security vulnerabilities, code that must be tested, code that interacts with features that *are* being used, etc. In other words, *each one of these unused features is actually a liability for the network operator.* The ideal solution might be to custom build equipment for each network, containing just the features required—but this is often not a choice available to either the vendor or the network operator. Instead, network engineers must somehow balance between required features and available features—and this process is definitely more a form of *art* than *engineering.*

So long as there are mismatches between the way networks can be built and the way businesses use networks, there will always be some interplay between *art* and *engineering* in networking. The percentage of each one will vary based on the network, tools, and the time within the network engineering field, of course, but the *art* component will probably always be more strongly represented in the networking field than it is in fields like electronics engineering.

Note

Some people might object to the use of the word *art* in this section. It is easy enough to replace *art* with *craft,* however, if this makes the concepts in this section easier to understand.

Circuit Switching

The first large discussion in the computer networking world was whether networks should be circuit switched or packet switched. The basic difference between these two is the concept of a circuit—do the transmitter and receiver "see" the network as a single wire, or connection, preconfigured (or set up) with a specific set of properties before they begin communicating? Or do they "see" the network as a shared resource, where information is simply generated and transmitted "at will"? The former is considered circuit switched, while the latter is considered packet switched. Circuit switching tends to provide more traffic flow and delivery guarantees, while packet switching tends to deliver data at a much lower cost—the first of many tradeoffs you will encounter in network engineering. Figure 1-3 will be used to illustrate circuit switching, using Time Division Multiplexing (TDM) as an example.

In Figure 1-3, the total bandwidth of the links between any two devices is split up into eight equal parts; A is sending data to E using time slot A1 and F using time slot A2; B is sending data to E using time slot B1 and F using time slot B2. Each piece of information is a fixed length, so each one can be put into a single time slot in the

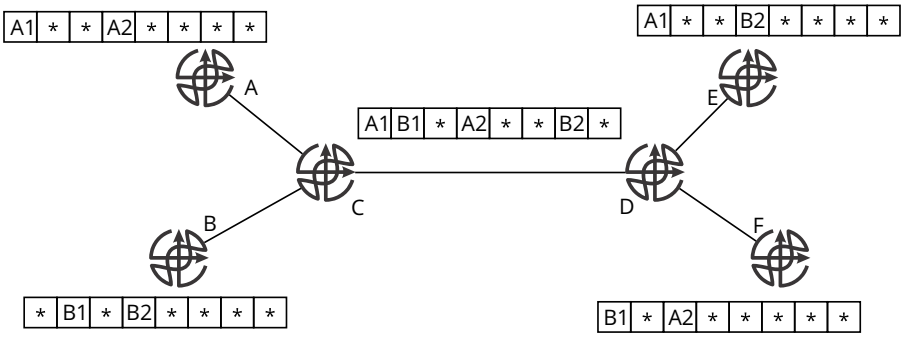


Figure 1-3 *Time Division Multiplexing Based Circuit Switching*

ongoing data stream (hence, each block of data represents a fixed amount of time, or slot, on the wire). Assume there is a controller someplace assigning a slot on each of the segments the traffic will traverse:

- **For [A,E] traffic:**
  - **At C:** slot 1 from A is switched to slot 1 toward D
  - **At D:** slot 1 from C is switched to slot 1 toward E
- **For [A,F] traffic:**
  - **At C:** slot 4 from A is switched to slot 4 toward D
  - **At D:** slot 4 from C is switched to slot 3 toward F
- **For [B,E] traffic:**
  - **At C:** slot 4 from B is switched to slot 7 toward D
  - **At D:** slot 7 from C is switched to slot 4 toward E
- **For [B,F] traffic:**
  - **At C:** slot 2 from B is switched to slot 2 toward D
  - **At D:** slot 2 from C is switched to slot 1 toward F

None of the packet processing devices in the network need to know which bit of data is going where; so long as C takes whatever is in slot 1 in A's data stream in each time frame and copies it to slot 1 in its outgoing stream toward D, and D copies it from slot 1 inbound from C to slot 1 outbound to E, traffic transmitted by A

will be delivered at E. There is an interesting point to note about this kind of traffic processing—to forward the traffic, none of the devices in the network actually need to know what the source or destination is. The blocks of data being transmitted through the network do not need to contain source or destination addresses; where they are headed, and where they are coming from, decisions are all based on the controllers' knowledge of open slots in each link. The set of slots assigned to any particular device-to-device communications is called a circuit, because it is bandwidth and network resources committed to the communications between the one pair of devices.

The primary advantages of circuit switched networks include:

- The devices do not need to read a header, or do any complex processing, to switch packets. This was extremely important in the early days of networking, when hardware had much lower transistor and gate counts, line speeds were lower, and the time to process a packet in the device was a large part of the overall packet delay through the network.
- The controller knows the available bandwidth and traffic being pushed toward the edge devices everywhere in the network. This makes it somewhat simple, given there is actually enough bandwidth available, to engineer traffic to create the most optimal paths through the network possible.

There are also disadvantages, including:

- The complexity of the controller ramps up significantly as the network and services it offers grow in scale. The load on the controller can become overwhelming, in fact, causing network outages.
- The bandwidth on each link is not used optimally. In Figure 1-3, the blocks of time (or *cells*) containing an * are essentially wasted bandwidth. The slots are assigned to a particular circuit ahead of time: slots used for the [A,E] traffic cannot be "borrowed" for the [A,F] traffic even when A has nothing to transmit toward E.
- The time required to react to changes in topology can be quite long in network terms; the local device must discover the change, report it to the controller, and the controller must reconfigure every network device along the path of each affected traffic flow.

TDM systems contributed a number of ideas to the development of the networks used today. In particular, TDM systems molded much of the early discussion

on breaking data into packets for transmission through the network, and laid the groundwork for much later work in QoS and flow control. One rather significant idea these early TDM systems bequeathed to the larger networking world is *network planes*.

Note

Quality of Service is briefly considered in a later section in this chapter, and then in more depth in Chapter 8, "Quality of Service," later in this book.

Specifically, TDM systems are divided into three planes:

- The **control plane** is the set of protocols and processes that build the information necessary for the network devices to forward traffic through the network. In circuit switched networks, the control plane is completely a separate plane; there is normally a separate network between the controller and the individual devices (though not always, particularly in newer circuit switched systems).
- The **data plane** (also known as the *forwarding* plane) is the path of information through the network. This includes decoding the signal received in a wire into frames, processing them, and pushing them back onto the wire, encoded according to the physical transport system.
- The **management plane** is focused on managing the network devices, including monitoring the available memory, monitoring queue depth, and monitoring when the device drops the information being transmitted through the network, etc. It is often difficult to distinguish between the management and the control planes in a network. For instance, if the device is manually configured to forward traffic in a particular way, is this a management plane function (because the device is being configured) or a control plane function (because this is information about how to forward information)?

Note

This question does not have a definitive answer. Throughout this book, however, anything that impacts the way traffic is forwarded through the network is considered part of the control plane, while anything that impacts the physical or logical state of the device, such as interface state, is considered part of the management plane. Do not expect these definitions to hold true in the real world.

Note

Frame Relay, SONET, ISDN, and X.25 are examples of circuit switched technology, some of which are still deployed at the time of writing. See the "Further Reading" section for suggested sources for learning about these technologies.

Packet Switching

In the early- to mid-1960s, packet switching was "in the air." A lot of people were rethinking the way networks had been built until then, and were considering alternatives to the circuit switched paradigm. Paul Baran, working for the RAND Corporation, proposed a packet switching network as a solution for survivability; around the same time, Donald Davies, in the UK, proposed the same type of system. These ideas made their way to the Lawrence Livermore Laboratory, leading to the first packet switched network (called *Octopus)* being put into operation in 1968. The ARPANET, an experimental packet switched network, began operation not long after, in 1970.

Packet Switched Operation

Note

The actual process of switching a packet is discussed in greater detail in Chapter 7, "Packet Switching."

The essential difference between circuit switching and packet switching is the role individual network devices play in the forwarding of traffic, as Figure 1-4 illustrates.

In Figure 1-4, A produces two blocks of data. Each of these includes a header describing, at a minimum, the destination (represented by the H in each block of data). This complete bundle of information—the original block of data and the header—is called a packet. The header can also describe what is inside the packet, and include any special handling instructions forwarding devices should take when processing the packet—these are sometimes called *metadata*, or "data about the data in the packet."

There are two packets produced by A: A1, destined to E; and A2, destined to F. B sends two packets as well: B1, destined to F, and B2, destined to E. When C receives

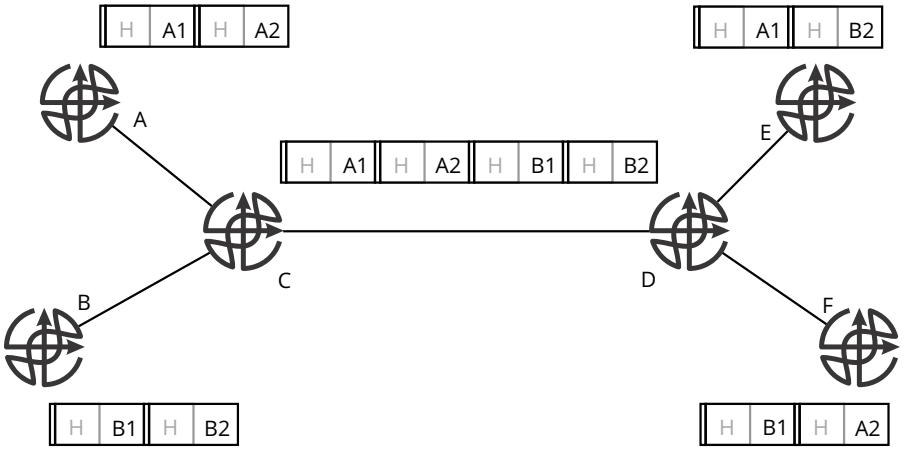


Figure 1-4 *Packet Switched Network*

these packets, it reads a small part of the packet header, often called a field, to determine the destination. C then consults a local table to determine which outbound interface the packet should be transmitted on. D does likewise, forwarding the packet out the correct interface toward the destination.

This way of forwarding traffic is called hop-by-hop forwarding, because each device in the network makes a completely independent decision about where to forward each individual packet. The local table each device consults is called a forwarding table; this normally is not one table, but many tables, potentially including a Routing Information Base (RIB) and a Forwarding Information Base (FIB).

> **Note**
>
> These tables, how they are built, and how they are used, are explained more fully in Chapter 7, "Packet Switching."

In the original circuit switched systems, the control plane is completely separate from packet forwarding through the network. With the move from circuit to packet switched, there was a corresponding move from centralized controller decisions to a distributed protocol running over the network itself. For the latter, each node is capable of making its own forwarding decisions locally. Each device in the network runs the distributed protocol to gain the information needed to build these local tables. This model is called a distributed control plane; thus the idea of a control plane was simply transferred from one model to the other, although they do not actually mean the same thing.

Note

Packet switching networks can use a centralized control plane, and circuit switching networks can use distributed control planes. At the time packet switched networks were first designed and deployed, however, they typically used distributed control planes. Software-Defined Networks (SDNs) brought the concept of centralized control planes back into the world of packet switched networks.

The first advantage the packet switched network has over a circuit switched network is the hop-by-hop forwarding paradigm. As each device can make a completely independent forwarding decision, packets can be dynamically forwarded around changes in the network topology, eliminating the need to communicate to the controller and await a decision. So long as there are at least two paths between the source and the destination (the network is two connected), packets handed to the network by the source will eventually be handed to the destination by the network.

The second advantage the packet switched network has over a circuit switched network is the way the packet switched network uses bandwidth. In the circuit switched network, if a particular circuit (really a time slot in the TDM example given) is not used, then the slot is simply wasted. In hop-by-hop forwarding, each device can best use the bandwidth available on each outbound link to carry the necessary traffic load. While this is *locally* more complex, it is *globally* simpler, and it makes better use of network resources.

The primary disadvantage of packet switched networks is the additional complexity required, particularly in the forwarding process. Each device must be able to read the packet header, look up the destination in a table, and then forward the information based on the table lookup results. In early hardware, these were difficult, time-consuming tasks; circuit switching was generally faster than packet switching. As hardware has improved over time, the speed of switching a variable length packet is generally close enough to the speed of switching a fixed length packet that there is little difference between packet and circuit switching.

Flow Control in Packet Switched Networks

In a circuit switched network, the controller allocates a specific amount of bandwidth to each circuit by assigning time slots from the source to the destination. What happens if the transmitter wants to send more traffic than the allocated time slots will support? The answer is simple—*it cannot*. In a sense, then, the ability to control the flow of packets through the network is built in to a circuit switched network;

there is no way to send more traffic than the network can forward, because "space" the transmitter has at its disposal for information sending is pre-allocated.

What about packet switched networks? If all the links in the network shown in Figure 1-4 have the same link speed, what happens if both A and B want to use the entire link capacity toward C? How will C decide how to send it all to D on a link that is half as large as the traffic it needs to handle? Here is where traffic flow control techniques can be used. Typically, they are implemented as a separate protocol/rule set "riding on top of" the underlying network, helping to "organize" packet transmission by building a virtual circuit between the two communicating devices.

Note

Flow and error control are discussed in detail in Chapter 2, "Data Transport Problems and Solutions."

The Transmission Control Protocol (TCP) provides flow control for Internet Protocol (IP) based packet switched networks. This protocol was first specified in 1973 by Vint Cerf and Bob Kahn.

The Protocol Wars

In the development of packet switched networks, a number of different protocols (or protocol stacks) were developed. Over time, all of them have been abandoned in favor of the IP suite of protocols. For instance, Banyan Vines had its own protocol suite based on IP called Vines Internet Protocol (VIP), and Novell Netware had its own protocol suite based on a protocol called IPX. Other standards bodies created standard protocol suites as well, such as the International Telecommunications Union's (ITU) suite of protocols built around Connectionless Mode Network Service (CLNS).

Why did all of these protocol suites fall by the wayside? Some of them were proprietary, and many governments and large organizations rejected proprietary solutions to packet switched networking for a wide range of reasons. The proprietary solutions were often not as well thought out, either, as they were generally developed and maintained by a small group of people. Standards-based protocols can be more complex, but they also tend to be developed and maintained by a larger group of experienced engineers. The protocol suite based on CLNS was a serious contender for some time, but it just never really caught on in the global Internet, which was becoming an important economic force at the time.

There were some specific technical reasons for this—for instance, CLNS does not number *wires,* but hosts. The aggregation of reachability information (concepts covered in more detail later in this book) is therefore limited in many ways.

An interesting reference for the discussion between the CLNS and IP protocol suites is *The Elements of Networking Style.*[1]

1. Padlipsky, *The Elements of Networking Style and Other Essays and Animadversions on the Art of Intercomputer Networking* (New York: Prentice-Hall, 1985).

Fixed Versus Variable Length Frames

In the late 1980s, a new topic of discussion washed over the network engineering world—Asynchronous Transfer Mode (ATM). The need for ever higher speed circuits, combined with slow progress in switching packets individually based on their destination addresses, led to a push for a new form of transport that would, eventually, reconfigure the entire set (or stack, because each protocol forms a layer on top of the protocol below, like a "stacked cake") of protocols used in modern networks. ATM combined the fixed length cell (or packet) size of circuit switching with a header from packet switching (although greatly simplified) to produce an "in between" technology solution. There were two key points to ATM: label switching and fixed call sizes; Figure 1-5 illustrates the first.

In Figure 1-5, G sends a packet destined to H. On receiving this packet, A examines a local table, and finds the next hop toward H is C. A's local table also specifies a

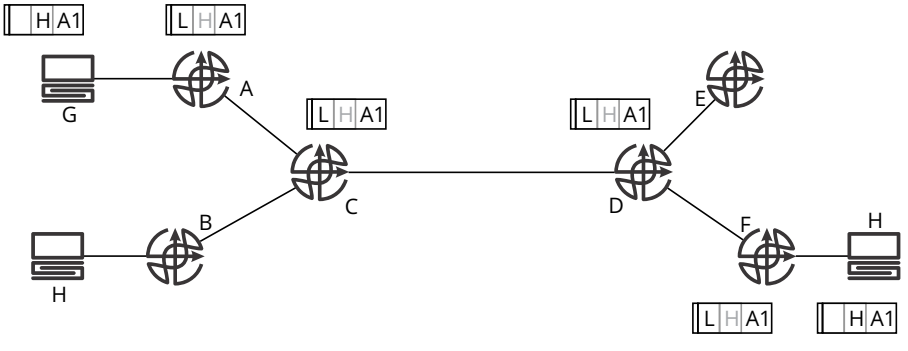


Figure 1-5 *Label Switching*

label, shown as L, rather than "just" information about where to forward the packet. A inserts this label into a dedicated field at the head of the packet and forwards it to C. When C receives the packet, it does not need to read the destination address in the header; rather, it just reads the label, which is a short, fixed length field. The label is looked up in a local table, which tells C to forward traffic to D for destination H. The label is very small, and is therefore easy to process for the forwarding devices, making switching much faster.

The label can also "contain" handling information for the packet, in a sense. For instance, if there are actually two streams of traffic between G and H, each one can be assigned a different label (or set of labels) through the network. Packets carrying one label can be given priority over packets carrying another label, so the network devices do not need to look at any fields in the header to determine how to process a particular packet.

This can be seen as a compromise between packet and circuit switching. While each packet is still forwarded hop by hop, a virtual circuit can also be defined by the label path through the network. The second point was that ATM was also based on a fixed sized cell: each packet was limited to 53 octets of information. Fixed size cells may seem to be a minor issue, but fixed size packets can make a huge performance difference. Figure 1-6 illustrates some factors involved in fixed cell sizes.

In Figure 1-6, packet 1 (A1) is copied from the network into memory on a line card or interface, *LC1;* then it travels across the internal fabric inside B (between memory locations) to *LC2*, being finally placed back onto the network at B's outbound interface. It might seem trivial from such a diagram, but perhaps the most

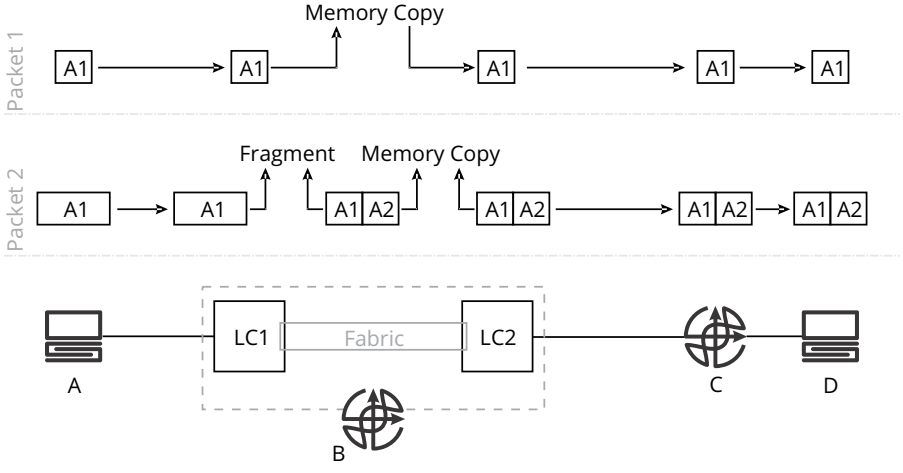


Figure 1-6 *Fixed Cell Sizes*

important factor in the speed at which a device can switch/process packets is the time it takes to copy the packet across any internal paths between memory locations. The process of copying information from one place in memory to another is one of the slowest operations a device can undertake, particularly on older processors. Making every packet the same (a fixed cell size) allowed code optimizations around the copy process, dramatically increasing switching speed.

Note

The process of switching a packet across an internal fabric is considered in Chapter 7, "Packet Switching."

Packet 2's path through B is even worse from a performance perspective; it is copied off the network into local memory first. When the destination port is determined by looking in the local forwarding table, the code processing the packet realizes the packet must be fragmented to fit into the largest packet size allowed on the outbound [B,C] link. The inbound line card, *LC1,* fragments the packet into *A1* and *A2,* creating a second header and adjusting any values in the header as needed. The packet is divided into two packets, A1 and A2. These two packets are copied in two operations across the fabric to the outbound line card, *LC2*. By using fixed size cells, ATM avoids the performance cost of fragmenting packets (at the time ATM was being proposed) incurred by almost every other packet switching system.

ATM did not, in fact, start at the network core and work its way to the network edge. Why not? The first answer lies in the rather strange choice of cell size. Why 53 octets? The answer is simple—and perhaps a little astounding. ATM was supposed to replace not only packet switched networks, but also the then-current generation of voice networks based on circuit switched technologies. In unifying these two technologies, providers could offer both sorts of services on a single set of circuits and devices.

What amount of information, or packet size, is ideal for carrying voice traffic? Around 48 octets. What amount of information, or packet size, is the minimum that makes any sort of sense for data transmission? Around 64 octets. Fifty-three octets was chosen as a compromise between these two sizes; it would not be perfect for voice transmission, as 5 octets of every cell carrying voice would be wasted. It would not be perfect for data traffic, because the most common packet size, 64 octets, would need to be split into two cells to be carried across an ATM network. A common line of thinking, at the time these deliberations were being held, was the data transport protocols would be able to adjust to the slightly smaller cell size, hence making 53 octets an optimal size to support a wide variety of traffic. The data transport protocols, however, did not adjust. To transport a 64-octet block of

data, one cell would contain 53 octets, and the second would contain 9 octets, with 42 octets of empty space. Providers discovered 50% or more of the bandwidth available on ATM links was consumed by empty cells—effectively wasted bandwidth. Hence, data providers stopped deploying ATM, voice providers never really started deploying it, and ATM died.

What is interesting is how the legacy of projects like ATM live on in other protocols and ideas. The label switching concept was picked up by Yakov Rekhter and other engineers, and developed into *label switching*. This keeps many of the fundamental advantages of ATM's quick lookup in the forwarding path, and bundling the metadata about packet handling into the label itself. Label switching eventually became Multiprotocol Label Switching (MPLS), which not only provides faster lookup, but also stacks of labels and virtualization. The basic idea was thus taken and expanded, impacting modern network protocols and designs in significant ways.

Note

MPLS is discussed in Chapter 9, "Network Virtualization."

The second legacy of ATM is the fixed cell size. For many years, the dominant network transport suite, based on TCP and IP, has allowed network devices to fragment packets while forwarding them. This is a well-known way to degrade the performance of a network, however. A *do not fragment* bit was added to the IP header, telling network devices to drop packets rather than fragmenting them, and serious efforts were put into discovering the largest packet that can be transmitted through the network between any pair of devices. A newer generation of IP, called IPv6, removed fragmentation by network devices from the protocol specification.

Calculating Loop-Free Paths

Overlapping many of these previous discussions within the network engineering world was another issue that often made it more difficult to decide whether packet or circuit switching was the better solution. How should loop-free paths be computed in a packet switched network?

As packet switched networks have, throughout the history of network engineering, been associated with distributed control planes, and circuit switched networks have been associated with centralized control planes, the issue of computing loop-free paths efficiently had a major impact on deciding whether packet switched networks were viable or not.

Note

Loop-free paths are discussed in Part II, "The Control Plane."

In the early days of network engineering, the available processing power, memory, and bandwidth were often in short supply. Table 1-1 provides a little historical context.

Table 1-1 *History of Computing Power, Memory, and Bandwidth*

Year	MIPS	Memory (Cost/MB)	Bandwidth (LAN)
1984	3.2 (Motorola 68010)	1331	2Mb/s
1987	6.6 (Motorola 68020)	154	10Mb/s
1990	44 (Motorola 68040)	98	16Mb/s
1996	541 (Intel Pentium Pro)	8	100Mb/s
1999	2,054 (Intel Pentium III)	1	100Mbp/s
2006	49,161 (Intel Core 2, 4 cores)	0.1	4Gb/s
2014	238,310 (Intel i7, 4 cores)	0.001	100Gb/s

In 1984, when many of these discussions were occurring, any difference in the amount of processor and memory between two ways of calculating loop-free paths through a network would have a material impact on the cost of building a network. When bandwidth is at a premium, reducing the number of bits a control plane required to transfer the information required to calculate a set of loop-free paths through a network makes a real difference in the amount of user traffic the network can handle. Reducing the number of bits required for the control to operate also makes a large difference in the stability of the network at lower bandwidths.

For instance, using a Type Length Vector (TLV) format to describe control plane information carried across the network adds a few octets of information to the overall packet length—but in the context of a 2Mbps link, aggravated by a chatty control plane, the costs could far outweigh the longer-term advantage of protocol extensibility.

Note

TLVs are discussed in Chapter 2, "Data Transport Problems and Solutions."

The protocol wars were rather heated at some points; entire research projects were undertaken, and papers written, about why and how one protocol was better than another. As an example of the kind of back and forth these arguments generated, a shirt seen at the Internet Engineering Task Force (IETF) during which the Open Shortest Path First (OSPF) Protocol was being developed said: *IS-IS = 0*. The "IS-IS" here refers to *Intermediate System-to-Intermediate System*, a control plane (routing protocol) originally developed by the International Organization for Standardization (ISO).

There was a wide variety of mechanisms proposed to solve the problems of calculating loop-free paths through a network; ultimately three general classes of solutions have been widely deployed and used:

- **Distance Vector** protocols, which calculate loop-free paths hop by hop based on the path cost
- **Link State** protocols, which calculate loop-free paths across a database synchronized across the network devices
- **Path Vector** protocols, which calculate loop-free paths hop by hop based on a record of previous hops

The discussion over which protocol is best for each specific network, and for what particular reasons, still persists; it is probably a never-ending conversation, as there is (probably) no final answer to the question. Instead, as with fitting a network to a business, there will probably always be some degree of art (or craft) involved in making a particular control plane work on a particular network. Much of the urgency in the question, however, has been drawn out by the increasing speed of networks—in processing power, memory, and bandwidth.

Quality of Service

As real-time traffic started to be carried over packet switched networks, QoS became a major problem. Voice and video both rely on the network being able to carry traffic between hosts quickly (having low delay), and with small amounts of variability in interpacket spacing (jitter). Discussions around QoS actually began in the early days of packet switched networking, but reached a high point around the time ATM was being considered. In fact, one of the main advantages of ATM was the ability to closely control the way in which packets were handled as they were carried over a packet switched network. With the failure of ATM in the

market, two distinct lines of thought emerged about applications that require strong controls on *jitter* and *delay*:

- These applications would never work on packet switched networks; these kinds of applications would always need to be run on a separate network.
- It is just a matter of finding the right set of QoS controls to allow such applications to run on packet switched networks.

Note

Quality of Service is discussed in detail in Chapter 8, "Quality of Service."

The primary application most providers and engineers were concerned about was voice, and the fundamental question came down to this: is it possible to provide decent voice over a network also carrying large file transfers and other "non-real-time" traffic? Complex schemes were invented to allow packets to be classified and marked (called QoS marking) so network devices would know how to handle them properly. Mapping systems were developed to carry these QoS markings from one type of network to another, and a lot of time and effort were put into researching queueing mechanisms—the order in which packets are sent out on an interface. Figure 1-7 shows a sample chart of one QoS system and the mapping between applications and QoS markings will suffice to illustrate the complexity of these systems.

The increasing link speeds, shown previously in Table 1-1, had two effects on the discussion around QoS:

- Faster links will (obviously) carry more data. As any individual voice and video stream becomes a shrinking part of the overall bandwidth usage, the need to strongly balance the use of bandwidth between different applications became less important.
- The amount of time required to move a packet from memory onto the wire through a physical chip is reduced with each increase in bandwidth.

As available bandwidth increased, the need for complex queueing strategies to counter jitter became less important. This increase in speed has been augmented by newer queueing systems that are much more effective at managing different kinds of traffic, reducing the necessity of marking and handling traffic in a fine-grained fashion.

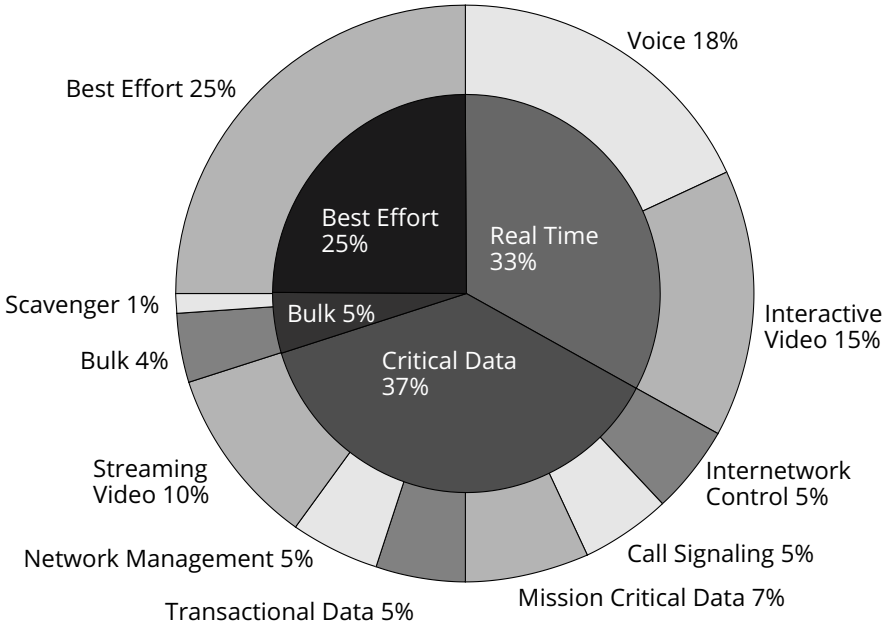


Figure 1-7 *QoS Planning and Mapping*

These increases in bandwidth were often enabled by changing from copper to glass fiber. Fiber not only offers larger bandwidths but also more reliable transmission of data. The way physical links are built also evolved, making them more resistant to breakage and other material problems. A second factor increasing bandwidth availability was the growth of the Internet. As networks became more common and more connected, a single link failure had a lesser impact on the amount of available bandwidth and on the traffic flows across the network.

As processors became faster, it became possible to develop systems where dropped and delayed packets would have less effect on the quality of a real-time stream. Increasing processor speeds also made it possible to use very effective compression algorithms, reducing the size of each stream. On the network side, faster processors meant the control plane could compute a set of loop-free paths through the network faster, reducing both direct and indirect impacts of link and device failures.

Ultimately, although QoS is still important, it can be much simplified. Four to six queues are often enough to support even the most difficult applications. If more are needed, some systems can now either engineer traffic flows through a network or

actively manage queues, to balance between the complexity of queue management and application support.

The Revenge of Centralized Control Planes

In the 1990s, in order to resolve many of the perceived problems with packet switched networks, such as complex control planes and QoS management, researchers began working on a concept called *Active Networking*. The general idea was that the control plane for a packet switched network could, and should, be separated from the forwarding devices in order to allow the network to interact with the applications running on top of it.

The basic concept of separating the control and data planes more distinctly in packet switching networks was again considered in the formation of the Forwarding and Control Element Separation (ForCES) working group in the IETF. This working group was primarily concerned with creating an interface applications can use to install forwarding information onto network devices. The working group was eventually shut down in 2015 and its standards were never widely implemented.

In 2006, researchers began looking for a way to experiment with control planes in packet switched networks without the need to code modifications on the devices themselves—a particular problem, as most of these devices were sold by vendors as unmodifiable appliances (or black boxes). The eventual result was *OpenFlow,* a standard interface that allows applications to install entries directly in the forwarding table (rather than the routing table; this is explained more fully in several places in Part I of this book, "The Data Plane"). The research project was picked up as a feature by several vendors, and a wide array of controllers have been created by vendors and open source projects. Many engineers believed OpenFlow would revolutionize network engineering by centralizing the control plane.

The reality is likely to be far different—what is likely to happen is what has always happened in the world of data networking: the better parts of a centralized control plane will be consumed into existing systems, and the fully centralized model will fall to the wayside, leaving in its path changed ideas about how the control plane interacts with applications and the network at large.

Complexity

The technologies described thus far—circuit and packet switching, control planes, and QoS—are very complex. In fact, there appears to be no end to the increasing

complexity in networks, particularly as applications and businesses become more demanding. This section will consider two specific questions in relation to complexity and networks:

- What is network complexity?
- Can network complexity be "solved"?

The final parts of this section will consider a way of looking at complexity as a set of tradeoffs.

Why So Complex?

While the most obvious place to begin might be with a definition of complexity, it is actually more useful to consider why complexity is required in a more general sense. To put it more succinctly, is it possible to "solve" complexity? Why not just design simpler networks and protocols? Why does every attempt to make anything simpler in the networking world end up apparently making things more complex in the long run?

For instance, by tunneling on top of (or through) IP, the control plane's complexity is reduced, and the network is made simpler overall. Why then do tunneled overlays end up containing so much complexity?

There are two answers to this question. *First,* human nature being what it is, engineers will always invent ten different ways to solve the same problem. This is especially true in the virtual world, where new solutions are (relatively) easy to deploy, it is (relatively) easy to find a problem with the last set of proposed solutions, and it is (relatively) easy to move some bits around to create a new solution that is "better than the old one." This is particularly true from a vendor perspective, when building something new often means being able to sell an entirely new line of products and technologies—even if those technologies look very much like the old ones. The virtual space, in other words, is partially so messy because it is so easy to build something new there.

The *second* answer, however, lies in a more fundamental problem: complexity is necessary to deal with the uncertainty involved in difficult to solve problems. Figure 1-8 illustrates.

Adding complexity seems to allow a network to handle future requirements and unexpected events more easily, as well as provide more services over a smaller set of base functions. If this is the case, why not simply build a single protocol running on a single network able to handle all the requirements potentially thrown at it and can handle any sequence of events you can imagine? A single network running a single

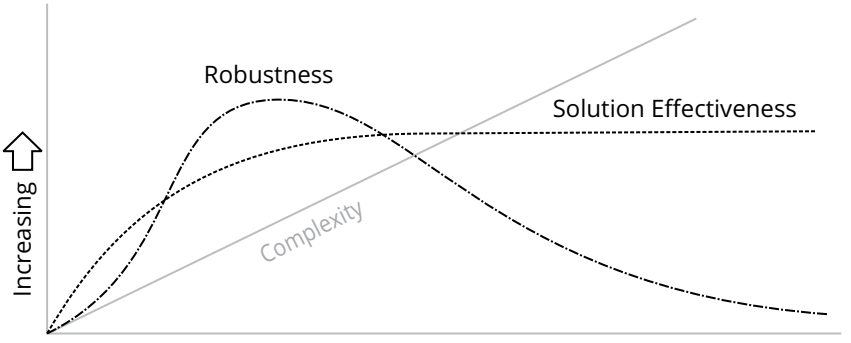


Figure 1-8 *Complexity, Effectiveness, and Robustness*

protocol would certainly reduce the number of moving parts network engineers need to deal with, making all our lives simpler, right? In fact, there are a number of different ways to manage complexity, for instance:

1. Abstract the complexity away, to build a black box around each part of the system, so each piece and the interactions between these pieces are more immediately understandable.
2. Toss the complexity over the cubicle wall—to move the problem out of the networking realm into the realm of applications, or coding, or a protocol. As RFC1925 says, "It is easier to move a problem around (e.g., by moving the problem to a different part of the overall network architecture) than it is to solve it."
3. Add another layer on top, to treat all the complexity as a black box by putting another protocol or tunnel on top of what's already there. Returning to RFC1925, "It is always possible to add another level of indirection."
4. Become overwhelmed with the complexity, label what exists as "legacy," and chase some new shiny thing perceived to be able to solve all the problems in a much less complex way.
5. Ignoring the problem and hoping it will go away. Arguing for an exception "just this once," so a particular business goal can be met, or some problem fixed, within a very tight schedule, with the promise that the complexity issue will be dealt with "later," is a good example.

Each of these solutions, however, has a set of tradeoffs to consider and manage. Further, at some point, any complex system becomes brittle—*robust yet fragile*. A system is robust yet fragile when it is able to react resiliently to an expected set of circumstances, but an unexpected set of circumstances will cause it to fail. To give an example from the real world—knife blades are required to have a somewhat unique combination of characteristics. They must be hard enough to hold an edge and cut, and yet flexible enough to bend slightly in use, returning to their original shape without any evidence of damage, and they must not shatter when dropped. It has taken years of research and experience to find the right metal to make a knife blade from, and there are still long and deeply technical discussions about which material is right for specific properties, under what conditions, etc.

> "Trying to make a network proof against predictable problems tends to make it fragile in dealing with unpredictable problems (through an ossification effect as you mentioned). Giving the same network the strongest possible ability to defend itself against unpredictable problems, it necessarily follows, means that it MUST NOT be too terribly robust against predictable problems. Not being too robust against predictable problems is necessary to avoid the ossification issue, but not necessarily sufficient to provide for a robust ability to handle unpredictable network problems." —Tony Przygienda

Complexity is necessary, then: it cannot be "solved."

Defining Complexity

Given complexity is necessary, engineers are going to need to learn to manage it in some way, by finding or building a model or framework. The best place to begin in building such a model is with the most fundamental question: What does complexity mean in terms of networks? Can you put a network on a scale and have the needle point to "complex"? Is there a mathematical model into which you can plug the configurations and topology of a set of network devices to produce a "complexity index"? How do the concepts of scale, resilience, brittleness, and elegance relate to complexity? The best place to begin in building a model is with an example.

Control Plane State versus Stretch

What is network stretch? In the simplest terms possible, it is the difference between the shortest path in a network and the path that traffic between two points actually takes. Figure 1-9 illustrates this concept.

Assuming the cost of each link in this network is 1, the shortest physical path between Routers A and C will also be the shortest logical path: [A,B,C]. What

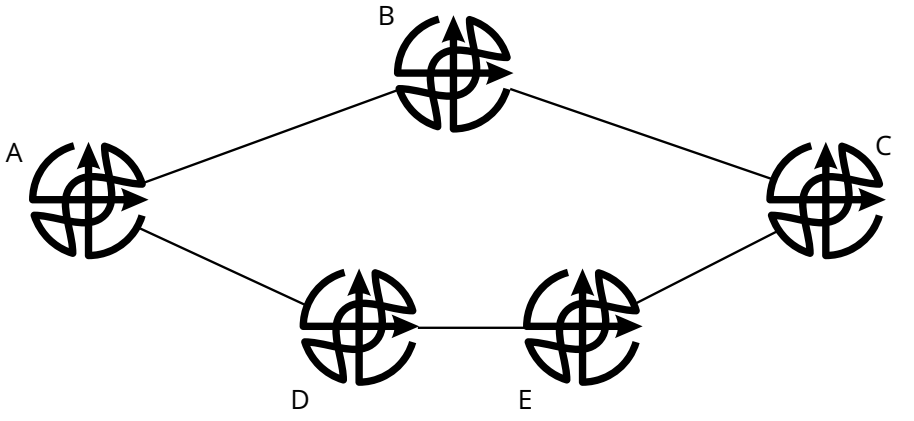


Figure 1-9 *A Small Network to Illustrate State and Stretch*

happens, however, if the metric on the [A,B] link is changed to 3? The shortest physical path is still [A,B,C], but the shortest logical path is now [A,D,E,C]. The differential between the shortest physical path and the shortest logical path is the distance a packet being forwarded between Routers A and C must travel—in this case, the stretch can be calculated as (4 [A,D,E,C])−(3 [A,B,C]), for a stretch of 1.

How Is Stretch Measured?

The way stretch is measured depends on what is most important in any given situation, but the most common way is by comparing hop counts through the network, as is used in the examples here. In some cases, it might be more important to consider the metric along two paths, the delay along two paths, or some other metric, but the important point is to measure it consistently across every possible path to allow for accurate comparison between paths.

It is sometimes difficult to differentiate between the physical topology and the logical topology. In this case, was the [A,B] link metric increased because the link is actually a slower link? If so, whether this is an example of stretch, or an example of simply bringing the logical topology in line with the physical topology is debatable.

In line with this observation, it is much easier to define policy in terms of stretch than almost any other way. Policy is any configuration that increases the stretch of a network. Using *Policy-Based Routing*, or *Traffic Engineering*, to push traffic off the shortest physical path and onto a longer logical path to reduce congestion on specific links, for instance, is a policy—it increases stretch.

Increasing stretch is not always a bad thing. Understanding the concept of stretch simply helps us understand various other concepts and put a framework around

complexity and optimization tradeoffs. The shortest path, physically speaking, is not always the best path.

Stretch, in this illustration, is very simple—it impacts every destination, and every packet flowing through the network. In the real world, things are more complex. Stretch is actually per source/destination pair, making it very difficult to measure on a network-wide basis.

Defining Complexity: A Model

Three components—state, optimization, and surface—are common in virtually every network or protocol design decision. These can be seen as a set of tradeoffs, as illustrated in Figure 1-10 and described in the list that follows.

- Increasing optimization always moves toward more state or more interaction surfaces.
- Decreasing state always moves toward less optimization or more interaction surfaces.
- Decreasing interaction surfaces always moves toward less optimization or more state.

These are no ironclad rules, of course; they are contingent on the specific network, protocols, and requirements, but they are generally true often enough to make this a useful model for understanding tradeoffs in complexity.

Interaction Surfaces

While state and optimization are fairly intuitive, it is worthwhile to spend just a moment more on interaction surfaces. The concept of interaction surfaces is difficult to grasp primarily because it covers such a wide array of ideas. Perhaps an example would be helpful; assume a function that

- Accepts two numbers as input
- Adds them
- Multiplies the resulting sum by 100
- Returns the result

This single function can be considered a subsystem in some larger system. Now assume you break this single function into two functions, one of which does the

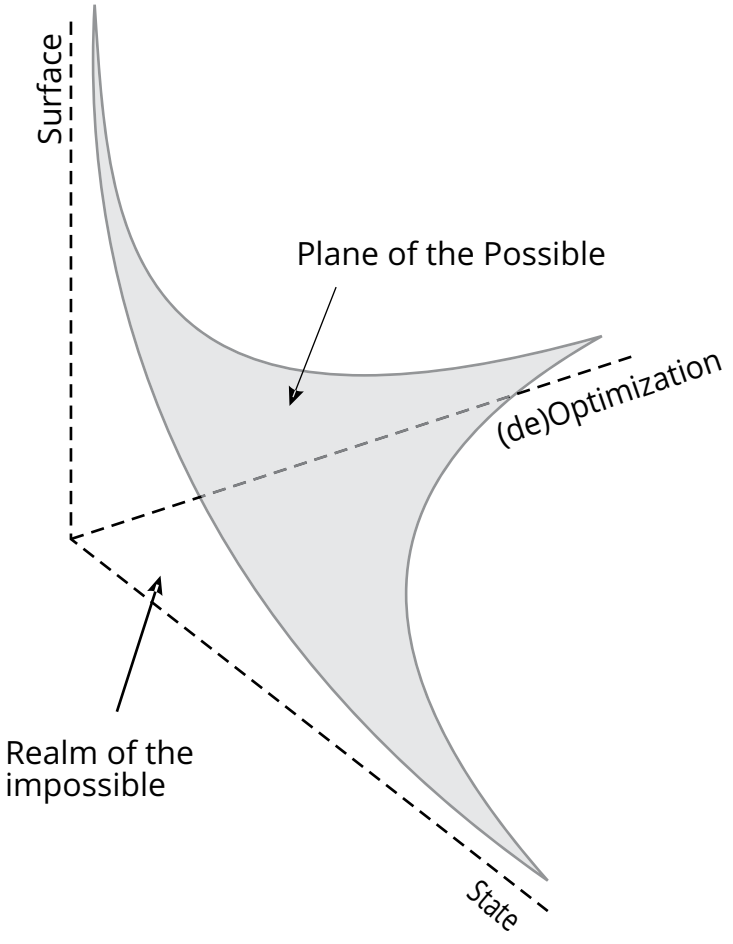


Figure 1-10 *The Plane of the Possible*

addition, and the other of which does the multiplication. You have created two simpler functions (each one only does one thing), but you have also created an interaction surface between the two functions—you have created two interacting subsystems within the system where there only used to be one.

As another example, assume you have two control planes running on a single network. One of these two control planes carries information about destinations reachable outside the network (external routes), while the other carries destinations reachable inside the network (internal routes). While these two control planes are

different systems, they will still interact in many interesting and complex ways. For instance, the reachability to an external destination will necessarily depend on reachability to the internal destinations between the edges of the network. These two control planes must now work together to build a complete table of information that can be used to forward packets through the network.

Even two routers communicating within a single control plane can be considered an interaction surface. This breadth of definition is what makes it so very difficult to define what an interaction surface is.

Interaction surfaces are not a bad thing; they help engineers and designers divide and conquer in any given problem space, from modeling to implementation. At the same time, interaction surfaces are all too easy to introduce without thought.

Managing Complexity through the Wasp Waist

The wasp waist, or hourglass model, is used throughout the natural world, and widely mimicked in the engineering world. While engineers do not often consciously apply this model, it is actually used all the time. Figure 1-11 illustrates the hourglass model in the context of the four-layer Department of Defense (DoD) model that gave rise to the Internet Protocol (IP) suite.

At the bottom layer, the physical transport system, there are a wide array of protocols, from Ethernet to Satellite. At the top layer, where information is marshaled and presented to applications, there is a wide array of protocols, from Hypertext Transfer Protocol (HTTP) to TELNET (and thousands of others besides). A funny thing happens when you move toward the middle of the stack, however: the number of protocols decreases, creating an hourglass. Why does this work to control complexity? Going back through the three components of complexity—state, surface, and complexity—exposes the relationship between the hourglass and complexity.

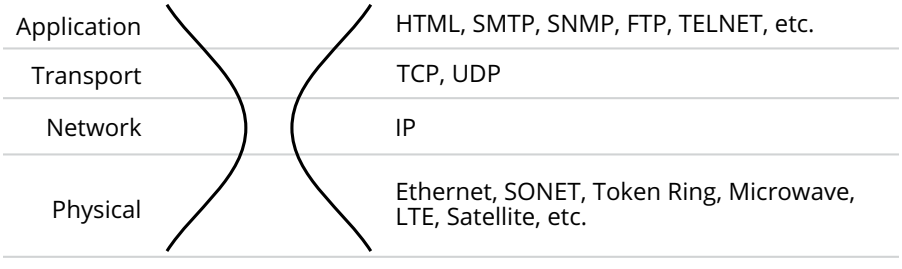


Figure 1-11 *The DoD Model and the Wasp Waist*

- State is divided by the hourglass into two distinct types of state: information about the network and information about the data being transported across the network. While the upper layers are concerned with marshaling and presenting information in a usable way, the lower layers are concerned with discovering what connectivity exists and what the connectivity properties actually are. The lower layers do not need to know how to format an FTP frame, and the upper layers do not need to know how to carry a packet over Ethernet—state is reduced at both ends of the model.
- Surfaces are controlled by reducing the number of interaction points between the various components to precisely one—the Internet Protocol (IP). This single interaction point can be well defined through a standards process, with changes in the one interaction point closely regulated to prevent massive rapid changes that will reflect up and down the protocol stack.
- Optimization is traded off by allowing one layer to reach into another layer, and by hiding the state of the network from the applications. For instance, TCP does not really know the state of the network other than what it can gather from local information. TCP could potentially be much more efficient in its use of network resources, but only at the cost of a layer violation, which opens up difficult-to-control interaction surfaces.

The layering of a stacked network model is, then, a direct attempt to control the complexity of the various interacting components of a network.

Complexity and Tradeoffs

A very basic law of complexity might be stated thus: in any complex system, there will exist sets of three-way tradeoffs. The State/Optimization/Surface (SOS) model described here is one set of such tradeoffs. Another one, more familiar to engineers who work primarily in databases, is Consistency/Accessibility/Partitioning (the CAP theorem). Yet another, often found in a wider range of contexts, is Quick/Cost/Quality (QSQ). These are not components of complexity, but what can be called the *consequents* of complexity. Engineers need to be adept at spotting these kinds of tradeoff triangles, accurately understanding the "corners" of the triangle, determining where along the *plane of the possible* the most optimal solution lies, and being able to articulate why some solutions simply are not possible or desirable.

If you have not found the tradeoffs, you have not looked hard enough is a good rule of thumb to follow in all engineering work.

Final Thoughts

This chapter is not intended to provide detail, but rather to frame key terms within the scope of the history of computer network technology. The computer networking world does not have a long history (for example, human history reaches back at least 6,000 years, and potentially many millions, depending on your point of view), but this history still contains a set of switchback turns and bumpy pathways, often making it difficult for the average person to understand how and why things work the way they do.

With this introduction in hand, it is time to turn to the first topic of interest in understanding how networks really work—the data plane.

Further Reading

Brewer, Eric. "Towards Robust Distributed Systems." Presented at the ACM Symposium on the Principles of Distributed Computing, July 19, 2000. http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf.

Buckwalter, Jeff T. *Frame Relay: Technology and Practice*. 1st edition. Reading, MA: Addison-Wesley Professional, 1999.

Cerf, Vinton G., and Edward Cain. "The DoD Internet Architecture Model." *Computer Networks* 7 (1983): 307–18.

Gorrell, Mike. "Salt Lake County Data Breach Exposed Info of 14,200 People." *The Salt Lake Tribune*. Accessed April 23, 2017. http://www.sltrib.com/home/3705923-155/data-breach-exposed-info-of-14200.

Ibe, Oliver C. *Converged Network Architectures: Delivering Voice over IP, ATM, and Frame Relay*. 1st edition. New York: Wiley, 2001.

Kumar, Balaji. *Broadband Communications: A Professional's Guide to ATM, Frame Relay, SMDs, Sonet, and Bisbn*. New York: McGraw-Hill, 1995.

"LAN Emulation." *Microsoft TechNet*. Accessed August 4, 2017. https://technet.microsoft.com/en-us/library/cc976969.aspx.

"LAN Emulation (LANE)." *Cisco*. Accessed August 4, 2017. http://www.cisco.com/c/en/us/tech/asynchronous-transfer-mode-atm/lan-emulation-lane/index.html.

Padlipsky, Michael A. *The Elements of Networking Style and Other Essays and Animadversions on the Art of Intercomputer Networking*. Prentice-Hall, 1985.

Russell, Andrew L. "OSI: The Internet That Wasn't." Professional Organization. *IEEE Spectrum*, September 27, 2016. https://spectrum.ieee.org/tech-history/cyberspace/osi-the-internet-that-wasnt

"Understanding the CBR Service Category for ATM VCs." *Cisco*. Accessed June 10, 2017. http://www.cisco.com/c/en/us/support/docs/asynchronous-transfer-mode-atm/atm-traffic-management/10422-cbr.html.

White, Russ, and Jeff Tantsura. *Navigating Network Complexity: Next-Generation Routing with SDN, Service Virtualization, and Service Chaining*. Indianapolis, IN: Addison-Wesley Professional, 2015.

Review Questions

1. One specific realm where different business assumptions can be clearly seen is in choosing to use a small number of large network devices (such as a chassis-based router that supports multiple line cards) or using a larger number of smaller devices (so-called *pizza box*, or *one rack unit*, routers having a fixed number of interfaces available) to build a campus or data center network. List a number of different factors that might make one option more expensive than the other, and then explain what sorts of business conditions might dictate the use of one instead of the other *for both options*.

2. One "outside representation" of code bloat in software applications is *nerd knobs;* while there are many definitions of a nerd knob, they are generally considered a configuration command that will modify some small, specific, point of operation in the way a protocol or device operates. There are actually some research papers and online discussions around the harm from nerd knobs; you can also find command sets from various network devices across a number of software releases through many years. In order to see the growth in complexity in network devices, trace the number of available commands, and try to judge how many of these would be considered nerd knobs versus major features. Is there anything you can glean from this information?

3. TDM is not the only kind of multiplexing available; there is also Frequency Division Multiplexing (FDM). Would FDM be useful for dividing a channel in the same way that TDM is? Why or why not?

4. What is an inverse multiplexer, and what would it be used for?

5. Read the two references to ATM LAN Emulation (LANE), in the "Further Reading" section. Describe the complexity in this solution from within the complexity model; where are state and interaction surfaces added, and what sort of optimization is being gained with each addition? Do you think the ATM LANE solution presents a good set of tradeoffs for providing the kinds of services it is designed to offer versus something like a shared Ethernet network?

6. Describe, in human terms, why delay and jitter are bad in real time (interactive) voice and video communications. Would these same problems apply to recorded voice and video stored and played back at some later time? Why or why not?
7. How would real-time (interactive) voice and video use the network differently than a large file transfer? Are there specific points at which you can compare the two kinds of traffic, and describe how the network might need to react differently to each traffic type?
8. The text claims the "wasp waist" is a common strategy used in nature to manage complexity. Find several examples in nature. Research at least one other set of protocols (protocol stack) than TCP/IP, such as Banyan Vines, Novell's IPX, or the OSI system. Is there a "wasp waist" in these sets of protocols, as well? What is it?
9. Are there wasp waists in other areas of computing, such as the operating systems used in personal computers, or mobile computing devices (such as tablets and mobile phones)? Can you identify them?
10. Research some of the arguments against removing fragmentation from the Internet Protocol in IPv6. Summarize the points made by each side. Do you agree with the final decision to remove fragmentation?

Chapter 2

Data Transport Problems and Solutions

Learning Objectives

After reading this chapter, you should be able to:

- Understand the concept of marshaling data, different marshaling options, and the tradeoffs between them
- Understand the concepts of dictionaries, grammars, and metadata within the context of data marshaling
- Understand the concepts of fixed length fields, type length values, and shared data dictionaries
- Understand the difference between error detection and error correction
- Understand the fundamental concepts of checking for errors in data transmission
- Understand the relationship between addressing and multiplexing
- Understand the basic theory behind multicast and anycast
- Understand flow control mechanisms, including windowed flow control

When transport protocols dream, do they dream of applications? They probably should, as the primary purpose of a network is to support applications—and the primary resource that applications need from a network is data moved from one process (or processor) to another. But how can data be transmitted over a wire, or through the air, or over an optical cable?

Perhaps it is best to begin with a more familiar example: human language. The authors of this book wrote it using formatting, language, and vocabulary enabling you to read and understand the information presented. What problems does a language need to overcome to make the communication, this writing and reading, possible?

Thoughts must be captured in a form that allows them to be retrieved by a receiver. In human languages, information is packaged into words, sentences, paragraphs, chapters, and books. Each level of this division implies some unit of information, and some organizational system. For instance, sounds or ideas are encapsulated into letters or symbols; sounds or ideas are then combined into words; words are combined into sentences, etc. Sentences follow a particular grammatical form so you can decode the meaning from the symbols. This encoding of thoughts and information into symbols formatted which allows a reader (receiver) to retrieve the original meaning will be called marshaling the data in this book.

One aspect of marshaling is *definitional*—the process of associating one set of symbols to a particular meaning. Metadata, or data about the data, allows you to understand how to interpret information in a flow or stream.

There must be some way of managing errors in transmission or reception. Suppose you have a pet dog who likes to chase after a particular ball. The ball drops out of a basket one day, and bounces into the street. The dog chases and appears to be heading directly into the path of an oncoming car. What do you do? Perhaps you shout "Stop!"—and then maybe "No!"—and perhaps "Stay!" Using several commands that should result in the same action—the dog stopping before he runs into the street—is making certain the dog has correctly received, and understood, the message. Shouting multiple messages will, you hope, ensure there is no misunderstanding in what you are telling the dog to do.

This is, in fact, a form of error correction. There are many kinds of error correction built into human language. For instance, yu cn prbbly stll rd ths sntnce. Human languages overspecify the information they contain, so a few missed letters do not cause the entire message to be lost. This overspecification can be considered a form of forward error correction. This is not the only form of error correction human languages contain, however. They also contain questions, which can be asked to verify, validate, or gain missing bits or context of information previously "transmitted" through the language.

There must be some way to talk to one person, or a small group of people, using a single medium—air—within a larger crowd. It is not uncommon to need to talk to one person out of a room full of people. Human language has built in ways of dealing with this problem in many situations, such as calling someone's name, or speaking loudly enough to be heard by the person you are directly facing (the implementation of language can be *directional*, in other words). The ability to speak to one person among many, or a specific subset of people, is multiplexing.

Finally, there must be some way to control the flow of a conversation. With a book, this is a simple matter; the writer produces text in parts, which are then collected into a format the reader can read, and reread, at a completely different pace.

Not many people think of a book as a form of flow control, but putting thoughts into written form is an effective way to disconnect the speed of the sender (the speed of writing) from the speed of the receiver (the speed of reading). Spoken language has other forms of flow control, such as "um," and the glazed-over look in a listener's eyes when she has lost the line of reasoning a speaker is following, or even physical gestures indicating the speaker should slow down.

To summarize, successful communication systems need to solve four problems:

- Marshaling the data; converting ideas into symbols and a grammar the receiver will understand
- Managing errors, so the ideas are correctly transmitted from the sender to the receiver
- Multiplexing, or allowing a common media or infrastructure to be used for conversations between many different pairs of senders and receivers
- Flow control, or the ability to make certain the receiver is actually receiving and processing the information before the sender transmits more

The following sections examine each of these problems as well as some of the solutions available in each problem space.

Digital Grammars and Marshaling

Consider the process you are using to read this book. You examine a set of marks created to contrast with a physical carrier, ink on paper. These marks represent certain symbols (or, if you are hearing this book, certain sounds on a white noise background), which you then interpret as letters. These letters, in turn, you can put together using rules of spacing and layout to form words. Words, through punctuation and spacing, you can form into sentences.

At each stage in the process there are several kinds of things interacting:

- A physical carrier onto which the signal can be imposed. This work of representing information against a carrier is grounded in the work of Claude Shannon, and is outside the scope of this book; further reading is suggested in the following section for those who are interested.
- A symbolic representation of units of information used to translate the physical symbols into the first layer of logical content. When you are interpreting symbols, two things are required: a dictionary, which describes the range of possible logical symbols that can correspond to a certain physical state, and a grammar, which describes how to determine which logical symbol relates to this instance of physical state. These two things, combined, can be described as a protocol.

- A way to convert the symbols into words and then the words into sentences. Again, this will consist of two components, a dictionary and a grammar. Again, these can be described as protocols.

As you move "up the stack," from the physical to the letters to the words to the sentences, etc., the dictionary will become less important, and the grammar, which allows you to convert the context into meaning, more important—but these two things exist at every layer of the reading and/or listening process. The dictionary and grammar are considered two different forms of metadata you can use to turn physical representations into sentences, thoughts, lines of argument, etc.

Digital Grammars and Dictionaries

There really is not much difference between a human language, such as the one you are reading right now, and a digital language. A digital language is not called a *language,* however; it is called a protocol. More formally:

A protocol is a dictionary and a grammar (metadata) used to translate one kind of information into another.

Protocols do not work in just one direction, of course; they can be used to encode as well as decode information. Languages are probably the most common form of protocol you encounter on a daily basis, but there are many others, such as traffic signs; the user interfaces on your toaster, computer, and mobile devices; and every human language.

Given you are developing a protocol, which primarily means developing a dictionary and a grammar, there are two kinds of optimization you can work toward:

- **Resource Efficiency.** How many resources are used in encoding any particular bit of information? The more metadata included inline, with the data itself, the more efficient the encoding will be—but the more implementations will rely on dictionaries to decode the information. Protocols that use very small signals to encode a lot of information are generally considered compact.
- **Flexibility.** In the real world, things change. Protocols must somehow be designed to deal with change, hopefully in a way not requiring a "flag day" to upgrade the protocol.

The metadata tradeoff is one of many you will find in network engineering; either include more metadata, allowing the protocol to better handle future requirements, or include less metadata, making the protocol more efficient and compact. A good rule of thumb, one you will see repeated many times throughout this book, is: *if you have not found the tradeoff, you have not looked hard enough*.

What Is a Flag Day?

If you need to switch from one version of a protocol that is installed and running on many computers to a newer version of the same protocol, or perhaps even a different protocol, you have three choices.

First, you can design the protocol (or protocols) so the old and new versions can overlap, or run on the same network at the same time. This is sometimes called the ships in the night solution; the old and new protocols (or versions of the same protocol) do not interact at all.

Second, you can pick a day (and potentially a time, down to the millisecond in some cases) to switch from the old protocol to the new. This is called a flag day. How did this term become attached to this kind of protocol changeover event? In 1966, every system running the Multics operating system needed to be switched from one definition of characters to another, specifically to replace ASCII 1965 with ASCII 1967. A holiday was chosen for the change, which would give all the Multics system administrators a full day to replace their software and have the systems operating for the first business day after the change. The day chosen was *Flag Day* in the United States, June 14, 1966. Hence the term *flag day* become forever associated with a change requiring every host in the system to be rebooted at (roughly) the same time to ensure proper operation.[1]

The most famous flag day is the transition from the Network Control Program (NCP) transport protocol to the Transmission Control Protocol (TCP) on the *entire Internet* (as it existed then) in 1983. The process of moving from one to the other required an Internet Engineering Task Force (IETF) Request for Comment (RFC) to describe and coordinate the process—RFC801.[2]

Third, you can design the protocol so a single version can contain multiple versions of the same information, with each version being formatted differently. Senders can send in either format, and receivers should be able to interpret the data in either format. When all the systems have been upgraded to the newer version of the software, the older encoding can be replaced. This mechanism relies heavily on a principle laid out in RFC760:

> In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).[3]

1. "Flag Day."
2. Postel, *NCP/TCP Transition Plan*.
3. *Internet Protocol* This quote, or something similar, is attributed to Jon Postel.

A dictionary in a protocol is a table of digital patterns to symbols and operations. Perhaps the most commonly used digital dictionaries are character codes. Table 2-1 replicates part of the Unicode character code dictionary.

Table 2-1 *A Partial Unicode Dictionary or Table*

Code	Glyph	Decimal	Description	#
U+0030	0	&#048;	Digit Zero	0017
U+0031	1	&#049;	Digit One	0018
U+0032	2	&#050;	Digit Two	0019
U+0033	3	&#051;	Digit Three	0020
U+0034	4	&#052;	Digit Four	0021
U+0035	5	&#053;	Digit Five	0022
U+0036	6	&#054;	Digit Six	0023
U+0037	7	&#055;	Digit Seven	0024
U+0038	8	&#056;	Digit Eight	0025
U+0039	9	&#057;	Digit Nine	0026
U+003A	:	&#058;	Colon	0027
U+003B	;	&#059;	Semicolon	0028
U+003C	<	&#060;	Less-than sign	0029

Using Table 2-1, if a computer is "reading" an array representing a series of letters, it will print out (or treat in processing) the number *6* if the number in the array is 0023, the number *7* if the number in the array is 0024, etc. This table, or dictionary, relates specific numbers to specific symbols in an alphabet, just like a dictionary relates a word to a range of meanings.

How can the computer determine the difference between the price of a banana and the letters in the word *banana*? Through the context of the information. For instance, perhaps the array in question is stored as a string, or a series of letters; the array being stored as a string variable type provides the metadata, or the context, which indicates the values in these particular memory locations should be treated as letters rather than the numeric values contained in the array. This metadata, acted on by the computer, provides the grammar of the protocol.

In protocols, dictionaries are often expressed in terms of what any particular field in a packet contains, and grammars are often expressed in terms of how the packet is built, or what fields are contained at what locations in a packet.

There are several ways to build dictionaries and basic (first-level) grammars; several of these will be considered in the following sections.

Fixed Length Fields

Fixed length fields are the simplest of the dictionary mechanisms to explain. The protocol defines a set of fields, what kind of data each field contains, and how large each field is. This information is "baked into" the protocol definition, so every implementation is built to these same specifications, and hence can interoperate with one another. Figure 2-1 illustrates a fixed length field encoding used in the Open Shortest Path First (OSPF) protocol taken from RFC2328.[4]

The row of numbers across the top of Figure 2-1 indicates the individual bits in the packet format; each row contains 32 bits of information. The first 8 bits indicate the version number, the second 8 bits always have the number 5, the following 16 bits contain the total packet length, etc. Each of these fields is further defined in the protocol specification with the kind of information carried in the field and how it is encoded. For instance:

- The version number field is encoded as an unsigned integer. This is metadata indicating the dictionary and grammar used for this packet. If the packet format needs to be changed, the version number can be increased, allowing transmitters and receivers to use the correct dictionary and grammar when encoding and decoding the information in the packet.
- The number 5 indicates the kind of packet within the protocol; this is part of a dictionary defined elsewhere in the standards document, so it is simply inserted as a fixed value in this illustration. This particular packet is a *Link State Acknowledgment Packet*.

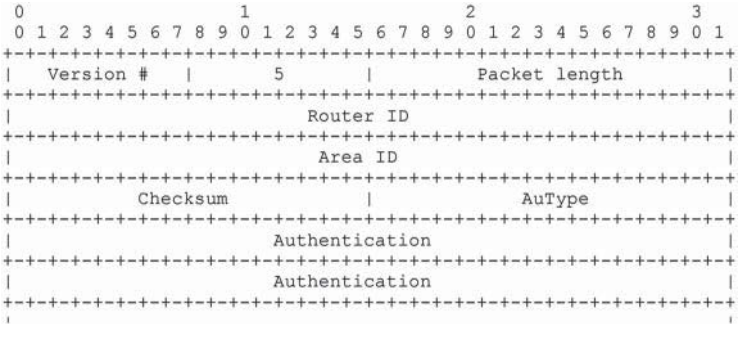


Figure 2-1 *OSPF Fixed Length Field Definition in the Protocol Specification*

4. Moy, *OSPF Version 2*, 201.

- The packet length is encoded as an unsigned integer indicating the number of octets (or sets of 8 bits) contained in the complete packet. This allows the packet size to vary in length depending on how much information needs to be carried.

The fixed length field format has several advantages. Primarily, the location of any piece of information within the packet will be the same from packet to packet, which means it is easy to optimize the code designed to encode and decode the information around the packet format. For instance, a common way of processing a fixed length packet format is to create an in-memory data structure matching the packet format precisely; when the packet is read off the wire, it is simply copied into this data structure. The fields within the packet can then be read directly.

Fixed length formats tend to be somewhat compact. The metadata needed to encode and decode the data is carried "outside the protocol," in the form of a protocol specification. The packets themselves contain only the value, and never any information about the values. On the other hand, fixed length formats can waste a lot of space by buffering the fields so they are always the same length. For instance, the decimal number *1* can be represented with a single binary digit (a single bit), while the decimal number *4* requires 3 binary digits (three bits); if a fixed length field must be able to represent any number between 0 and 4, it will need to be at least 3 bits long, even though two of those bits will sometimes be "wasted" in representing smaller decimal numbers.

Fixed length formats also often waste space by aligning the field sizes on common processor memory boundaries to improve the speed of processing. A field required to take values between 0 and 3, even though it only needs two bits to represent the full set of values, may be encoded as an 8-bit field (a full octet) in order to ensure the field following is always aligned on an octet boundary for faster in-memory processing.

Flexibility is where fixed length encoding often runs into problems. If some field is defined as an 8-bit value (a single octet) in the original specification, there is no obvious way to modify the length of the field to support new requirements. The primary way this problem is solved in fixed length encoding schemes is through the version number. If the length of a field must be changed, the version number is modified in packet formats supporting the new field length. This allows implementations to use the old format until all the devices in the network are upgraded to support the new format; once they are all upgraded, the entire system can be switched to the new format, whether larger or smaller.

Type Length Value

The Type Length Value (TLV) format is another widely used solution to the problem of marshaling data. Figure 2-2 shows an example from the Intermediate System to Intermediate System (IS-IS) routing protocol.

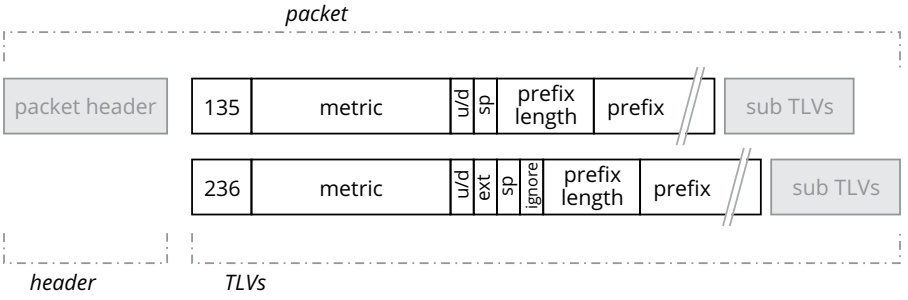


Figure 2-2 *An Example of a TLV Format from IS-IS*

In Figure 2-2, a packet consists of a header, which is normally fixed length, and then a set of TLVs. Each TLV is formatted based on its type code. In this case, there are two TLV types shown (there are many other types in IS-IS; two are used for illustration here). The first type is a 135, which carries Internet Protocol version 4 (IPv4) information. This type has several fields, some of which are fixed length—such as the metric. Others, however, such as the prefix, are variable length; the length of the field depends on the value placed in some other field within the TLV. In this case, the *prefix length* field determines the length of the prefix field. There are also sub-TLVs, which are similarly formatted, and carry information associated with this IPv4 information. The type 236 is similar to the 135, but it carries IPv6, rather than IPv4, information.

Essentially, the TLV can be considered a complete set of self-contained information carried within a larger packet. The TLV consists of three parts:

- The type code, which describes the format of the data
- The length, which describes the total length of the data
- The value, or the data itself

TLV-based formats are less compact than fixed length formats because they carry more metadata within the packet itself. The type and length information carried in the data provides the information about where to look in the dictionary for information about the formatting, as well as information about the grammar to use (how each field is formatted, etc.). TLV formats trade off the ability to change the formatting of the information being carried by the protocol without requiring every device to upgrade, or allowing some implementations to choose not to support every possible TLV, against the additional metadata carried across the wire.

TLVs are generally considered a very flexible way of marshaling data in protocols; you will find this concept to be almost ubiquitous.

Shared Object Dictionaries

One of the major problems with fixed length fields is the *fixedness* of the field definitions; if you want to modify a fixed length field protocol, you need to bump the version number and modify the packet, or you must create a new packet type with different encodings for the fields. TLV formatting solves this by including metadata inline, with the data being transmitted, at the cost of carrying more information and reducing compactness. Shared compiled dictionaries attempt to solve this problem by placing the dictionary in a sharable file (or library) rather than in a specification. Figure 2-3 illustrates the process.

In Figure 2-3, the process begins with a developer building a data structure to marshal some particular set of data to be transferred across the network. Once the data structure has been built, it is compiled into a function, or perhaps copied into a library of functions (1), and copied over to the receiver (2). The receiver then uses this library to write an application to process this data (3). On the transmitter side, the raw data is encoded into the format (4), and then carried by a protocol across the network to the receiver (5). The receiver uses its shared copy of the data format (6) to decode the data, and pass the decoded information to the receiving application (7).

This kind of system combines the flexibility of the TLV-based model with the compactness of a fixed field protocol. While the fields are fixed length, the field definitions are given in a way that allows for fast, flexible updates when the marshaling format needs to be changed. So long as the shared library is decoupled from the application using the data, the dictionary and grammar can be changed by distributing a new version of the original data structure.

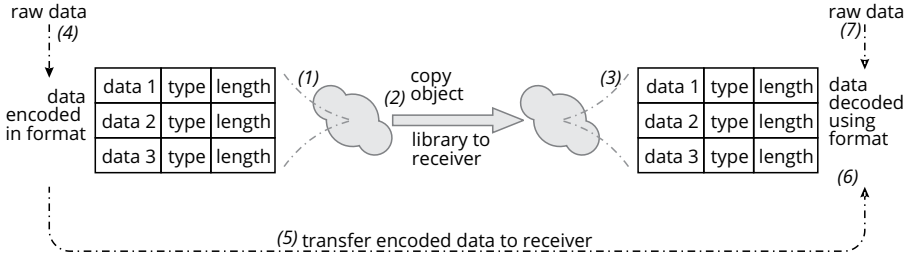


Figure 2-3 *Shared Compiled Dictionaries*

Would a flag day be required if a new version of the data structure is distributed? Not necessarily. If a version number is included in the data structure, so the receiver could match the received data with the correct data structure, then multiple versions of the data structure could exist in the system at one time. Once no sender is found using an older data format, the older structure can be safely discarded throughout the entire system.

Note

gRPC is an example of a compiled shared library marshaling system; see the "Further Reading" section for resources.

Note

While the fixed format and TLV systems count on the developers reading the specifications, and writing code as a form of sharing the grammar and dictionary, shared data structure systems, as described in this section, count on the shared dictionary being distributed in some other way. There are many different ways this could be done; for instance, a new version of software can be distributed to all the senders and receivers, or some form of distributed database can be used to ensure all the senders and receivers receive the updated data dictionaries, or some part of an application that specifically manages marshaling data can be distributed and paired with an application that generates and consumes the data. Some systems of this kind transfer the shared dictionary as part of their initial session setup. All of these are possible, and outside the scope of this present text.

Errors

No data transmission medium can be assumed to be perfect. If the transmission medium is shared, like Radio Frequency (RF), there is the possibility of interference, or even datagram collisions. This is where more than one sender attempts to transmit information simultaneously. The result is a garbled message that cannot be understood by the intended receiver. Even a dedicated medium, such as a point-to-point undersea optical (lightwave) fiber cable, can experience errors due to cable degradation or point events—even seemingly insane events, such as solar flares causing radiation, which in turn interferes with data transmission through a copper cable.

There are two key questions a network transport must answer in the area of errors:

- How can errors in the transmission of data be detected?
- What should the network do about errors in data transmission?

The following sections consider some of the possible answers to these questions.

Error Detection

The first step in dealing with errors, whether they are because of a transmission media failure, memory corruption in a switching device along the path, or any other reason, is to *detect* the error. The problem is, of course, when a receiver examines the data it receives, there is nothing to compare the data to in order to detect the error.

Parity checks are the simplest detection mechanisms. Two complementary parity checking algorithms exist. With even parity checking, one additional bit is added to each block of data. If the sum of bits in the block of data is even—that is, if there are an even number of 1 bits in the data block—the additional bit is set to 0. This preserves the even parity state of the block. If the sum of bits is odd, the additional bit is set to 1, which sets the entire block to an even parity state. Odd parity uses the same additional bit strategy, but it requires the block to have odd parity (an odd number of 1 bits).

As an example, calculate even and odd parity for these four octets of data:

```
00110011 00111000 00110101 00110001
```

Simply counting the digits reveals there are 14 *1s* and 18 *0s* in this data. To provide for error detection using a parity check, you add one bit to the data, either making the total number of *1s* in the newly enlarged set of bits even for even parity, or odd for odd parity. For instance, if you want to add an even parity bit in this case, the additional bit should be set to 0. This is because the number of *1s* is already an even number. Setting the additional parity bit to 0 will not add another *1,* and hence will not change whether the total number of *1s* is even or odd. For even parity, then, the final set of bits is

```
00110011 00111000 00110101 00110001 0
```

On the other hand, if you wanted to add a single bit of *odd* parity to this set of bits, you would need to make the additional parity bit a *1*, so there are now 15 *1s* rather than 14. For odd parity, the final set of bits is

```
00110011 00111000 00110101 00110001 1
```

To check whether or not the data has been corrupted or changed in transit, the receiver can simply note whether even or odd parity is in use, add the number of *1s,* and discard the parity bit. If the number of *1s* does not match the kind of parity in use (even of odd), the data has been corrupted; otherwise, the data appears to be the same as what was originally transmitted.

This new bit is, of course, transmitted along with the original bits. What happens if the parity bit itself is somehow corrupted? This is actually okay; assume even parity checking is in place, and a transmitter sends

```
00110011 00111000 00110101 00110001 0
```

The receiver, however, receives

```
00110011 00111000 00110101 00110001 1
```

The parity bit itself has been flipped from a 0 to a 1. The receiver will count the *1s,* determining there are 15; since even parity checking is in use, the received data will be flagged as having an error *even though it does not*. The parity check is potentially too sensitive to failures, but it is better to err on the side of caution in the case of error detection.

There is one problem with the parity check: it can detect only a single bit flip in the transmitted signal. For instance, if even parity is in use, and the transmitter sends

```
00110011 00111000 00110101 00110001 0
```

The receiver, however, receives

```
0011001 0 00111000 00110101 0011000 0 0
```

The receiver will count the number of *1s* and find it is 12; since the system is using even parity, the receiver will assume the data is correct and process it normally. However, the two bits marked out in bold have *both* been corrupted. If an even number of bits, in any combination, is modified, the parity check cannot detect the change; only when the change involves an odd number of bits can the parity check detect the modification of the data.

The Cyclic Redundancy Check (CRC) can detect a wider range of modifications in transmitted data by using division (rather than addition) in cycles across the entire data set, one small piece at a time. Working through an example is the best way to understand how a CRC is calculated. A CRC calculation begins with a polynomial, as shown in Figure 2-4.

In Figure 2-4, a three-term polynomial, $x^3 + x^2 + 1$, is expanded to include all the terms—including terms preceded by 0 (and hence do not impact the result of

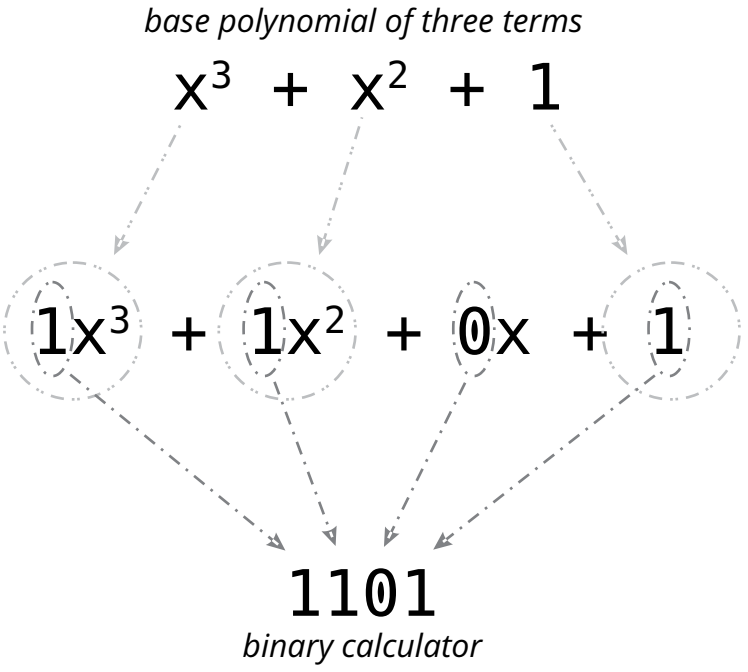


Figure 2-4 *A Polynomial Used to Calculate a CRC*

the calculation regardless of the value of x). The four coefficients are then used as a *binary calculator,* which will be used to calculate the CRC.

To perform the CRC, begin with the original binary data set, and add three extra bits (because the original polynomial, without the coefficients, has three terms; hence this is called a three-bit CRC check), as shown here:

```
10110011 00111001 (original data)
10110011 00111001 000 (with the added CRC bits)
```

These three bits are required to ensure all the bits in the original data are included in the CRC; as the CRC moves from left to right across the original data, the last bits in the original data will be included only if these padding bits are included. Now begin at the left four bits (because the four coefficients are represented as four bits). Use the Exclusive OR (XOR) operation to compare the far-left bits against the CRC bits, and save the result, as shown here:

```
10110011 00111001 000 (padded data)
1101 (CRC check bits)
----
01100011 00111001 000 (result of the XOR)
```

Note

XOR'ing two binary digits results in a 0 if the two digits match, and a 1 if they do not.

The check bits, called a *divisor,* are moved one bit to the right (some steps can be skipped here) and the operation is repeated until the end of the number is reached:

```
10110011 00111001 000
1101

01100011 00111001 000
 1101

00001011 00111001 000
    1101

00000110 00111001 000
     110 1

00000000 10111001 000
         1101

00000000 01101001 000
         1101

00000000 00000001 000
                1 101

00000000 00000000 101
```

The CRC is in the final three bits that were originally added on as padding; this is the "remainder" of the division process of moving across the original data plus the original padding. It is simple for the receiver to determine whether the data has been changed by leaving the CRC bits in place (101 in this case), and using the original divisor across the data, as shown here:

```
10110011 00111001 101
1101

01100011 00111001 101
 1101
```

```
00001011 00111001 101
    1101

00000110 00111001 101
     110 1

00000000 10111001 101
         1101

00000000 01101001 101
          1101

00000000 00000001 101
                1 101

00000000 00000000 000
```

If the data has not been changed, the result of this operation should always result in 0. If a bit has been changed, the result will not be 0, as shown here:

```
10110011 00111000 000
1101

01100011 00111000 000
 1101

00001011 00111000 000
    1101

00000110 00111000 000
     110 1

00000000 10111000 000
         1101

00000000 01101000 000
          1101

00000000 00000000 000
                1 101

00000000 00000001 000
```

The CRC might seem like a complex operation, but it plays to a computer's strong points—finite length binary operations. If the length of the CRC is set the same as a standard small register in common processors, say eight bits, calculating the CRC is a fairly straightforward and quick process. CRC checks have the advantage of being resistant to multibit changes, unlike the parity check described previously.

Error Correction

Detecting an error is only half of the problem, however. Once the error is detected, what should the transport system do? There are essentially three options.

The transport system can simply throw the data away. In this case, the transport is effectively transferring the responsibility of what to do about the error up to higher-level protocols or perhaps the application itself. As some applications may need a complete data set with no errors (think a file transfer system, or a financial transaction), they will likely have some way to discover any missing data and retransmit it. Applications that do not care about small amounts of missing data (think a voice stream) can simply ignore the missing data, reconstructing the information at the receiver as well as possible given the missing information.

The transport system can signal the transmitter that there is an error, and let the transmitter decide what to do with this information (generally the data in error will be retransmitted).

The transport system can go beyond throwing data away by including enough information in the original transmission and determine where the error is and attempt to correct it. This is called *Forward Error Correction* (FEC). Hamming codes, one of the first FEC mechanisms developed, is also one of the simplest to explain. The Hamming code is best explained by example; Table 2-2 will be used to illustrate.

Table 2-2 *An Illustration of the Hamming Code*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
	0001	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**
	P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7	D8
			1		0	1	1		0	0	1	1
P1	1		X		X		X		X		X	
P2		0	X			X	X			X	X	
P4				1	X	X	X					X
P8								0	X	X	X	X

In Table 2-2:

- Each bit in the 12-bit space that is a power of two (1, 2, 4, 6, 8, etc.) and the first bit are set aside as parity bits.
- The 8-bit number to be protected with FEC, 10110011, has been distributed across the remaining bits in the 12-bit space.
- Each parity bit is set to 0, and then parity is calculated for each parity bit by adding the number of *1s* in positions where the binary bit number has the same bit set as the parity bit. Specifically:
  - P1 has the far-right bit set in its bit number; the other bits in the number space that also have the far right bit set are included in the parity calculation (see the second row in the table to find all the bit positions in the number with the far-right bit set). These are indicated in the table with an *X* in the P1 row. The total number of *1s* is an odd number, 3, so the P1 bit is set to *1* (this example is using even parity).
  - P2 has the second bit from the right set; the other bits in the number space that have the second from the right bit set are included in the parity calculation, as indicated with an *X* in the P2 row of the table. The total number of *1s* is an even number, 4, so the P2 bit is set to 0.
  - P4 has the third bit from the right set, so the other bits that have the third from the right bit set in their position numbers, as indicated with an *X* in the P3 row. There are an odd number of *1s* in the marked columns, so the P4 parity bit is set to 1.

To determine if any information has changed, the receiver can check the parity bits in the same way the sender has calculated them; the total number of *1s* in any set should be an even number, including the parity bit. If one of the data bits has been flipped, the receiver should never find a single parity error, because each of the bit positions in the data is covered by multiple parity bits. To discover which data bit is incorrect, the receiver adds the positions of the parity bits that are in error; the result is the bit position that has been flipped. For instance, if the bit in position 9, which is the fifth data bit, is flipped, then parity bits P1 and P8 would be both in error. In this case, 8 + 1 = 9, so the bit in position 9 is in error and flipping it would correct the data. If a *single* parity bit is in error—for example, P1 *or* P8—then it is that parity bit which has been flipped, and the data itself is correct.

While the Hamming code is ingenious, there are many bit flip patterns it cannot detect. A more modern code, such as *Reed-Solomon,* can detect and correct a wider range of error conditions while adding less additional information to the data stream.

Note

There are a large number of different kinds of CRC and error correction codes used throughout the communications world. CRC checks are classified by the number of bits used in the check (the number of bits of padding, or rather the length of the polynomial), and, in some cases, the specific application. For instance, the Universal Serial Bus uses a 5-bit CRC (CRC-5-USB); the Global System for Mobile Communications (GSM), a widely used cellular telephone standard, uses CRC-3-GSM; Code Division Multi-Access (CDMA), another widely used cellular telephone standard, uses CRC-6-CDMA2000A, CRC-6-CDMA2000B, and CRC-30; and some car area networks (CANs), used to tie together various components in a vehicle, use CRC-17-CAN and CRC-21-CAN. Some of these various CRC functions are not a single function, but rather a class, or family, of functions, with many different codes and options within them.

Multiplexing

You walk into a room and shout, "Joe!" Your friend, Joe, turns around and begins a conversation on politics and religion (the two forbidden topics, of course, in any polite conversation). This ability to use a single medium (the air through which your voice travels) to address one person, even though many other people are using the same medium for other conversations at the same time, is what is called, in network engineering, multiplexing. More formally:

> **Multiplexing is used to allow multiple entities attached to the network to communicate over a shared network.**

Why is the word *entities* used here instead of hosts? Returning to the "conversation with Joe" example, imagine the one way you can communicate with Joe is through his teenaged child, who only texts (never talks). In fact, Joe is part of a family of several hundred to several thousand people, and all the communications for this entire family must come through this one teenager, and each person in the family has multiple conversations running concurrently, sometimes on different topics with the same person. The poor teenager must text very quickly, and keep a lot of information in her head, like "Joe is having four conversations with Mary," and must keep the information in each conversation completely separate from the other. This is closer to how network multiplexing really works; consider:

- There could be millions (or billions) of hosts connected to a single network, all sharing the same physical network to communicate with one another.
- Each of these hosts actually contains many applications, possibly several hundred, each of which can communicate with any of the hundreds of applications on any other host connected to the network.
- Each of these applications may, in fact, have several conversations to any other application running on any other host in the network.

If this is starting to sound complicated, that is because it is. The question this section needs to answer, then, is this:

How do hosts multiplex effectively over a computer network?

The following sections consider the most commonly used solutions in this space, as well as some interesting problems tied up in this basic problem, such as multicast and anycast.

Addressing Devices and Applications

Computer networks use a series of hierarchically arranged addresses to solve these problems; Figure 2-5 illustrates.

In Figure 2-5, there are four levels of addressing shown:

- At the physical link level, there are interface addresses that allow two devices to address a particular device individually.
- At the host level, there are host addresses that allow two hosts to address a particular host directly.
- At the process level, there are *port numbers* that, combined with the host address, allow two processes to address a particular process on a particular device.

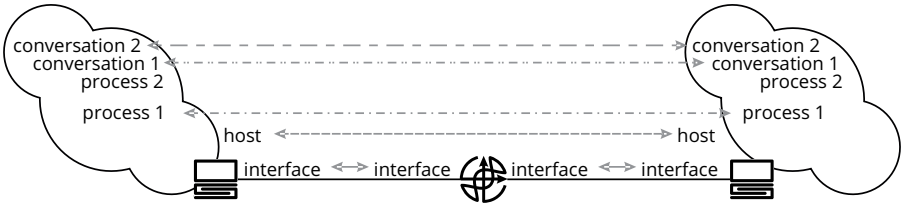


Figure 2-5 *Addressing Multiple Levels of Entities in a Network*

- At the conversation level, the set of source port, destination port, source address, and destination address can be combined to uniquely identify a particular conversation, or flow.

This diagram and explanation appear very clean. In real life, things are much messier. In the most widely deployed addressing scheme, the Internet Protocol (IP), there are no *host-level addresses*. Instead, there are logical and physical addresses on a per interface basis.

Note

IP and IP addressing will be considered in more detail in Chapter 5, "Higher Layer Data Transports."

Multiplexing and multiplexing identifiers (addresses) are stacked hierarchically on top of one another in a network.

Note

Mechanisms that associate one kind of address with another between some layers will be considered more fully in Chapter 6, "Interlayer Discovery."

There are some situations, however, in which you want to send traffic to more than one host at a time; for these situations, there are multicast and anycast. These two special kinds of addressing will be considered in the following sections.

On Physical Links, Broadcasts, and Failure Domains

The clear-cut model illustrated in Figure 2-5 is made more complex when you consider the concept of broadcast domains and physical connectivity. Some media types (notably Ethernet, which is covered in more detail in Chapter 4, "Lower Layer Transports") are designed so every device connected to the same physical link receives every packet transmitted onto the physical media—hosts just ignore packets not addressed to one of the addresses associated with the physical interface connected to the physical wire. In modern networks, however, physical Ethernet wiring rarely allows every device to receive every other device's packets; instead, there is a switch in the middle of the network that blocks packets not destined to a particular device from being transmitted on the physical wire connected to that host.

In these protocols, however, there are explicit addresses set aside for packets that *should be* transmitted to every host that would normally receive every packet if the switch was not there, or that every host should receive and process (normally, this is some form of the all 1s or all 0s version of the address). These are called broadcasts. Any device that will receive, and process, a broadcast sent by a device is said to be part of the device's broadcast domain. The concept of a broadcast domain has traditionally been closely associated with a failure domain, because network failures impacting one device on a broadcast domain often impact every device on the broadcast domain (see Chapter 23, "Redundant and Resilient," for more information on failure domains).

Do not be surprised if you find all of this rather confusing, because it is, in fact, rather confusing. The basic concepts of broadcasts and broadcast domains still exist, and are still important in understanding the operation of a network, but the meaning of the term can change, or even not apply, in some situations. Be careful when considering any situation to make certain you really understand how and where such broadcast domains really are, and how specific technologies impact the relationship between physical connectivity, addressing, and broadcast domains.

Multicast

Note

This short explanation cannot really do justice to the entire scope of solutions available to build multicast trees; see the "Further Reading" section at the end of the chapter for more material to consider in this area.

If you have a network like the one shown in Figure 2-6, and you need A to distribute the same content to G, H, M, and N, how would you go about doing this?

You could either generate four copies of the traffic, sending one stream to each of the receivers using normal *(unicast)* forwarding, or you could somehow send the traffic to a single address that the network knows to replicate so all four hosts receive a copy. This latter option is called multicast, which means using a single address to transmit traffic to multiple receivers. The key problem to solve in multicast is to forward and replicate traffic as it passes through the network so each receiver who is interested in the stream will receive a copy.

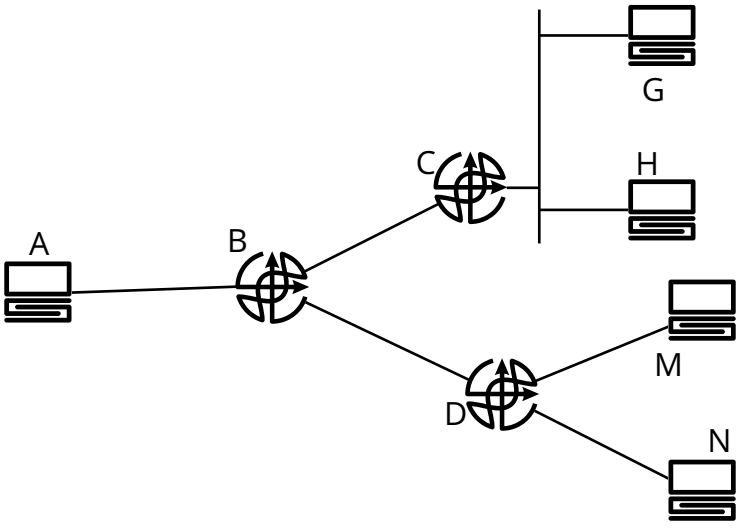


Figure 2-6 *A Multicast Example*

Note

The set of devices interested in receiving a stream of packets from a multicast source is called a multicast group. This can be a bit confusing because the address used to describe the multicast stream is also called a multicast group in some situations. The two uses are *practically* interchangeable in that the set of devices interested in receiving a particular set of multicast packets will join the multicast group, which, in effect, means listening to a particular multicast address.

Note

In cases where the multicast traffic is bidirectional, this problem is much more difficult to solve. For instance, assume there is a requirement to build a multicast group with every host in the network shown in Figure 2-6 except N, and further that any multicast transmitted to the multicast group's address be delivered to every host within the multicast group.

The key problem for multicast to solve can be broken into two problems:

- How do you discover which devices would like to receive a copy of traffic transmitted to the multicast group?

- How do you determine which devices in the network should replicate the traffic, and on which interfaces they should send copies?

One possible solution is to use local requests to build a tree through which the multicast traffic should be forwarded through the network. An example of such a system is *Sparse Mode* in Protocol Independent Multicast (PIM). In this this process, each device sends a *join* message for the multicast streams it is interested in; these joins are passed upstream in the network until the sender (the host sending packets through the multicast stream) is reached. Figure 2-7 is used to illustrate this process.

In Figure 2-7:

1. A is sending some traffic to a multicast group (address); call it *Z*.
2. N would like to receive a copy of Z, so it sends a request (a join) to its upstream router, D, for a copy of this traffic.
3. D does not have a source for this traffic, so it sends a request to the routers it is connected to for a copy of this traffic; in this case, the only router D sends the request to is B.

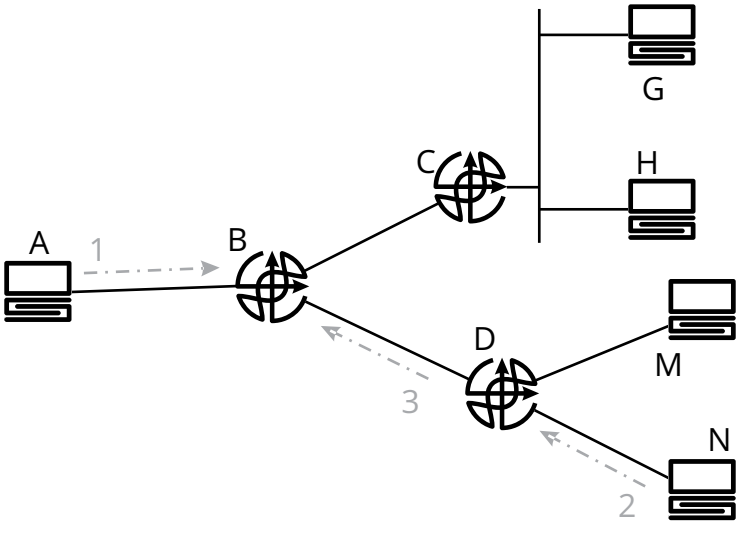


Figure 2-7 *Sparse mode multicast*

At each hop, the router receiving the request will place the interface on which it received the request into its *Outbound Interface List (OIL),* and begin forwarding traffic received in the given multicast group received on any other interface. In this way, a path from the receiver to the originator of the traffic can be built; this is called a *reverse path tree*.

A second option for discovering which hosts are interested in receiving traffic for a specific multicast group is through some sort of registration server. Each host that would like to receive a copy of the stream can register its desire with a server. There are several ways the host can discover the presence of the server, including

- Treating the multicast group address like a domain name, and looking up the address of the registration server by querying for the multicast group address
- Building and maintaining a list, or mapping, of groups to servers mapping in a local table
- Using some form of hash algorithm to compute the registration server from the multicast group address

The registrations can either be tracked by devices on the path to the server, or, once the set of receivers and transmitters is known, the server can signal the appropriate devices along the path which ports should be configured for replicating and forwarding packets.

Anycast

Another problem multiplexing solutions face is being able to address a specific instance of a service residing in implemented on multiple hosts using a single address. Figure 2-8 illustrates.

In Figure 2-8, some service, *S,* needs to be designed to increase its performance. To accomplish this goal, a second copy of the service has been created, with the two copies being named S1 and S2. These two copies of the service are running on two servers, M and N. The problem anycast seeks to solve is this:

How can clients be directed to the most optimal instance of a service?

One way of solving this problem is to direct all the clients to a single device and have a *load balancer* split the traffic to the servers based on the topological location of the client, the load of each server, and other factors. This solution is not always ideal, however. For instance, what if the load balancer cannot handle all the connection requests generated by the clients who want to reach various copies of the

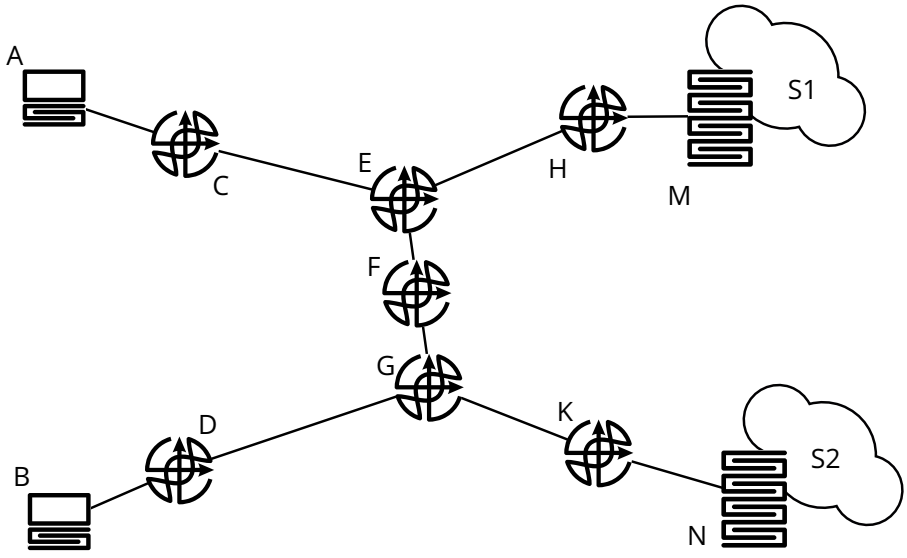


Figure 2-8 *An Anycast Example*

service? What sorts of complexities are going to be added to the network to allow the load balancer to track the health of the various copies of the service?

> **Note**
>
> Load balancing is considered in Chapter 7, "Packet Switching."

Anycast solves this problem by assigning *the same address* to each copy of the service. In the network illustrated in Figure 2-8, then, M and N would use the same address to provide reachability to S1 and S2. M and N would have different addresses assigned and advertised to provide reachability to other services, and to the devices themselves, as well.

H and K, the first hop routers beyond M and N, would advertise this same address into the network. When C and D receive two routes to the same destination, they will choose the closest route in terms of metrics. In this case, if every link in the same network is configured with the same metric, then C would direct traffic sourced from A, and destined to the service's address, toward M. D, on the other hand, will direct traffic sourced from B, and destined to the service's address, toward N. What happens if two instances of the service are about the same distance apart? The router will choose one of the two paths using a local hash algorithm.

Note

See Chapter 7 for more information about equal cost multipath switching, and how using a hash ensures the same path is used for each packet in a flow. Routing is generally stable enough, even in the Internet, to use anycast solutions with stateful protocols.[5]

5. Palsson et al., "TCP over IP Anycast—Pipe Dream or Reality?"

Anycast is often used for large-scale services that must scale by provisioning a lot of servers to support the single service. Examples include the following:

- Most large-scale Domain Name Service (DNS) system servers are actually a set of servers accessible through an anycast address.
- Many large-scale web-based services, particularly social media and search, where a single service is implemented on a large number of edge devices.
- Content caching services often use anycast in distributing and serving information.

Designed correctly, anycast can provide effective load balancing as well as optimal performance for services.

Flow Control

Do you remember your great aunt (or was it your second cousin once removed?) who talked so fast that you could not understand a word she was saying? Some computer programs talk too fast, too. Figure 2-9 illustrates.

In Figure 2-9:

- At *Time* 1 (T1), the sender is transmitting about four packets for every three the receiver can process. The receiver has a five-packet buffer to store unprocessed information; there are two packets in this buffer.
- At T2, the sender has transmitted four packets, and the receiver has processed three; the buffer at the receiver is now holding three packets.
- At T3, the sender has transmitted four packets, and the receiver has processed three; the buffer at the receiver is now holding four packets.
- At T4, the sender has transmitted four packets, and the receiver has processed three; the buffer at the receiver is now holding five packets.

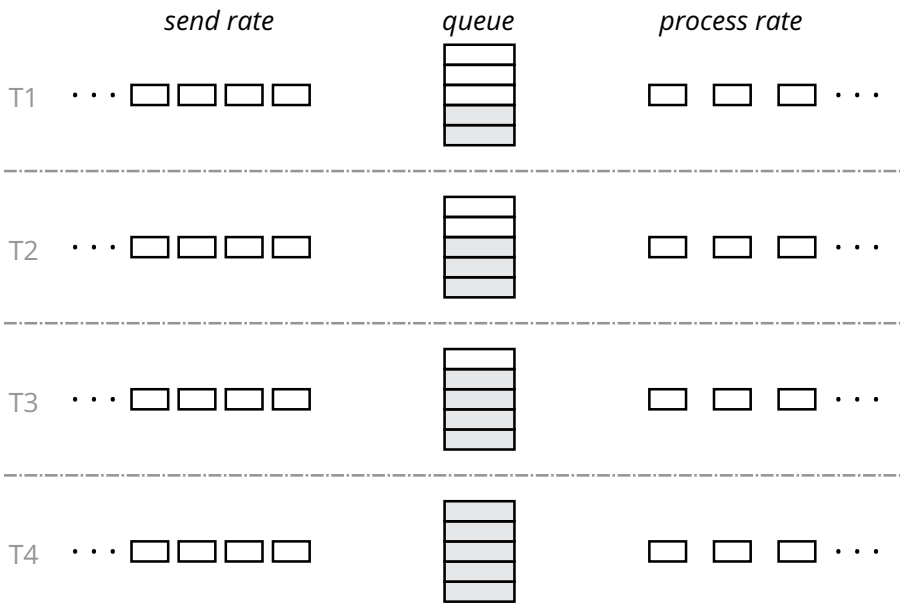


Figure 2-9 *Buffer Overflow Example*

The next packet transmitted will be dropped by the receiver because there is no space in the buffer to store it while the receiver is processing packets so they can be removed. What is needed is some sort of feedback loop to tell the transmitter to slow down the rate at which it is sending packets, as illustrated in Figure 2-10.

This kind of feedback loop requires either implicit signaling or explicit signaling between the receiver and the transmitter. Implicit signaling is more widely deployed. In implicit signaling, the transmitter assumes the packet has not been received based on some observation about the traffic stream. For instance, the receiver may acknowledge the receipt of some later packet, or the receiver may simply not acknowledge receiving a particular packet, or the receiver may not send anything for a long period of time (in network terms). In explicit signaling, the receiver somehow directly informs the sender that a specific packet has not been received.

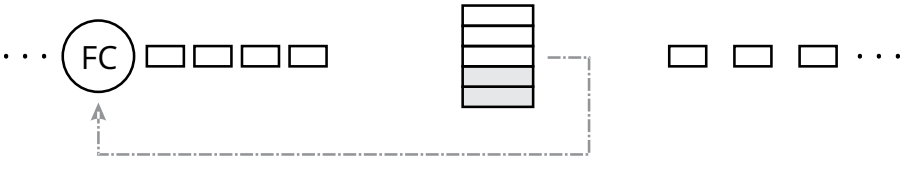


Figure 2-10 *A Feedback Loop to Control Packet Flow*

Windowing

Windowing, combined with implicit signaling, is by far the most widely deployed flow control mechanism in real networks. Windowing essentially consists of the following:

1. A transmitter sends some amount of information to the receiver.
2. The transmitter waits before deciding if the information has been correctly received or not.
3. If the receiver acknowledges receipt within a specific amount of time, the transmitter sends new information.
4. If the receiver does not acknowledge receipt within a specific amount of time, the transmitter resends the information.

Implicit signaling is normally used with windowing protocols by simply not acknowledging the receipt of a particular packet. Explicit signaling is *sometimes* used when the receiver knows it has dropped a packet, when received data contains errors, data is received out of order, or data is otherwise corrupted in some way. Figure 2-11 illustrates the simplest windowing scheme, a single packet window.

In a single packet window (also sometimes called a ping pong), the transmitter sends a packet only when the receiver has acknowledged (shown as an ack in the illustration) the receipt of the last packet transmitted. If the packet is not received, the receiver will not acknowledge it. On sending a packet, the sender sets a timer,

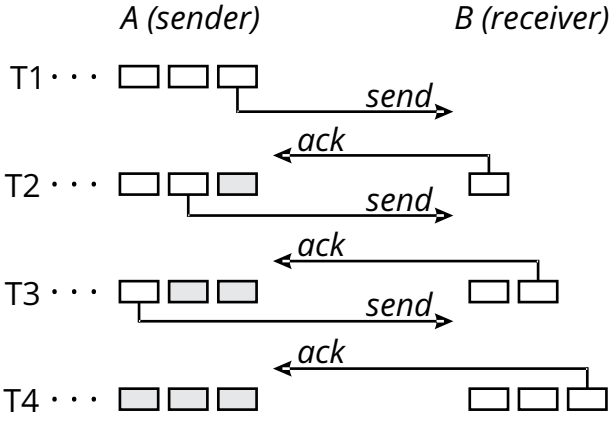


Figure 2-11 *A Single Packet Window*

normally called the retransmit timer; once this timer wakes up (or expires), the sender will assume the receiver has not received the packet, and resend it.

How long should the sender wait? There are a number of possible answers to this question, but essentially the sender can either wait a fixed amount of time, or it can set a timer based on information inferred from previous transmissions and network conditions. A simple (and naïve) scheme would be to

- Measure the length of time between sending a packet and receiving an acknowledgment, called the Round Trip Time (RTT, though normally written in the lowercase, so rtt).
- Set the retransmit timer to this number plus some small amount of buffer time to account for any variability in the rtt over multiple transmissions.

Note

More information about various ways to calculate the retransmit timer are considered in Chapter 5.

It is also possible for the receiver to receive two copies of the same information:

1. A transmits a packet and sets its retransmit timer.
2. B receives the packet, but
   a. Is not able to acknowledge receipt because it is out of memory or is experiencing high processor utilization or some other condition.
   b. Sends an acknowledgment, but the acknowledgment is dropped by a network device.
3. The retransmit timer at A times out, so the sender transmits another copy of the packet.
4. B receives this second copy of the same information.

How can the receiver detect duplicated data? It does seem possible for the receiver to compare the packets received to see if there is duplicate information, but this will not always work—perhaps the sender *intended* to send the same information twice. The usual method of detecting duplicate information is by including some sort of sequence number in transmitted packets. Each packet is given a unique

sequence number while being built by the sender; if the receiver receives two packets with the same sequence number, it assumes the data is duplicated and discards the copies.

A window size of 1, or a ping pong, requires one round trip between the sender and the receiver for each set of data transmitted. This would generally result in a very slow transmission rate. If you think of the network as the end-to-end railroad track, and each packet as a single train car, the most efficient use of the track, and the fastest transmission speed, is going to be when the track is always full. This is not physically possible, however, in the case of a network because the network is used by many sets of senders and receivers, and there are always network conditions that will prevent the network utilization from reaching 100%. There is some balance between the increased efficiency and speed of sending more than one packet at a time, and the multiplexing and "safety" of sending fewer packets at a time (such as one). If a correct balance point can be calculated in some way, a fixed window flow control scheme may work well. Figure 2-12 illustrates.

In Figure 2-12, assuming a three-packet fixed window:

- At T1, T2, and T3, A transmits packets; A does not need to wait for B to acknowledge anything to send these three packets, as the window size is fixed at 3.
- At T4, B acknowledges these three packets, which allows A to transmit another packet.
- At T5, B acknowledges this new packet, even though is it only one packet. B does not need to wait until A has transmitted three more packets to acknowledge a single packet. This acknowledgment allows A to have enough *budget* to send three more packets.
- At T5, T6, and T7, A sends three more packets, filling its window. It must now wait until B acknowledges these three packets to send more information.
- At T8, B acknowledges the receipt of these three packets.

In windowing schemes where the window size is more than one, there are four kinds of acknowledgments a receiver can send to the transmitter:

- **Positive acknowledgment:** The receiver acknowledges the receipt of each packet individually. For instance, if sequence numbers 1, 3, 4, and 5 have been received, the receiver will acknowledge receiving those specific packets. The transmitter can infer which packets the receiver has not received by noting which sequence numbers have not been acknowledged.

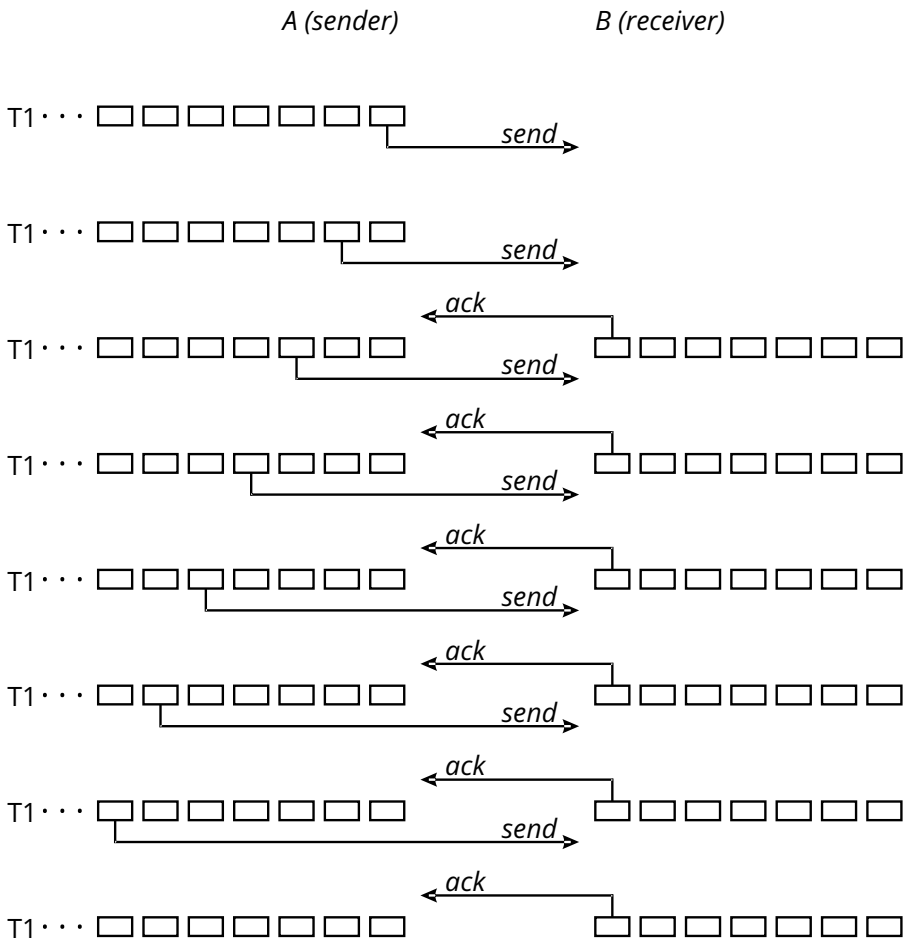


Figure 2-12 *An Example of Fixed Window Flow Control*

- **Negative acknowledgment:** The receiver sends a negative acknowledgment for packets it infers are missing, or were corrupted when received. For instance, if sequence numbers 1, 3, 4, and 5 have been received, the receiver may infer that sequence number 2 is missing and send a negative acknowledgment for this packet.
- **Selective acknowledgment:** This essentially combines positive and negative acknowledgment, as above; the receiver sends both positive and negative acknowledgments for each sequence of received information.

- **Cumulative acknowledgment:** Acknowledgment of the receipt of a sequence number implies receipt of all information with lower sequence numbers. For instance, if sequence number 10 is acknowledged, the information contained in sequence numbers 1–9 is implied, as well as the information contained in sequence number 10.

A third windowing mechanism is called sliding window flow control. This mechanism is very similar to a fixed window flow control mechanism, *except the size of the window is not fixed*. In sliding window flow control, the transmitter can dynamically modify the size of the window as network conditions change. The receiver does not know what size the window is, only that the sender transmits packets, and, from time to time, the receiver acknowledges some or all of them using one of the acknowledgment mechanisms described in the preceding list.

Sliding window mechanisms add one more interesting question to the questions already considered in other windowing mechanisms: What size should the window be? A naïve solution might just calculate the rtt and set the window size to some multiple of the rtt. More complex solutions have been proposed; some of these will be considered in Chapter 5, in the discussion of the Transmission Control Protocol (TCP).

Negotiated Bit Rates

Another solution, more often used in circuit switched rather than packet switched networks, is for the sender, receiver, and network to negotiate a bit rate for any particular flow. A wide array of possible bit rates have been designed for a number of different networking technologies; perhaps the "most complete set" is for *Asynchronous Transfer Mode* (ATM)—look for ATM networks in your nearest networking history museum, because ATM is rarely deployed in production networks any longer. The ATM bit rates are:

- **Constant Bit Rate (CBR):** The sender will be transmitting packets (or information) at a constant rate; hence, the network can plan around this constant bandwidth load, and the receiver can plan around this constant bit rate. This bit rate is normally used for applications requiring time synchronization between the sender and receiver.
- **Variable Bit Rate (VBR):** The sender will be transmitting traffic at a variable rate. This rate is normally negotiated with several other pieces of information about the flow that help the network and the receiver plan resources, including:
  - The peak rate, or the maximum packets per second the sender plans to transmit

  - The sustained rate, or the rate at which the sender plans to transmit normally
  - The maximum burst size, or the largest number of packets the sender intends to transmit over a very short period of time

- **Available Bit Rate (ABR):** The sender intends to rely on the capability of the network to deliver traffic on a best-effort basis, using some other form of flow control, such as a sliding window technique, to prevent buffer overflows and adjust transmitted traffic to the available bandwidth.

Final Thoughts on Transport

This chapter begins with the fundamentals of understanding the entire scope of the network engineering problem space: transporting data across the network. Four specific problems were uncovered by considering the *human language* space, and several solutions were presented at a high level:

- To marshal the data, fixed length and TLV-based systems were considered, as well as the concepts of *metadata, dictionaries,* and *grammars.*
- To manage errors, two methods were considered to detect errors, parity checks and the CRC; and one method was considered for error correction, the Hamming Code.
- To allow multiple senders and receivers to use the same physical media, several concepts in multiplexing were considered, including multicast and anycast.
- To prevent buffer overflows, several kinds of windowing were explored, and negotiated bit rates defined.

Like many other areas you will encounter in this book, the world of transport can become an entire specialty. Understanding the basics, however, is important for every network engineer. The next chapter will consider some models that will help you to put data transport, which is generally associated with forwarding, or the data plane, into a larger context. Chapters 4 and 5 will consider several different examples of transport protocols, pulling the concepts in this chapter and the next into real-life examples.

Further Reading

Some of these further reading resources are provided to help in answering the study questions for this chapter.

Conran, Matt. "Know Anycast? Think Before You Talk." Blog. *CacheFly*, February 22, 2017. https://insights.cachefly.com/anycast-think-before-you-talk-part-i.

———. "Anycast—Think Before You Talk." Blog. *CacheFly*, February 22, 2017. https://insights.cachefly.com/anycast-think-before-you-talk-part-ii.

"Flag Day." Accessed June 6, 2017. http://www.catb.org/jargon/html/F/flag-day.html.

Gleick, James. *The Information: A History, A Theory, A Flood*. New York: Vintage, 2012.

"Grpc /." Accessed June 7, 2017. http://www.grpc.io/docs/tutorials/basic/c.html.

Internet Protocol. Request for Comments 791. RFC Editor, 1981. doi:10.17487/RFC0791.

Koopman, P. "32-Bit Cyclic Redundancy Codes for Internet Applications." In *Proceedings International Conference on Dependable Systems and Networks*, 459–68, 2002. doi:10.1109/DSN.2002.1028931.

Koopman, P., and T. Chakravarty. "Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks." In *International Conference on Dependable Systems and Networks, 2004*, 145–54, 2004. doi:10.1109/DSN.2004.1311885.

Loveless, Josh, Ray Blair, and Arvind Durai. *IP Multicast, Volume I: Cisco IP Multicast Networking*. 1st edition. Indianapolis, IN: Cisco Press, 2016.

———. *IP Multicast, Volume II: Advanced Multicast Concepts and Large-Scale Multicast Design*. 1st edition. Indianapolis, IN: Cisco Press, 2017.

McPherson, Danny R., David Oran, Dave Thaler, and Eric Osterweil. *Architectural Considerations of IP Anycast*. Request for Comments 7094. RFC Editor, 2014. doi:10.17487/RFC7094.

Moon, Todd K. *Error Correction Coding: Mathematical Methods and Algorithms*. 1st edition. Hoboken, NJ: Wiley-Interscience, 2005.

Morelos-Zaragoza, Robert H. *The Art of Error Correcting Coding*. 2nd edition. Chichester; Hoboken, NJ: Wiley, 2006.

Moy, John. "OSPF Specification." Request for Comment. RFC Editor, October 1989. doi:10.17487/RFC1131.

———. "OSPF Version 2." Request for Comment. RFC Editor, April 1998. doi:10.17487/RFC2328.

Palsson, Bret, Prashanth Kumar, Samir Jafferali, and Zaid Ali Kahn. "TCP over IP Anycast—Pipe Dream or Reality?" Blog. *LinkedIn Engineering Blog*, September 2015. https://engineering.linkedin.com/network-performance/tcp-over-ip-anycast-pipe-dream-or-reality.

Postel, J. *NCP/TCP Transition Plan*. Request for Comments 801. RFC Editor, 1981. doi:10.17487/RFC0801.

Shannon, Claude E., and Warren Weaver. *The Mathematical Theory of Communication*. 4th edition. Champaign, IL: University of Illinois Press, 1949.

Soni, Jimmy, and Rob Goodman. *A Mind at Play: How Claude Shannon Invented the Information Age*. New York: Simon & Schuster, 2017.

Stone, James V. *Information Theory: A Tutorial Introduction*. 1st edition. England: Sebtel Press, 2015.

"Understanding the CBR Service Category for ATM VCs." *Cisco*. Accessed June 10, 2017. http://www.cisco.com/c/en/us/support/docs/asynchronous-transfer-mode-atm/atm-traffic-management/10422-cbr.html.

Warren, Henry S. *Hacker's Delight*. 2nd edition. Upper Saddle River, NJ: Addison-Wesley Professional, 2012.

Williamson, Beau. *Developing IP Multicast Networks, Volume I*. Indianapolis, IN: Cisco Press, 1999.

Review Questions

1. While TLVs almost always require more space to carry a piece of information than a fixed length field, there are some cases where the fixed length field will be less efficient. Carrying IPv6 addresses is one specific instance of a TLV being more efficient than a fixed length field. Describe why this is. Comparing the way routing protocols carry IPv4 and IPv6 addresses is a good place to start in understanding the answer. In particular, examine the way IPv4 addresses are carried in OSPF version 2, and compare this with the way these same addresses are carried in BGP.
2. Consider the following data types and determine whether you would use a fixed length field or a TLV to carry each one, and why.
   a. The time and date
   b. A person's full name

c. A temperature reading

d. The square footage of a building

e. A series of audio or video clips

f. A book broken down into sections such as paragraphs and chapters

g. The city and state in an address

h. The house number or postal code in an address

3. What is the relationship between the bit error rate (BER) and the amount of information required to detect and/or repair errors in a data transmission stream? Can you explain why this might be?

4. Under some conditions, it makes more sense to send enough information to correct data on receipt (such as using a Hamming code). In others, it makes more sense to discover the error and throw the data away. These conditions would not be just the link type, however, or just the application; they would be a combination of the two. What link characteristics, combined with what kinds of application characteristics, would suggest the use of FEC? Which ones would suggest the use of error detection combined with retransmitting the data? It might be best to think of specific applications and specific link types first, and then generalize from there.

5. How many bit flip changes can a parity check detect?

6. Implicit and explicit signaling have different characteristics, or rather different tradeoffs. Describe at least one positive and one negative aspect of each form of signaling for error detection and/or correction.

7. In a large-scale deployment of anycast, it is possible for packets from a single stream to be delivered to multiple receivers. There are two broad solutions to this problem; the first is for receivers to force the sender to reset their state if a packet appears to be misdelivered in this way. Another is to constrict the interface between the sender and receiver in a way that allows state to be contained to a single transaction. One form of this latter solution is called atomic transactions, and is often implemented in RESTful interfaces. Consider these two possible solutions, and describe the kinds of applications, giving specific examples of applications, that might be better suited for each of these two solutions.

8. Would you always consider the dictionary and the grammar forms of metadata? Why or why not?

9. Find three other kinds of metadata that do not involve the way the data is formatted, but rather describe the data in a way that might be useful to an attacker trying to understand a specific process, such as transferring funds between two accounts. Is there a specific limit to what might be considered metadata, or is it more accurate to say "metadata is in the eye of the beholder"?
10. Consider the negotiated bit rates explained toward the end of the chapter. Is it possible to truly provide a constant bit rate in a packet switched network? Does your answer depend on the network conditions? If so, what conditions would impact the answer to the question?

Chapter 3

Modeling Network Transport

Learning Objectives

After reading this chapter, you should be able to:

- Understand the value of protocol stack models to network engineering
- Understand the Department of Defense (DoD) network protocol stack model, including the purpose of each layer
- Understand the Open Systems Interconnect (OSI) network protocol stack model, including the purpose of each layer
- Understand the Recursive Internet Architecture (RINA) model, and how it is different from the DoD and OSI models
- Understand the difference between the connection-oriented and connectionless models

The set of problems and solutions considered in the preceding chapter provides some insight into the complexity of network transport systems. How can engineers engage with the apparent complexity involved in such systems?

The first way is to look at the basic problems transport systems solve, and understand the range of solutions available for each of those problems. The second is to build models that will aid in the understanding of transport protocols by

- Helping engineers classify transport protocols by their purpose, the information each protocol contains, and the interfaces between protocols

- Helping engineers know which questions to ask in order to understand a particular protocol, or to understand how a particular protocol interacts with the network over which it runs, and the applications that it carries information for
- Helping engineers understand how single protocols fit together to make a transport system

Chapter 1, "Fundamental Concepts," provided a high-level overview of the transport problem and solution spaces. This chapter will tackle the second way in which engineers can understand protocols more fully: models. Models are essentially abstract representations of the problems and solutions considered in the previous chapter; they provide a more visual and module-focused representation, showing how things fit together. This chapter will consider this question:

How can transport systems be modeled in a way that allows engineers to quickly and fully grasp the problems these systems need to solve, as well as the way multiple protocols can be put together to solve them?

Three specific models will be considered in this chapter:

- The United States Department of Defense (DoD) model
- The Open Systems Interconnect (OSI) model
- The Recursive Internet Architecture (RINA) model

Each of these three models has a different purpose and history. A second form of protocol classification, *connection oriented* versus *connectionless,* will also be considered in this chapter.

United States Department of Defense (DoD) Model

In the 1960s, the US Defense Advanced Research Projects Agency (DARPA) sponsored the development of a packet switched network to replace the telephone network as a primary means of computer communications. Contrary to the myth, the original idea was not to survive a nuclear blast, but rather to create a way for the various computers then being used at several universities, research institutes, and government offices to communicate with one another. At the time, each computer system used its own physical wiring, protocols, and other systems; there was no way to interconnect these devices in order to even transfer data files, much less create anything like the "world wide web," or cross-execute software. These original models

were often designed to provide terminal-to-host communications, so you could install a *remote terminal* into an office or shared space, which could then be used to access the shared resources of the system, or *host*. Much of the original writing around these models reflects this reality.

One of the earliest developments in this area was the DoD model, shown in Figure 3-1.

The DoD model separated the job of transporting information across a network into four distinct functions, each of which could be performed by one of many protocols. The idea of having multiple protocols at each layer was considered somewhat controversial until the late 1980s, and even into the early 1990s. In fact, one of the key differences between the DoD and the original incarnation of the OSI model is the concept of having multiple protocols at each layer.

In the DoD model:

- The **physical layer** is responsible for getting the 0s and 1s modulated, or serialized, onto the physical link. Each link type has a different format for signaling a 0 or a 1; the physical layer is responsible for translating 0s and 1s into physical signals.
- The **internet layer** is responsible for transporting data between systems that are not connected through a single physical link. The internet layer, then, provides networkwide addresses, rather than link local addresses, and also provides some means for discovering the set of devices and links that must be crossed to reach these destinations.
- The **transport layer** is responsible for building and maintaining sessions between communicating devices and providing a common transparent data transmission mechanism for streams or blocks of data. Flow control and reliable transport may also be implemented in this layer, as in the case of TCP.

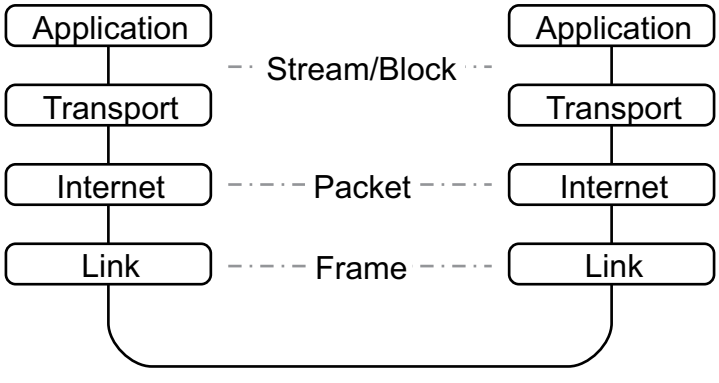


Figure 3-1 *The Four-Layer DoD Model*

- The **application layer** is the interface between the user and the network resources, or specific applications that use and provide data to other devices attached to the network.

The application layer, in particular, seems out of place in a model of network transport. Why should the application using the data be considered part of the transport system? Because early systems considered the *human user* the ultimate user of the data, and the application as primarily a way to munge data to be presented to the actual user. Much of the machine-to-machine processing, heavy processing of data before it is presented to a user, and simple storage of information in digital format were not even considered viable use cases. As information was being transferred from one person to another, the application was just considered a part of the transport system.

Two other points might help the inclusion of the application make more sense. First, in the design of these original systems, there were two components: a terminal and a host. The terminal was really a display device; the application lived on the host. Second, the networking software was not thought of as a separate "thing" in the system; routers had not yet been invented, nor any other separate device to process and forward packets. Rather, a host was just connected to either a terminal or another host; the network software was just another application running on these devices.

Over time, as the OSI model came into more regular use, the DoD model was modified to include more layers. For instance, in Figure 3-2, a diagram replicated from a 1983 paper on the DoD model, there are seven layers (seven being a magic number for some reason).[1]

Here three layers have been added:

- The utility layer is a set of protocols living between the more generic transport layer and applications. Specifically, the Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and other protocols were seen as being a part of this layer.
- The network layer from the four-layer version has been divided into the network layer and the internetwork layer. The network layer represents the differing packet formats used on each link type, such as radio networks and Ethernet (still very new in the early 1980s). The internetwork layer unifies the view of the applications and utility protocols running on the network into a single internet datagram service.

1. Cerf and Cain, "The DoD Internet Architecture Model."

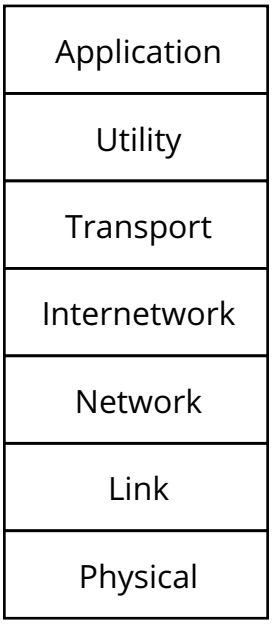


Figure 3-2 *A Later Version of the DoD Model*

- The link layer has been inserted to differentiate between the encoding of information onto the various link types and a device's connection to the physical link. Not all hardware interfaces provided a link layer.

Over time, these expanded DoD models fell out of favor; the four-layer model is the one most often referenced today. There are several reasons for this:

- The utility and application layers are essentially duplicates of one another in most cases. FTP, for instance, multiplexes content on top of the Transmission Control Protocol (TCP), rather than as a separate protocol or layer in the stack. TCP and the User Datagram Protocol (UDP) eventually solidified as the two protocols in the transport layer, with everything else (generally) running on top of one of these two protocols.
- With the invention of devices primarily intended to forward packets (routers and switches), the separation between the network and internetwork layers was overcome by events. The original differentiation was primarily between lower-speed long haul (wide area) links and shorter-run local area links; routers generally took the burden of installing links into wide area networks out of the host, so the differentiation became less important.

- Some interface types simply do not have a way to separate signal encoding from the host interface, as was envisioned in the split between the link and physical layers. Hence these two layers are generally munged into a single "thing" in the DoD model.

The DoD model is historically important because

- It is one of the first attempts to codify network functionality into a model.
- It is the model on which the TCP/IP suite of protocols (on which the global Internet operates) was designed; the artifacts of this model are important in understanding many aspects of TCP/IP protocol design.
- It had the concept of multiple protocols at any particular layer in the model "built in." This set the stage for the overall concept of narrowing the focus of any particular protocol, while allowing many different protocols to operate at once over the same network.

Open Systems Interconnect (OSI) Model

In the 1960s, carrying through to the 1980s, the primary form of communications was the switched circuit; a sender would ask a network element (a switch) to connect it to a particular receiver, the switch would complete the connection (if the receiver was not busy), and traffic would be transmitted over the resulting circuit. If this sounds like a traditional telephone system, this is because it is, in fact, based on the traditional network system (now called Plain Old Telephone Service [POTS]). Large telephone and computer companies were deeply invested in this model, and received a lot of revenue from systems designed around circuit switching techniques. As the DoD model (and its set of accompanying protocols and concepts) started to catch on with researchers, these incumbents decided to build a new standards organization that would, in turn, build an alternate system providing the "best of both worlds." They would incorporate the best elements of packet switching, while retaining the best elements of circuit switching, creating a new standard that would satisfy everyone. In 1977, this new standards organization was proposed, and adopted, as part of the International Organization for Standardization (ISO).

This new ISO working group designed a layered model similar to the proposed (and rejected) packet-based model, grounded in database communications. The primary goal was to allow intercommunication between the large database-focused systems dominant in the late 1970s. The committee was divided between telecom engineers and the database contingent, making the standards complex. The

protocols developed needed to provide for both connection-oriented and connectionless session control, and invent the entire application suite to create email, file transfer, and many other applications (remember, *applications are part of the stack)*. For instance, various transport modes needed to be codified to carry a wide array of services. In 1989—a full ten years later—the specifications were still not completely done. The protocol had not reached widespread deployment, even though many governments, large computer manufacturers, and telecom companies supported it over the DoD protocol stack and model.

But during the ten years the DoD stack continued to develop; the Internet Engineering Task Force (IETF) was formed to shepherd the TCP/IP protocol stack, primarily for researchers and universities (the Internet, as it was then known, did not allow commercial traffic, and would not until 1992). With the failure of the OSI protocols to materialize, many commercial networks, and networking equipment, turned to the TCP/IP protocol suite to solve real-world problems "right now."

Further, because the development of the TCP/IP protocol stack was being paid for under grants by the U.S. government, the specifications were free. There were, in fact, TCP/IP implementations written for a wide range of systems available because of the work of universities and graduate students who needed the implementations for their research efforts. The OSI specifications, however, could only be purchased in paper form from the ISO itself, and only by members of the ISO. The ISO was designed to be a "members only" club, meant to keep the incumbents firmly in control of the development of packet switching technology. The "members only" nature of the organization, however, worked against the incumbents, eventually playing a role in their decline.

The OSI model, however, made many contributions to the advancement of networking; for instance, the careful attention paid to Quality of Service (QoS) and routing issues paid dividends in the years after. One major contribution was the concept of clear modularity; the complexity of interconnecting many different systems, with many different requirements, drove the OSI community to call for clear lines of responsibility, and well-defined interfaces between the layers.

A second was the concept of machine-to-machine communication. Middle boxes, then called gateways, now called routers and switches, were explicitly considered part of the networking model, as shown in Figure 3-3.

You probably do not even need to see this image to remember the OSI model—everyone who's ever been through a networking class, or studied for a network engineering certification, is familiar with using the seven-layer model to describe the way networks work.

The genius of modeling a network in this way is it makes the interactions between the various pieces much easier to see and understand. Each pair of layers, moving vertically through the model, interacts through a socket, or Application

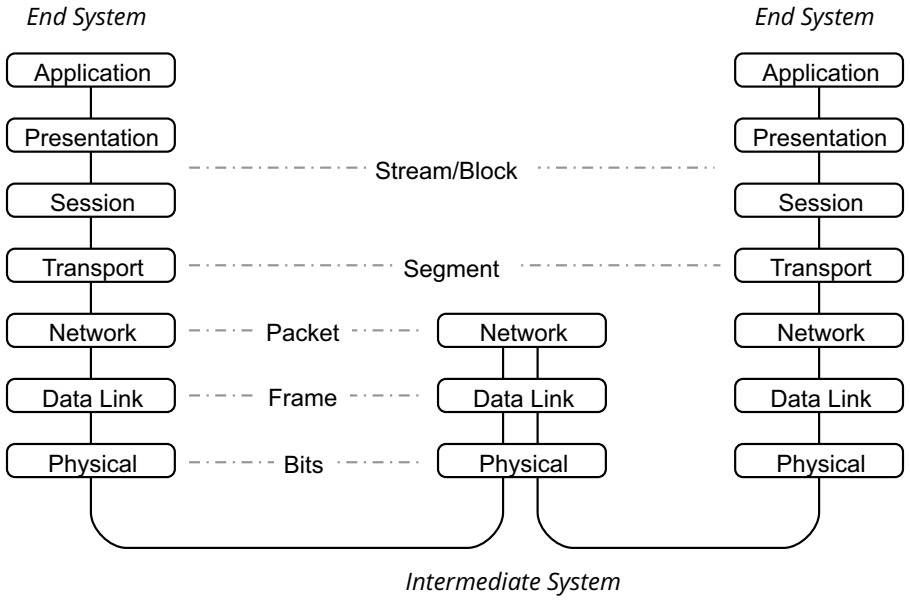


Figure 3-3 *The OSI Model, Including the Concept of an Intermediate System*

Programming Interface (API). So to connect to a particular physical port, a piece of code at the data link layer would connect to the socket for that port. This allows the interaction between the various layers to be abstracted and standardized. A piece of software at the network layer does not need to know how to deal with various sorts of physical interfaces, only how to get data to the data link layer software on the same system.

Each layer has a specific set of functions to perform.

The physical layer, also called layer 1, is responsible for getting the 0s and 1s modulated, or serialized, onto the physical link. Each link type will have a different format for signaling a 0 or 1; the physical layer is responsible for translating 0s and 1s into these physical signals.

The data link layer, also called layer 2, is responsible for making certain transmitted information is actually sent to the right computer connected to the same link. Each device has a different data link (layer 2) address that can be used to send traffic to a specific device. The data link layer assumes each frame within a flow of information is separate from all other frames within the same flow, and only provides communication for devices connected through a single physical link.

The network layer, also called layer 3, is responsible for transporting data between systems not connected through a single physical link. The network layer, then, provides networkwide (or layer 3) addresses, rather than link local addresses, and also

provides some means for discovering the set of devices and links that must be crossed to reach these destinations.

The transport layer, also called layer 4, is responsible for the transparent transfer of data between different devices. Transport layer protocols can be either be "reliable," which means the transport layer will retransmit data lost at some lower layer, or "unreliable," which means data lost at lower layers must be retransmitted by some higher layer application.

The session layer, also called layer 5, does not really transport data, but rather manages the connections between applications running on two different computers. The session layer makes certain the type of data, the form of the data, and the reliability of the data stream are all exposed and accounted for.

The presentation layer, also called layer 6, actually formats data in a way to allow the application running on the two devices to understand and process the data. Encryption, flow control, and any other manipulation of data required to provide an interface between the application and the network happen here. Applications interact with the presentation layer through sockets.

The application layer, also called layer 7, provides the interface between the user and the application, which in turn interacts with the network through the presentation layer.

Not only can the interaction between the layers be described in precise terms within the seven-layer model, the interaction between parallel layers on multiple computers can be described precisely. The physical layer on the first device can be said to communicate with the physical layer on the second device, the data link layer on the first device with the data link layer on the second device, and so on. Just as interactions between two layers on a device are handled through sockets, interactions between parallel layers on different devices are handled through network protocols.

Ethernet describes the signaling of 0s and 1s onto a physical piece of wire, a format for starting and stopping a frame of data, and a means of addressing a single device among all the devices connected to a single wire. Ethernet, then, falls within both the physical and data link layers (1 and 2) in the OSI model.

IP describes the formatting of data into packets, and the addressing and other means necessary to send packets across multiple data link layer links to reach a device several hops away. IP, then, falls within the network layer (3) of the OSI model.

TCP describes session setup and maintenance, data retransmission, and interaction with applications. TCP, then, falls within the transport and session layers (4 and 5) of the OSI model.

One of the more confusing points for engineers who only ever encounter the TCP/IP protocol stack is the different way the protocols designed in/for the OSI stack interact with devices. In TCP/IP, addresses refer to interfaces (and, in a world of networks with a lot of virtualization, multiple addresses can refer to a single interface,

or to an anycast service, or to a multicast data stream, etc.). In the OSI model, however, each device has a single address. This means the protocols in the OSI model are often referred to by the types of devices they are designed to connect. For instance, the protocol carrying reachability and topology (or routing) information through the network is called the *Intermediate System to Intermediate System* (IS-IS) protocol, because it runs between intermediate systems. There is also a protocol designed to allow intermediate systems to discover end systems; this is called the *End System to Intermediate System* (ES-IS) protocol (you did not expect *creative* names, did you?).

Note

It is one of the sad facts of network engineering history that proponents of the TCP/IP protocol suite developed an early dislike of the OSI protocol suite, to the point of rejecting the lessons learned in their development. While this has largely worn down into a rather more mild bit of fun in more recent years, the years lost to rejecting a protocol based on its origins, rather than its technical merits, are a lesson in humility in network engineering. Focus on the ideas, rather than the people; learn from everyone and every project you can; do not allow your ego to get in the way of the larger project, or solving the problem at hand.

Recursive Internet Architecture (RINA) Model

The DoD and OSI models have two particular focal points in common:

- They both contain *application layers;* this makes sense in the context of the earlier world of network engineering, as the application and network software were all part of a larger system.
- They combine the concepts of what data should be contained where with the concept of what goal is accomplished by a particular layer.

This leads to some odd questions, such as

- The Border Gateway Protocol (BGP), which provides routing (reachability) between independent entities (autonomous systems), runs on top of the transport layer in both models. Does this make it an application? At the same time, this protocol is providing reachability information the network layer needs to operate. Does this make it a network layer protocol?

- IPsec adds information to the Internet Protocol (IP) header, and specifies the encryption of information being carried across the network. Because IP is a network layer, and IPsec (sort of) runs on top of IP, does this make IPsec a transport protocol? Or, because IPsec run parallel to IP, is it a network layer protocol?

Arguing over these kinds of questions can provide a lot of entertainment at a technical conference or standards meeting; however, they also point to some amount of ambiguity in the way these models are defined. The ambiguity comes from the careful mixture of form and function found in these models; do they describe where information is contained, who uses the information, what is done to the information, or a specific goal that needs to be met to resolve a specific problem in transporting information through a network? The answer is—all of the above. Or perhaps, *it depends*.

This leads to the following observation: there are really only four functions any data-carrying protocol can serve: transport, multiplexing, error correction, and flow control. If these sound familiar, they should—because these are the same four functions uncovered in the investigation of human language in Chapter 2, "Data Transport Problems and Solutions."

There are two natural groupings within these four functions: transport and multiplexing, error and flow control. So most protocols fall into doing one of two things:

- The protocol provides transport, including some form of translation from one data format to another; and multiplexing, the capability of the protocol to keep data from different hosts and applications separate.
- The protocol provides error control, either through the capability to correct small errors or to retransmit lost or corrupted data; and flow control, which prevents undue data loss because of a mismatch between the network's capability to deliver data and the application's capability to generate data.

From this perspective, Ethernet provides transport services and flow control, so it is a mixed bag concentrated on a single link, port to port (or tunnel endpoint to tunnel endpoint) within a network. IP is a multihop protocol (a protocol that spans more than one physical link) providing transport services, while TCP is a multihop protocol that uses IP's transport mechanisms and provides error correction and flow control. Figure 3-4 illustrates the iterative model.

Each layer of the model has one of the same two functions, just at a different scope. This model has not caught on widely in network protocol work, but it provides a much simpler view of network protocol dynamics and operations than either

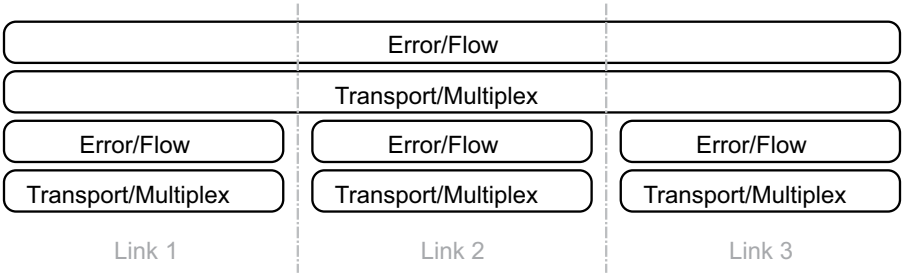


Figure 3-4 *The RINA Model*

the seven- or four-layer models, and it adds in the concept of scope, which is of vital importance in considering network operation. The scope of information is the foundation of network stability and resilience.

Connection Oriented and Connectionless

The iterative model also brings the concepts of connection-oriented and connectionless network protocols out into the light of day again.

Connection-oriented protocols set up an end-to-end connection, including all the state to transfer meaningful data, before sending the first bit of data. The state could include such things as the Quality of Service requirements, the path the traffic will take through the network, the specific applications that will send and receive the data, the rate at which data can be sent, and other information. Once the connection is set up, data can be transferred with very little overhead.

Connectionless services, on the other hand, combine the data required to transmit data with the data itself, carrying both in a single packet (or protocol data unit). Connectionless protocols simply spread the state required to carry data through the network to every possible device that might need the data, while connection-oriented models constrain state to only devices that need to know about a specific flow of packets. The result is single device or link failures in a connectionless network can be healed by moving the traffic onto another possible path, rather than redoing all the work needed to build the state to continue carrying traffic from source to destination.

Most modern networks are built with connectionless transport models combined with connection-oriented Quality of Service, error control, and flow control models. This combination is not always ideal; for instance, Quality of Service is normally configured along specific paths to match specific flows that should be following those paths. This treatment of Quality of Service as more connection oriented than

the actual traffic flows being managed causes strong disconnects between the ideal state of a network and various possible failure modes.

Final Thoughts

Knowing a number of models, and how they apply to various network protocols, can help you quickly understand a protocol you have not encountered before and diagnose problems in an operational network. Knowing the history of the protocol models can help you understand why particular protocols were designed the way they were, particularly the problems the protocol designers thought needed to be solved, and the protocols surrounding the protocol when it was originally designed. Different kinds of models abstract a set of protocols in different ways; knowing several models, and how to fit a set of protocols into each of the models, can help you understand the protocol operation in different ways, rather than a single way, much like seeing a vase in a painting is far different than seeing it in a three-dimensional presentation.

Of particular importance are the two concepts of connectionless and connection-oriented protocols. These two concepts will be foundational in understanding flow control, error management, and many other protocol operations.

The next chapter is going to apply these models to lower layer transport protocols.

Further Reading

Cerf, Vinton G., and Edward Cain. "The DoD Internet Architecture Model." *Computer Networks* 7 (1983): 307–18.

Day, J. *Patterns in Network Architecture: A Return to Fundamentals*. Indianapolis, IN: Pearson Education, 2007.

Grasa, Eduard. "Design Principles of the Recursive InterNetwork Architecture." In *3rd FIArch Workshop*. Brussels, 2011. http://www.future-internet.eu/fileadmin/documents/fiarch23may2011/06-Grasa_DesignPrinciplesOTheRecursiveInterNetworkArchitecture.pdf.

Maathuis, I., and W. A. Smit. "The Battle between Standards: TCP/IP Vs OSI Victory through Path Dependency or by Quality?" In *Standardization and Innovation in Information Technology, 2003. The 3rd Conference on*, 161–76, 2003. doi:10.1109/SIIT.2003.1251205.

Padlipsky, Michael A. *The Elements of Networking Style and Other Essays and Animadversions on the Art of Intercomputer Networking*. Prentice-Hall, 1985.

Russell, Andrew L. "OSI: The Internet That Wasn't." Professional Organization. *IEEE Spectrum*, September 27, 2016. https://spectrum.ieee.org/tech-history/cyberspace/osi-the-internet-that-wasnt.

White, Russ, and Denise Donohue. *The Art of Network Architecture: Business-Driven Design*. 1st edition. Indianapolis, IN: Cisco Press, 2014.

Review Questions

1. Research the protocols in the X.25 stack, which predates the three network models described in this chapter. Does the X.25 protocol stack show a layered design? Which layers of the DoD and OSI models does each protocol in the X.25 stack fit into? Can you describe each protocol in terms of the RINA model?
2. Research the protocols in the IBM Systems Network Architecture (SNA) stack, which predates the three network models described in this chapter. Does the SNA protocol stack show a layered design? Which layers of the DoD and OSI models does each protocol in the SNA stack fit in to? Can you describe each protocol in terms of the RINA model?
3. Billing is considered in some protocol stacks and models (such as the X.25 stack), and not in others. Why do you think this might be the case? Consider the way in which network utilization is used in the IP and X.25 stacks, specifically the use of bandwidth versus packets as a primary measurement system.
4. How does a layered network model contribute to the modularity of network protocol stacks?
5. How does a layered network model improve an engineer's understanding of how a network works?
6. Draw a diagram comparing the DoD and OSI models. Does each layer from one model fit neatly into the other?
7. Consider the OSI and RINA models; can you figure out which services from the RINA model fit into which layers in the OSI model?
8. Consider the connectionless versus connection-oriented models of protocol operation in light of the State/Optimization/Surface model, specifically in terms of state and optimization. Can you explain where adding state in a connection-oriented model increases optimal use of network resources? How does it decrease the optimal use of network resources?

9. In older network models, applications were often considered part of the protocol stack. Over time, however, applications seem to have been largely separated out of the network protocol stack, and considered as a "user" or "consumer" of network services. Can you think of a particular shift in the design of end hosts in relationship to the applications running on end hosts that would cause this shift in thinking in network engineering?

10. Do you think fixed length packets (or frames, or cells) make more sense from a protocol design perspective than variable length packets? How much state does a variable length packet format add compared to a fixed length format? How much optimization is gained? A useful point of departure for answering this question would be a list or chart of the average packet lengths carried through the global Internet.

This page intentionally left blank

Chapter 4

Lower Layer Transports

Learning Objectives

After reading this chapter, you should be able to:

- Understand the mechanism Ethernet uses to prevent collisions among multiple transmitters on a single physical wire
- Understand how physical addresses are assigned in an Ethernet network
- Understand half and full duplex modes of operation in Ethernet networks
- Understand the mechanism 802.11 WiFi uses to prevent collisions among multiple transmitters
- Understand the basic concepts of modulation, spatial multiplexing, and beamforming

Data transport protocols are often layered, with lower layers providing services along a single hop, a middle set of layers providing services end to end between two devices, and, potentially, a set of layers providing services end to end between two applications, or two instances of a single application. Figure 4-1 illustrates.

Each set of protocols is shown as a pair of protocols, because—as shown in the Recursive Internet Architecture (RINA) model in the previous chapter—transport protocols normally come in pairs, with each protocol in the pair taking on specific functions. This chapter will consider the physical and datalink protocols, as shown in Figure 4-1. Specifically, this chapter will consider two widely used protocols for point-to-point transport in networks: Ethernet and WiFi (802.11).

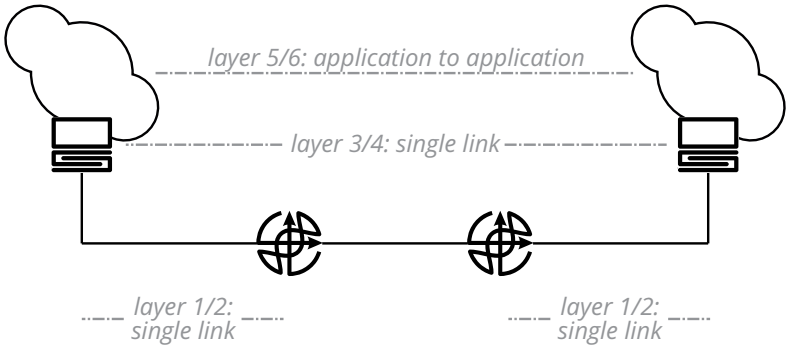


Figure 4-1 *Transport Layers*

Ethernet

Many of the early mechanisms designed to allow multiple computers to share a single wire were based on designs adopted from more telephone-oriented technologies. They generally focused on token passing and other more deterministic schemes for ensuring two devices did not try to use the single shared electrical medium at the same time. Ethernet, invented in the early 1970s by Bob Metcalf (who was working at Xerox at the time), resolved overlapping talkers in a different way—through a very simple set of rules to prevent the majority of overlapping transmissions, and then resolving any overlapping transmissions through detection and backoff.

The initial focus of any protocol that interacts with a physical medium is going to be in the area of multiplexing, as few other problems can be addressed until this first problem is solved. Therefore, this section will begin with a description of the multiplexing components of Ethernet and then move to other operational aspects.

Multiplexing

To understand the multiplexing problem Ethernet faced when it was first invented, consider the following problem: In a shared medium network, the entire shared medium is a single electrical circuit (or wire).

When one host transmits a packet, every other host on the network receives the signal. This is much like a conversation held in an open air environment; a sound

transmitted over the common medium (the air) is heard by every listener. There is no physical way to restrict the set of listeners during the transmission process.

CSMA/CD

The resulting system, called Carrier Sense Multiple Access with Collision Detection (CSMA/CD), operates using a set of steps:

1. The host *listens* on the medium to see if there are any existing transmissions in progress; this is the carrier sense part of the process.
2. On hearing there is no other transmission in progress, the host will begin serializing the bits in the frame onto the wire.

This part is simple—just listen before transmitting. It is possible, of course, for the transmissions of two (or more) hosts to collide as Figure 4-2 illustrates.

In Figure 4-2:

1. At time 1 (T1), A begins transmitting a frame onto the shared medium. It takes some amount of time for the signal to travel from one end of the wire to the other; this is called the propagation delay.

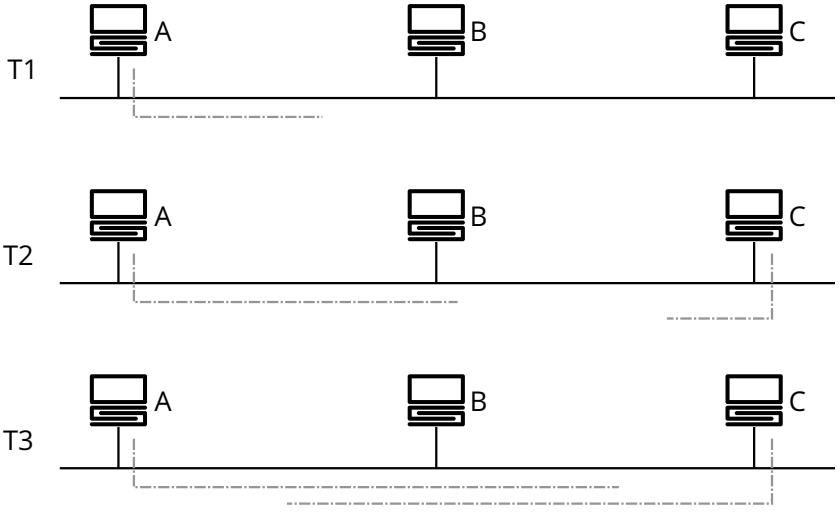


Figure 4-2 *Collisions in a Shared Medium*

2. At time 2 (T2), C listens for a signal on the wire, and, detecting none, begins transmitting a frame onto the shared medium. A collision has already occurred at this point, as both A and C are transmitting a frame at the same moment, but neither of them has yet detected the collision.
3. At time 3 (T3), the two signals actually collide on the wire, causing them both to be malformed, and hence unreadable.

A collision can be detected at A at the moment the signal from C reaches A by having A listen to its own signal as it is transmitted onto the wire. When the signal from C reaches A, A will receive the malformed signal caused by the combination of the two signals (the result of the collision). This is the *collision detection* portion (the CD portion) of CSMA/CD operation.

What should a host do when it detects a collision? In the original Ethernet design, the host will send a jam signal long enough to force any other host connected to the wire to sense the collision and stop transmitting. The length of the jam signal was originally set so the jam signal would consume at least the amount of time required to transmit a maximum-sized frame on the wire across the entire length of the wire. Why this specific amount of time?

- If a shorter than maximum frame was used in determining the amount of time the jam signal is transmitted, then a host with older interfaces (which cannot send and receive at the same time) may actually miss the entire jam signal while transmitting a single large frame, making the jam signal ineffective.
- It is important to allow enough time for the hosts connected at the very end of the wires to receive the jam signal, so they will sense the collision and take the following steps.

Once the jam signal is received, each host connected to the wire will set a back-off timer so they will each wait some random amount of time before attempting to transmit again. Because these timers are set to a random number, when the two hosts with frames waiting to be transmitted attempt their next transmission, the collision should not occur again.

If every host connected to the single wire receives the same signal at roughly the same time (given propagation delay through the wire), how does any particular host know whether it should actually receive a particular frame (or rather, copy the information within a frame from the wire to local memory)? This is the job of Media Access Control (MAC) addresses.

Each physical interface is assigned (at least) one MAC address. Each Ethernet frame contains a source and destination MAC address; the frame is formatted so

the destination MAC address is received before any data. Once the entire destination MAC address has been received, a host can decide whether it should continue receiving the packet or not. If the destination address matches the interface address, the host continues copying information off the wire and into memory. If the destination address does not match the local interface address, the host simply stops receiving the packet.

Ethernet Chipsets, Broadcast, Multicast, and Promiscuous Mode

The basic functionality of Ethernet requires Ethernet chipsets not only be able to receive traffic to the local MAC address, but also the ability for an Ethernet chipset to receive traffic to other MAC addresses. For instance, an Ethernet chipset should also accept any packets transmitted to a broadcast address, as frames transmitted to the broadcast address are intended for *every device connected to the physical medium*. A second class of addresses Ethernet chipsets must be able to accept are multicast addresses. A specific range of MAC addresses are set aside for use as multicast addresses; each address in this range is called a group, because it can be used to send a single frame to a group of *listeners*.

Each Ethernet chipset, then, is designed with a programmable set of addresses the chipset can be instructed to listen to. If the set of addresses the chipset must listen to is larger than the space set aside to hold this table of addresses, the Ethernet chipset can be set to promiscuous mode, which means it will copy every Ethernet frame off the wire and into memory.

Why not just set all Ethernet chipsets to promiscuous mode, so every frame is received on every host? Because this transfers the problem of determining which frames need to be locally processed from the Ethernet chipset (in hardware) to the software that processes these frames once the frame is completely copied into memory. Copying every frame off the wire, and pushing the matching job to software, has several negative side effects:

- The matching software must run someplace; this someplace is almost always the general processor that is also running all the rest of the software on the system. Determining which frames need to be processed locally or not on the main processor takes away cycles from other processes, slowing down the entire operation of the system and software.

- Frames copied off the wire must be stored someplace while they are waiting to be processed; this will generally be some kind of memory. Memory used to store packets waiting to be processed will not be available for use by other applications. Again, this can have a negative performance impact on the overall system.
- Determining which frames need to be processed locally in software is much slower than making this determination in hardware. Because of this, the rate at which a system that copies every frame into memory can process packets will be much lower than a system that does even some of this matching in hardware. Matching packets in software will also reduce the efficiency of the network interface on the host.

Given these disadvantages, does it ever make sense for a device to configure the Ethernet chipset in promiscuous mode? There are at least two cases where it does make sense:

- When a device is a router, which means it must receive most of the traffic transmitted on a particular interface to determine if each received packet needs to be forwarded. The operation of routers will be considered more fully in Chapter 7, "Packet Switching."
- When a device is monitoring the network. In this case, every packet must be captured to see all the traffic flowing across the wire. These kinds of devices are often used to troubleshoot network operation, protocol operation, or monitor network performance.

What about duplicate MAC addresses? If multiple hosts connected to the same medium have the same physical address, they would each receive, and potentially process, the same frames. There are ways to detect duplicate MAC addresses, but these are implemented as part of interlayer discovery rather than Ethernet itself; these will be considered in Chapter 6, "Interlayer Discovery." Ethernet itself assumes either

- MAC addresses will be properly assigned by the system administrator, if they are manually assigned.
- MAC addresses will be assigned by the device manufacturer so duplicate MAC addresses never occur, no matter how many hosts are connected to one another.

> **Note**
>
> Because MAC addresses are normally rewritten at every router (see Chapter 7 for more information), they only need to be unique within the segment or broadcast domain. While many older systems strove to ensure per segment or broadcast domain uniqueness, this must normally be enforced through manual configuration, and hence has largely been abandoned in favor of attempting to provide each device with a globally unique MAC address "baked into" the Ethernet chipset when it is created.

The first solution is difficult to implement in most large-scale networks; manual configuration of MAC addresses is extremely rare in the real world to the point of nonexistence. The second option essentially means MAC addresses must be assigned to individual devices so no two devices in the world share the same MAC address. How is this possible? By assigning MAC addresses out of a central repository managed through a standards organization. Figure 4-3 illustrates.

The MAC address is broken up into two sections: an Organizationally Unique Identifier (OUI) and a *network interface identifier*. The network interface identifier is assigned by the manufacturer of the Ethernet chipset. Companies producing Ethernet chipsets, in turn, are assigned the organizational identifiers by the Institute of Electrical and Electronic Engineers (the IEEE). So long as an organization (or manufacturer) always assigns addresses to a chipset with its OUI in the first three octets of the MAC address, and does not assign any two devices the same network interface identifier in the last three octets of the MAC address, *no two MAC addresses should be the same for any Ethernet chipset*.

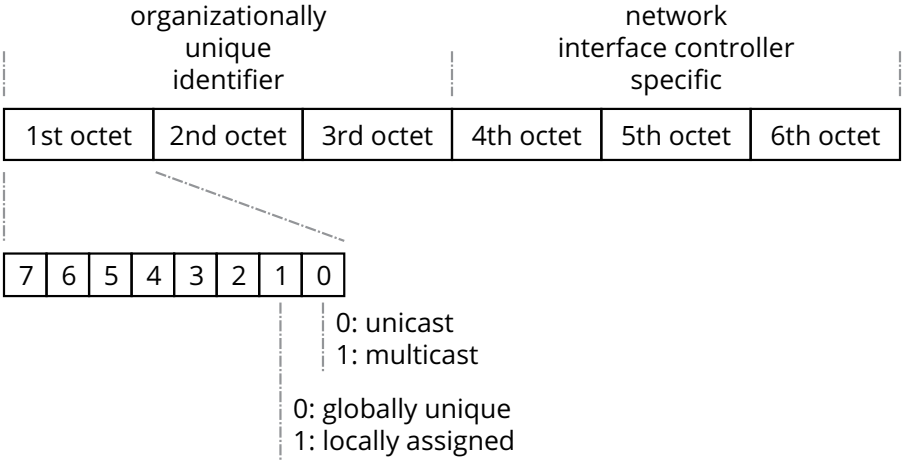


Figure 4-3 *MAC-48/EUI-48 Address Format*

Two bits within the OUI space are set aside to signal whether the MAC address has been locally assigned (which means the manufacturer's assigned MAC address has been overridden by the device's configuration), and whether the MAC address is intended as one of the following:

- Unicast address, which means it describes a single interface
- Multicast address, which means it describes a group of receivers

The MAC address consists of 48 bits; with these two bits removed, the MAC address space is 46 bits, which means it can describe 2^{46}—or 70,368,744,177,664—addressable interfaces. Because this is (potentially) not enough to account for the rapid number of new addressable devices, such as Bluetooth headsets and sensors, the length of a MAC address was increased to 64 bits to create the EUI-64 MAC address, which is constructed in the same way as the shorter 48-bit MAC address. These addresses can support 2^{62}—or 4,611,686,018,427,387,904—addressable interfaces.

The End of CSMA/CD

The shared medium model of Ethernet deployment has largely (though not completely!) been replaced in most networks. Rather than a shared medium, most Ethernet deployments now are switched, which means the single electrical circuit, or the single wire, is broken up into multiple circuits by connecting each device to a port on a switch. Figure 4-4 illustrates.

In Figure 4-4, each device is connected to a different set of wires, all of which terminate in a single switch. If the network interfaces at the three hosts (A, B, and C), and the switch network interfaces, can send or receive at any moment in time, rather than being able to do both, it is possible for A to send while the switch is also sending. In this case, the CSMA/CD process must still be followed in order to prevent collisions, even on networks where only two transmitters are connected to the same wire. This mode of operation is called half duplex.

If the Ethernet chipsets can both listen and transmit at the same time in order to detect collisions, however, this situation can be changed. The easiest way to manage this is to place the receive and transmit signals on *different physical wires* within the set of wires used in the Ethernet cable. Using different wires means there is no way for the transmissions from the two connected systems to collide, so the chipset can both transmit and receive at the same time. To enable this mode of operation, called full duplex, twisted pair Ethernet carries the signal in one direction on one pair of wires, and the signal in the opposite direction on another set of wires. In this case, CSMA/CD is no longer needed.

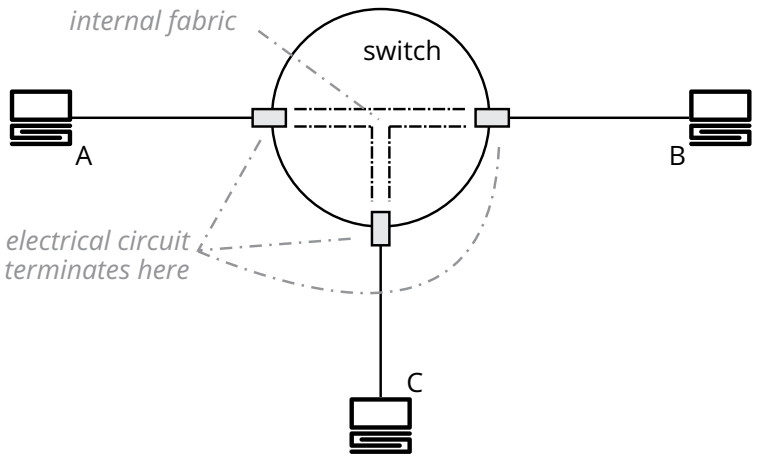


Figure 4-4 *Switched Ethernet Network Operation*

> **Note**
>
> The switch must learn which device (host) is connected to each port for this system to work; learning about the reachable destinations in a switched network is considered in Chapter 15, "Distance Vector Control Planes."

Error Control

CSMA/CD is designed to prevent one kind of detectable error in Ethernet: when collisions cause a frame to be malformed. Other kinds of errors can slip into a signal, however, as with any other electrical or optical system. For instance, in a twisted pair cabling system, if the twisted wires are "unwound" too much in installing a connector, one wire can transfer its signal to another wire through magnetic interference, causing cross talk. As a signal travels down a wire, it can reach the other end of the wire and reflect back along the length of the wire, as well.

How does Ethernet control for these errors? The original Ethernet standard included a 32-bit Cyclic Redundancy Check (CRC) in each frame, which can detect a large array of errors in transmission, as noted in Chapter 2, "Data Transport Problems and Solutions." At higher speeds, and on optical (rather than electrical) transport mechanisms, however, CRC can fail to detect enough errors to impact the operation of the protocol. To provide better error control, later (and faster) Ethernet standards have included more robust error control mechanisms.

For instance, Gigabit Ethernet specifies an 8B10B encoding scheme designed to ensure the correct synchronization of sender and receiver clocks; this scheme also detects some bit errors, as well. Ten-gigabit Ethernet is often implemented in

hardware with a Reed-Solomon code Error Correction (EC) system and a 16B18B encoding system, which provides good Forward Error Correction (FEC) and clock synchronization with 18% overhead.

> **Note**
>
> The 8B10B encoding scheme attempts to ensure there are approximately the same number of 0 and 1 bits in a data stream, which allows for efficient laser utilization and provides for clock synchronization to be embedded in the signal. The scheme works by encoding 8 bits of data (8B) into 10 transmitted bits on the wire (10B), which means there is about 25% overhead for each character transmitted. Single bit parity errors can be detected and corrected because the receiver knows how many 0s and 1s should have been received.

Data Marshaling

Ethernet transmits data in *packets* and *frames;* the packet is made up of the preamble information, the frame, and any trailing information. The frame contains a header, which is made up of fixed length fields, and the data being carried. Figure 4-5 illustrates an Ethernet packet; the frame is marked out as well.

In Figure 4-5, the preamble contains a beginning of frame marker, information the receiver can use to synchronize its clock to synchronize to the incoming packet, and other information. The destination address is received immediately after the preamble, so the receiver can quickly decide whether to copy this packet into memory or not. The addresses, protocol type, and carried data are all part of the frame. Finally, any FEC information and other trailers are added onto the frame to make up the final section(s) of the packet.

The type field is of particular interest, as this provides the information for the next layer up—the protocol providing the information carried in the data

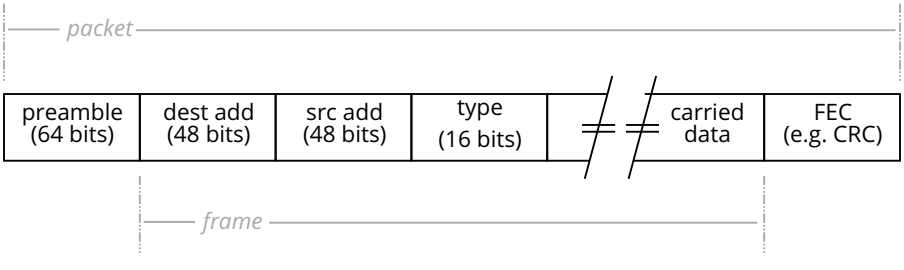


Figure 4-5 *An Ethernet Packet and Frame*

field—to identify the protocol. This information is opaque to Ethernet—the Ethernet chipset does not know how to interpret this information (only where it is), and how to carry it. Without this field there would be no consistent way for the carried data to be dispatched to the correct upper-layer protocol, or rather, for multiple upper-layer protocols to be properly multiplexed into Ethernet frames, and then properly demultiplexed.

Flow Control

In the original CSMA/CD implementation of Ethernet, the shared medium itself provided a sort of basic flow control mechanism. Assuming no two hosts can transmit at the same time, and information transmitted by some upper-layer protocol must be acknowledged or answered at least occasionally, the transmitter must periodically take a break to receive any acknowledgment or reply. There are sometimes situations where this rather rough form of flow control does not work; the Ethernet specification assumes some higher layer protocol will control the flow of information to prevent failures in this case.

In switched full duplex Ethernet, there is no CSMA/CD, as there is no shared medium. The two hosts connected to the pair of transmission channels can send data as quickly as the wires permit. This can, in fact, result in a situation where a host receives more data than it can process. To resolve this, a pause frame was developed for Ethernet. When a receiver sends the pause frame, the sender is supposed to stop sending traffic for a specified period of time.

Pause frames are not widely deployed.

Note

Many protocols do not contain all four of the functions described as part of the Recursive Internet Architecture (RINA) model described in Chapter 3, "Modeling Network Transport": error control, flow control, transport, and multiplexing. Even among those protocols implementing all four functions, all four are not always deployed. Normally, in this situation, the protocol and/or network designer is handing the function off to a lower or higher layer in the stack. This does work in some cases, but you should always be careful about assuming it is the correct thing to do. For instance, there is a difference between hop-by-hop encryption and end-to-end encryption. End-to-end is good for applications and protocols that do encrypt, but not every application does, in fact, encrypt data being transferred, nor does every host have an encrypted transport configured. In these cases, hop-by-hop encryption can be useful across less than secure links, such as wireless connections.

Wireless 802.11

Commonly called and marketed as *WiFi,* 802.11, which is widely deployed for carrying data over wireless in the unlicensed (in the United States) 2.4 and 5GHz radio spectrums. Microwave ovens, RADAR systems, Bluetooth, some amateur radio systems, and even baby monitors also use the 2.4GHz radio spectrums, so WiFi can both interfere with and be interfered with by these other systems.

Multiplexing

The 802.11 specifications generally use a form of frequency multiplexing to carry a large amount of information across a single channel, or set of frequencies. The frequency of a signal is simply the rate at which the signal switches polarity within a single second; hence a 2.4GHz signal is an electrical signal, carried across either a wire, an optical fiber, or the air, that switches polarity, from positive to negative (or negative to positive) 2.4×10^9 times per second.

> **Note**
>
> These are bare minimum descriptions; there is an entire field of radio and wave propagation you can study if you are so inclined; the goal here is to give you enough information to understand the basic concepts without overwhelming.

To understand the concept of wireless signaling, it is best to begin with the idea of carrier and modulation; Figure 4-6 illustrates.

In Figure 4-6, a single center frequency is chosen; the channel will be a range of frequencies on either side of this center frequency. Within the resulting channel, two carrier frequencies are chosen so they are orthogonal to one another—which means

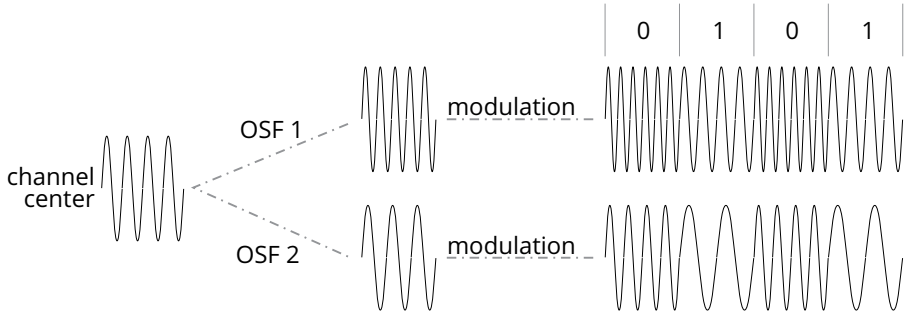


Figure 4-6 *Orthogonal Channels Using Frequency Modulation*

signals carried on these two carrier frequencies will not interfere with one another. These are marked as *OSF 1* and *OSF 2* in the figure. Each of these carrier frequencies is, in turn, actually a narrower channel, allowing the actual signal of 0s and 1s to be modulated onto the channel. Modulation, in this case, means varying the actual frequency of the signal around each OSF frequency.

Note

Modulation simply means somehow modifying the carrier in a way that allows a signal to be carried so a receiver can reliably decoded it.

Thus, the 802.11 specification uses an Orthogonal Frequency Division Multiplexing (OFDM) scheme, and encodes the actual data using Frequency Modulation (FM).

Note

One of the confusing points about multiplexing is it has two meanings, rather than one. Either it means to place multiple bits on the same medium at once, or it means allowing multiple hosts to communicate using the same medium at once. Which of these two meanings is intended can only be understood in a specific context. In this section, the meaning is the first, breaking a single medium up into channels to allow multiple bits to be transmitted at once. In most of the rest of this book, it means the second, allowing multiple hosts to transfer data over the same medium.

The speed at which data can be transmitted on such a system (the bandwidth) depends directly on the width of each channel and the ability of the transmitter to select orthogonal frequencies. To increase the speed of 802.11, then, two different techniques have been applied. The first is simply to increase the channel width, so more carrier frequencies can be used to carry data. The second is to find more efficient ways to pack data into a single channel by using more complex modulation methods. For instance, 802.11b can use a 40MHz wide channel in the 2.4GHz range, while 802.11ac can use either an 80 or 160MHz wide channel in the 5GHz range.

Spatial Multiplexing

Other forms of multiplexing to gain more bandwidth between two devices are also used in the 802.11 specification series. The 802.11n specification introduced Multiple Input Multiple Output (MIMO) antenna arrays, which allow the signal to follow different paths through the single medium (air). This might seem impossible, as there is only one "air" in a room, but wireless signals actually bounce off different objects within a room, which causes them to take multiple paths through the space. Figure 4-7 illustrates.

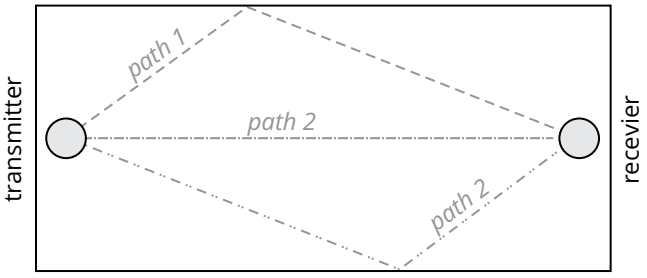


Figure 4-7 *Multiple Paths within a Single Room*

In Figure 4-7, assuming the transmitter is using an antenna that will transmit in all directions (an omnidirectional antenna), there are three paths through the single space, labeled 1, 2, and 3. The transmitter and receiver cannot "see" the three separate paths, but they can measure the strength of signal between each pair of antennas, and try sending different signals between apparently separated pairs until they find multiple paths over which different sets of data can be sent.

A second way multiple antennas can be used is in beamforming. Normally, a wireless signal transmitted from an antenna covers a circle (a ball in three dimensions, but this is difficult to meaningfully illustrate). In beamforming, the beam is shaped using one of various techniques to make it more oblong. Figure 4-8 illustrates these concepts.

In the unformed pattern, the signal is roughly a ball or globe around the tip of the antenna; drawn from the top, it looks much like a simple circle extending to the farthest point in the ball shape. By using a reflector, the beam can be shaped, or formed, into a more oblong shape. The space behind the reflector, and to the sides of the beam, will receive less (or even none, for very tight beams) of the transmission power. How can such a reflector be built? The simplest way is with a physical barrier tuned to repel the signal's power, much like a mirror reflects light, or a wall reflects sound. The key is the point in the transmission's signal the physical barrier is placed. Figure 4-9 will be used to explain the key points in the waveform, reflection, and cancellation.

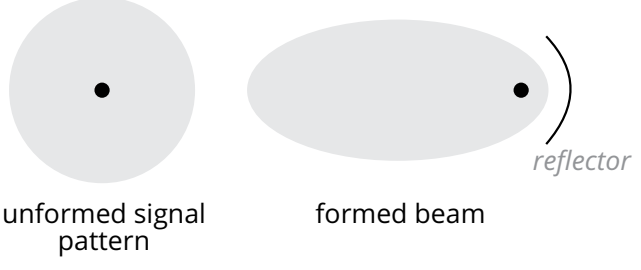


Figure 4-8 *Beam Formation*

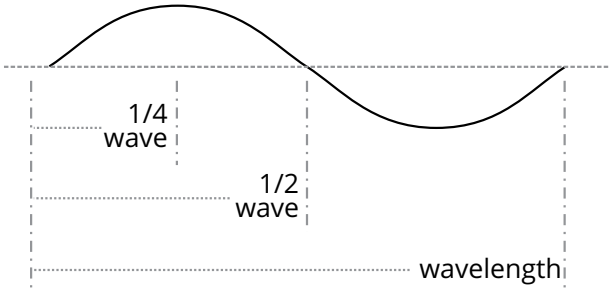


Figure 4-9 *A Signal Waveform*

A typical waveform follows a sine wave, which begins at zero power, increases to its maximum positive power, then moves back to zero power, and then through a positive negative power cycle. Each of these is a cycle; the frequency refers to the number of times this cycle repeats per second. The entire length of the wave in space, along a wire, or an optical fiber, is called the wavelength. The wavelength is inversely proportional to the frequency; the higher the frequency, the shorter the wavelength.

The key point to note in this diagram is the state of the signal at the quarter and half wavelength points. At the quarter wave point, the signal is at its highest power; if an object, or another signal, interferes at this point, the signal will either be absorbed or reflected. At the half wave point, the signal is at the minimum power; if there is no offset, or constant voltage on the signal, the signal will reach zero power. To reflect a signal, then, you can position a physical object so it reflects the power just at the quarter wave point. The physical distance required to do this will, of course, depends on the frequency, just as the wavelength depends on the frequency.

Physical reflectors are easy; what if you want to be able to dynamically form the beam without using a physical reflector? Figure 4-10 illustrates the principles you can use here.

The light gray dotted lines in Figure 4-10 provide a phase marker; two signals are in phase if their peaks are aligned, as shown on the left. The two signals shown in the middle are a quarter out of phase, as the peak of one signal is aligned with the zero point, or minimum, of the second signal. The third pair of signals, shown on the far right, are complementary, or 180 degrees out of phase, as the positive peak of one signal aligns with the negative peak of the second signal. The first pair of signals will add together; the third pair of signals will cancel out. The second pair may, if correctly crafted, reflect off one another. These three effects allow a beam to be formed, as shown in Figure 4-11.

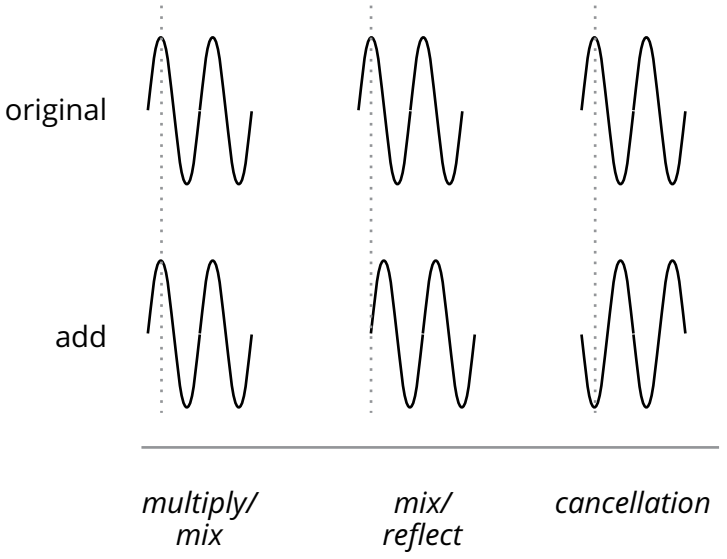


Figure 4-10 *Combinations of signals*

A single beamforming system may, or may not, use all of these components, but the general idea is to restrict the beam within a physical space within the medium—generally free air propagation. Beamforming allows the shared physical medium to be used as several different communication channels, as shown in Figure 4-12.

In Figure 4-12, the wireless router has used its beamforming capabilities to form three different beams, each directed at a different host. The router can now send traffic on all three of these formed beams at a higher rate than if it treated the entire space as a single shared medium, because the signals to A will not interfere or overlap with the information transmitted to B or C.

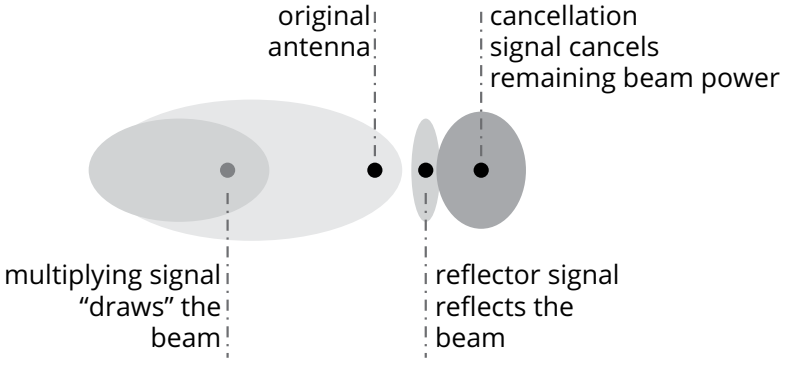


Figure 4-11 *Beamforming*

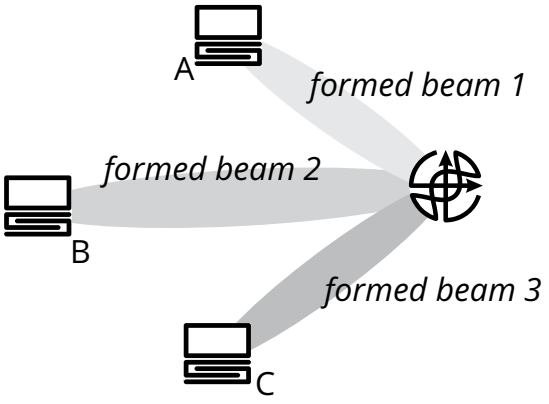


Figure 4-12 *Beamforming and Spatial Multiplexing*

Note

Directional methods such as beamforming only improve the traffic transmitted in one direction. For instance, if a wireless access point is capable of beamforming, and the host it is communicating with is not, the distance the two devices will be able to communicate across will be constrained by the host, as it cannot send a directional signal. However, the physical distance is not always the important point in beamforming technologies. The amount of information a wireless signal can carry is related to the power as well as other factors; the more power received at the receiver (not transmitted, received), the more information that can be transmitted. So if the access point can form a beam so it has twice as much power to the host as the host has connecting back to the access point, it will increase the speed at which the host can download data across the wireless link. It may, then, be worthwhile to have beamforming on one end of the wireless connection (and not the other). The answer to whether or not it is, is, as always, "it depends"—on the application, traffic pattern, and many other factors.

Channel Sharing

The multiplexing problem in wireless signals involves sharing a single channel, much like in wired network systems. Two specific problems dominate the solutions designed to share a single wireless medium: the hidden node problem and the transmission/reception power problem (which is also sometimes called receiver swamping). Figure 4-13 illustrates the hidden node problem.

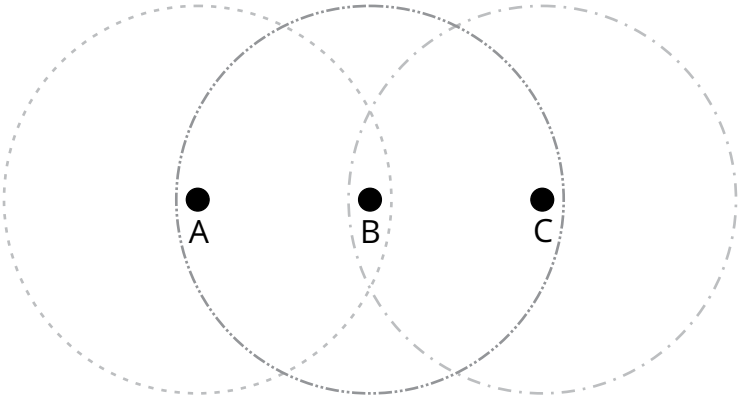


Figure 4-13 *The Hidden Node Problem in Wireless Networks*

The three circles in Figure 4-13 represent the three overlapping ranges of the wireless transmitters at A, B, and C. If A transmits toward B, C cannot hear the transmission. Even if C listens for a clear channel, it is possible for A and C to transmit at the same time, causing a collision at B.

The hidden node problem is made worse because of the power of transmission versus the power of the received signal, and the reality of air as a medium. A good rule of thumb for radio signal strength in air is the signal loses half of its power every wavelength of space it travels. At high frequencies, signals lose their strength very quickly, which means the transmitter must send a signal at a power orders of a magnitude larger than its receiver is capable of receiving.

It is very difficult to build a receiver able to "listen to" the local transmit signal at full strength without destroying the receive circuitry while also being able to "hear" the very low power signals required to extend device range. The transmitter, in other words, swamps the receiver with enough power to destroy the receiver in many situations. This makes it impossible, in a wireless network, for a transmitter to listen to the signal as it is being transmitted, and hence makes the collision detection mechanism used in Ethernet (for instance) impossible to implement.

The mechanism used by 802.11 to share a single channel among multiple transmitters must avoid the hidden channel and receiver swamping problems. 802.11 WiFi uses Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) to negotiate channel usage. CSMA/CA is similar to CSMA/CD:

1. Before transmitting, the sender listens to determine if another device is transmitting.

2. If another transmission is heard, the sender backs off for some random period of time before attempting again; this random backoff is designed to prevent several devices from hearing the same transmission, and all trying to transmit again at the same time at some point in the future.

3. If no other transmission is heard, the sender transmits the entire frame; it is impossible for the sender to receive the signal it is transmitting, so there is no way to detect a collision at this point.

4. The receiver sends an acknowledgment for the frame on receipt; if the sender does not receive an acknowledgment, it will assume a collision has occurred, back off for a random amount of time, and resend the frame.

Some WiFi systems can also use a Request to Send/Clear to Send (RTS/CTS) system. In this case:

1. The sender transmits an RTS.

2. When the channel is clear, and no other transmission is scheduled, the receiver sends a CTS.

3. On receiving the CTS, the sender transmits the data.

Which system will produce higher bandwidth depends on the number of senders and receivers using the channel, the length of the frames, and other factors.

Data Marshaling, Error Control, and Flow Control

Data marshaling in 802.11 is similar to Ethernet; there is a set of fixed length header fields in each packet, followed by the transported data, and finally a four-octet Frame Check Sequence (FCS), which contains a CRC over the contents of the packet. If the receiver can correct an error based on the CRC information, it will do so; otherwise, the receiver simply does not acknowledge receipt of the frame, which will lead to the frame being retransmitted by the sender.

A sequence number is included in each frame, as well, to ensure packets are received and processed in the order in which they were transmitted. Flow control is provided in the RTS/CTS system by the receiver waiting to send a CTS until it has enough clear buffer space to receive a new packet.

Goodput versus Throughput

The capacity of most links is measured in bandwidth, but bandwidth is only one determinant of the amount of data that can be pushed through a link. A second tier of factors to consider is flow control and channel efficiency. For instance, in Ethernet, if a collision takes place, the channel efficiency will be decreased slightly, and hence the total amount of transmitted data will be lower than the available theoretical bandwidth. This second measure, the total amount of data that can actually be transmitted on a particular link or channel, is often called the throughput. It is measured, like bandwidth, in a number of bits per second. There is a third set of variables to consider in actual data transmitted through a channel: the amount of throughput any headers, including error correction codes, take up. The amount of user data actually transmitted across a link is called the goodput, and is generally a good bit lower than the bandwidth. For instance, for a 54Gb/s 802.11 link, the actual goodput may in the 22Gb/s range. This goodput is the number you need to pay attention to when designing speeds and feeds in a network; the bandwidth will give you a theoretical maximum, but if you plan a 54Gb/s link to handle a stream containing 50Gb/s of data, you might be disappointed in the results.

Final Thoughts on Lower Layer Transmission Protocols

Lower layer transmission protocols tend to be dominated by physical concerns, such as how can a host know when to access the channel, and how can the channel be most efficiently used? The four elements are still important to consider. Multiplexing, for instance, still requires addresses to determine which host a particular frame is being transmitted to. In other words, multiplexing contains addressing *and* other solutions designed to solve problems found only when interacting with physical channels.

Many of the solutions in these lower layer protocols are also assumed to be a "first line of defense," rather than "the only line of defense." Error control in the physical layer tends to be simpler than mechanisms implemented in higher layers, which means these mechanisms are faster to check, but may also allow through

some number of errors that need to be detected and corrected at some higher layer. Flow control, in these layers, is focused on controlling traffic across a single link, and is often a side effect of channel access, rather than an explicit control mechanism.

Overall, the physical layer is the farthest from the application, and often gains the least amount of attention of the network designer; yet these protocols are still important, and they still follow the same problem and solution patterns that higher level protocols employ.

Further Reading

Correa, Colt, Charles M. Kozierok, Robert B. Boatright, Jeffrey Quesnelle, and Bob Metcalfe. *Automotive Ethernet—The Definitive Guide*. Intrepid Control Systems, 2014.

Gast, Matthew S. *802.11 Wireless Networks: The Definitive Guide*. 2nd edition. Beijing; Farnham: O'Reilly Media, 2005.

———. *802.11n: A Survival Guide: Wi-Fi Above 100 Mbps*. 1st edition. Sebastopol, CA: O'Reilly Media, 2012.

Geier, Jim. *Designing and Deploying 802.11 Wireless Networks: A Practical Guide to Implementing 802.11n and 802.11ac Wireless Networks for Enterprise-Based Applications*. 2nd edition. Indianapolis, IN: Cisco Press, 2015.

McLaughlin, Steven W., and David Warland. "Error Control Coding and Ethernet." presented at the IEEE 802.3 EFM Study Group, Portland, OR, July 10, 2017. http://www.ieee802.org/3/efm/public/jul01/presentations/mclaughlin_1_0701.pdf.

Metcalfe, Robert M., and David R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks." *Communications of the ACM* 19, no. 7 (July 1976): 395–404.

Perahia, Eldad, and Robert Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac*. 2nd edition. Cambridge, UK: Cambridge University Press, 2013.

Potter, Bruce, and Bob Fleck. *802.11 Security*. 1st edition. Sebastopol, CA: O'Reilly Media, 2002.

Rouzic, Jean-Pierre Le. *IEEE 802.11ac: An Analysis of the Standard*. CreateSpace Independent Publishing Platform, 2013.

Spurgeon, Charles E., and Joann Zimmerman. *Ethernet Switches: An Introduction to Network Design with Switches*. 1st edition. Beijing: O'Reilly Media, 2013.

———. *Ethernet: The Definitive Guide: Designing and Managing Local Area Networks*. 2nd edition. Beijing: O'Reilly Media, 2014.

Tropper, Carl. *Local Computer Network Technologies*. Elsevier, 2014.

Review Questions

1. How is optical modulation different from, or similar to, the electrical modulation described in the chapter?
2. On radio waves, you would normally use different channels at different frequencies to carry multiple signals over a single medium (such as the air in wireless networks, and even over wires in some wired networks). What mechanism is used to "channelize" an optical transmission media, and how does it work?
3. The chapter states multiplexing must often be solved before other problems in data transmission can be addressed. Why might this be?
4. Ethernet was originally designed to operate over thin and thick coax cable (10BASE5 and 10BASE2). Is it possible to enable full duplex operation over coax? Why or why not?
5. The chapter notes the pause frame is not widely deployed for Ethernet. Under what conditions would a pause frame be needed, and why would it not be widely deployed any longer?
6. Given audio is also a wave passing through air, it would make sense that the mix, multiply, and cancel interactions between signals in a different phase might also apply to audio engineering in a similar way. Find a site that explains the impacts of these same problems in audio design, and describe some solutions audio engineers use to solve these problems.
7. There are many ways to build directional signals. For instance, how does a dish-type antenna shape a wireless signal? A commonly used beamforming antenna type is the Log Periodic Dipole (LPD). How does this kind of antenna work? Would these kinds of antennas ever be useful in wireless networking?

8. Find the goodput for some wired and wireless link types. Are they radically different? Can you explain why?
9. The Ethernet specifications are designed to allow every device manufactured, worldwide, to have different MAC addresses. The chapter mentions older versions, and equipment, that required the network operator to configure Ethernet equipment with addresses manually. Can you describe, in terms of state, optimization, and surfaces, the tradeoffs between these two options? Try to find both positive and negative aspects of each possible way of solving the problem.
10. The chapter states an Ethernet chipset will be assigned at least one address, implying some chipsets may be assigned more than one. Describe at least two use cases where a single chip would need to have more than one MAC address.

This page intentionally left blank

Chapter 5

Higher Layer Data Transports

Learning Objectives

After reading this chapter, you should be able to:

- Understand the history and operation of the Internet Protocol (IP)
- Understand the purpose of IP
- Understand the concepts of IP addresses and aggregation
- Understand the structure of the IP header
- Understand the operation of the Transmission Control Protocol (TCP)
- Understand the basics of congestion control through sliding windows
- Understand the operation of QUIC
- Understand the purpose of the Internet Control Message Protocol (ICMP)

While the previous chapter considered two examples of point-to-point data transport over physical media, this chapter will consider four examples of end-to-end data transport. Figure 5-1 illustrates in terms of the Recursive Internet Architecture (RINA).

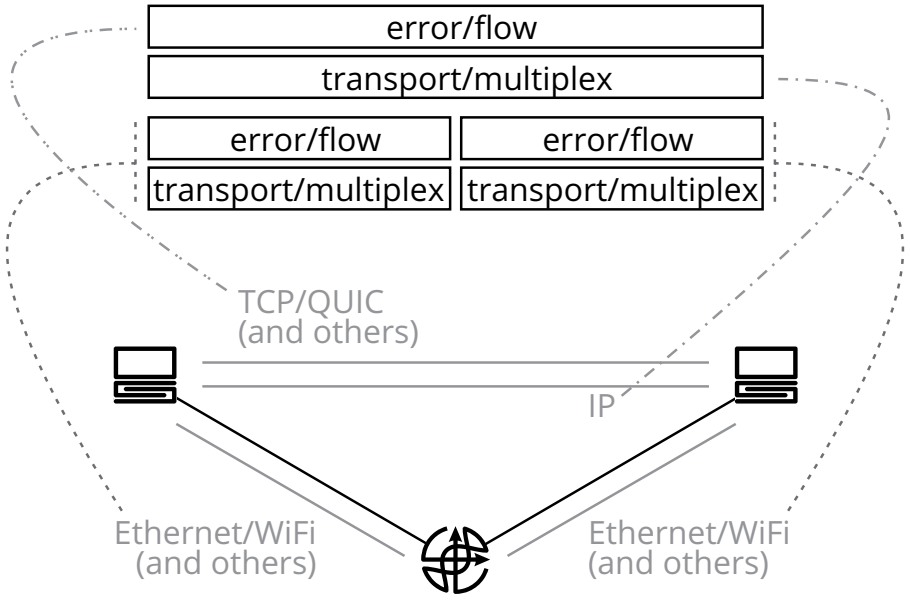


Figure 5-1 *Transport Protocol Examples, Protocol Function, and RINA*

Not every transport protocol maps precisely to a single functional layer in RINA, of course, but the mapping is close enough to be useful. The primary point to remember is—for each transport protocol, there are four questions you can ask:

- How does the protocol provide transport, or how does it marshal data?
- How does the protocol provide multiplexing services, or the ability to carry multiple streams of data on a single shared resource?
- How does the protocol provide error control, which should include not only error detection, but also resolving errors—either through retransmission or providing enough information to rebuild the original information?
- How does the protocol provide for flow control?

Each of these questions can have a number of subquestions, such as discovering the Maximum Transmission Unit (MTU), providing for replication of packets for multicast, etc.

This chapter will consider four protocols:

- The Internet Protocol (IP), which provides the bottom half of the second pair of layers. The primary focuses of IP are in the addressing scheme for multiplexing, and the ability to provide a single transport across many different physical transport systems.

- The Transmission Control Protocol (TCP), which provides one version of the top half of the second pair of layers. TCP provides error and flow control, as well as a place to carry multiplexing information for applications and other protocols that run on top of TCP.
- Quick User Datagram Protocol Internet Connections (QUIC), which provides another version of the top half of the second pair of layers. QUIC is much like TCP in its function, but has some significant differences from TCP in the way it operates.
- The Internet Control Message Protocol (ICMP).

The Internet Protocol

The Internet Protocol (IP) was originally documented in a series of Internet Protocol Specification documents called *IENs* in the middle of the 1970s, mostly written by Jonathan B. Postel. These documents described a protocol called TCP, which, when it was originally deployed, included the functionality contained in two protocols, IP and TCP. Postel noted this combination of functionality in a single protocol was not a good thing; in IEN #2, he states:

> We are screwing up in our design of internet protocols by violating the principle of layering. Specifically we are trying to use TCP to do two things: serve as a host level end to end protocol, and serve as an internet packaging and routing protocol. These two things should be provided in a layered and modular way. I suggest that a new distinct internetwork protocol is needed, and that TCP be used strictly as a host level end to end protocol. I also believe that if TCP is used only in this cleaner way it can be simplified somewhat. A third item must be specified as well—the interface between the internet host to host protocol and the internet hop by hop protocol.[1]

IEN #28, published in February of 1978, specified version 2 of this new Internet Protocol.[2] This was quickly replaced by IEN #48 in June 1978,[3] and again by IEN #54 in September of 1978.[4] In January 1980, IP became an IETF protocol with the publication of RFC760, which was also known as IEN #128,[5] and was updated with

1. Postel, "Comments on Internet Protocol and TCP," 1.
2. Postel, "Draft Internetwork Protocol Specification, Version 2."
3. Postel, "Internetwork Protocol Specification, Version 4," June 1978.
4. Postel, "Internetwork Protocol Specification, Version 4," September 1978.
5. "DoD Standard Internet Protocol."

the current specification, RFC791, in September of 1981.[6] At this point, the format of the IP version 4 (IPv4) header still in use today was in place.

> **Note**
>
> IPv4 is not covered in depth in this book; while it is widely deployed, version 6 of the IP protocol will be considered instead, as this is the protocol engineers will likely encounter more often in the future. In this spirit, all the examples in this book will use addresses in the version 6 format, as well. The "Further Reading" section lists resources of interest to readers who wish to learn more about IPv4.

The IPv4 address space is a 32-bit unsigned integer, which means it can number, or address, 2^{32} devices—about 4.2 billion devices. This sounds like a lot, but the reality is far different for several reasons:

- Each address represents one *interface,* rather than one *device*. In fact, IP addresses are often used to represent a *service,* or a *virtual host* (or *machine),* which means a single device will often consume more than one IP address.
- Large numbers of addresses are wasted in the process of aggregation.

In the early 1990s, it became obvious the Internet was going to run out of addresses in the IPv4 address space; charts like the one shown in Figure 5-2 show the available IPv4 address space over time starting in the mid-1990s.[7]

The easy solution to this situation would have been to extend the IPv4 address space to encompass some larger number of devices, but experience with the IPv4 protocol in the field led the Internet Engineering Task Force (IETF) to take on a larger task: to redesign IPv4. The work on the replacement began in 1990, with the first drafts achieving standard status in 1998. The IPv6 address space contains 2^{128} addresses, or around 3.4×10^{38}.

IPv6 is designed to provide services for several different protocols, such as TCP and QUIC, which are discussed in later sections in this chapter. As such, IPv6 provides only two services of the four required to carry data through a network: transport, which includes marshaling data, and multiplexing. These two functions are discussed in greater detail in the following sections.

6. "Internet Protocol."
7. Mro, *IPv4 Exhaustion*.

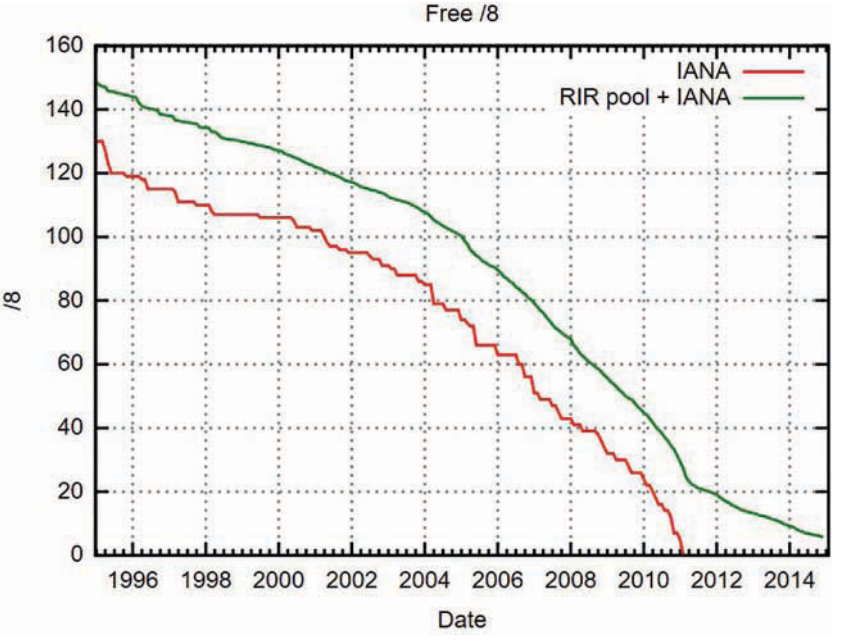


Figure 5-2 *IPv4 Address Space Usage over Time*

Transport and Marshaling

IP provides a "base layer" on which a wide array of higher layer protocols run, on many different kinds of physical links. To do so, IP must solve two problems:

- Run on a lot of different physical and lower layer protocols while presenting a consistent set of services to higher layers
- Adapt to the wide variety of frame sizes provided by lower layers

To create a single protocol on which all upper layer protocols can run, IP must "fit into" the frame type of many different kinds of physical layer protocols.

A series of drafts describe how to run IP on top of a particular physical layer, including MPEG-2 networks,[8] Asynchronous Transfer Mode,[9] optical networks,[10]

8. Fairhurst et al., *A Framework for Transmission of IP Datagrams over MPEG-2 Networks*.

9. Cole, Shur, and Villamizar, *IP over ATM: A Framework Document*.

10. Luciani, Rajagopalan, and Awduche, *IP over Optical Networks: A Framework*.

Point-to-Point Protocol (PPP),[11] the Vertical Blanking Interval (VBI) in television,[12] Fiber Distributed Data Interface (FDDI),[13] avian carriers,[14] and a number of other physical layer protocols (see the "Further Reading" section below). These drafts largely work out how to carry an IP datagram (or packet) in the frame (or packet) of the underlying physical layer, and how to enable interlayer discovery, such as the Address Resolution Protocol (ARP) to work on each media type (see Chapter 6, "Interlayer Discovery," for more information).

IP must also specify how to carry large blocks of data across the various MTUs available on different kinds of physical links. While the original Ethernet specification chose an MTU of 1,500 octets to balance between large packet sizes and maximum channel utilization, many other physical layers have been designed with larger MTUs. Further, applications do not tend to send information in neat, MTU-sized chunks. IP manages these two problems by providing for fragmentation; Figure 5-3 illustrates.

If an application (or higher-level protocol) passes 2,000 octets of data to be transmitted to IP, the IP implementation will

- Determine the MTU along the path through which the data must be transmitted; this is normally a matter of reading a configured or default value set by the system software

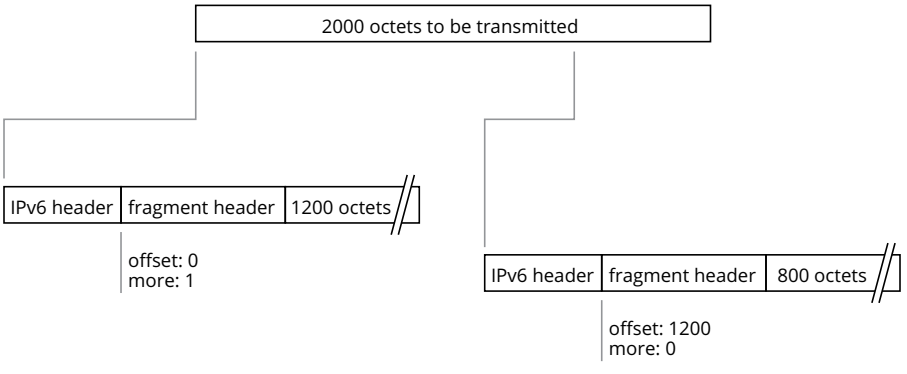


Figure 5-3 *Fragmentation in IPv6*

11. Varada, *IP Version 6 over PPP*.
12. Panabaker, Wegerif, and Zigmond, *The Transmission of IP Over the Vertical Blanking Interval of a Television Signal*.
13. Katz, *Transmission of IP and ARP over FDDI Networks*.
14. Waitzman, *Standard for the Transmission of IP Datagrams on Avian Carriers*.

- Break up the information into multiple fragments, based on the MTU minus the projected size of the headers, including tunnel headers, etc.—the metadata that must be transmitted along with the data
- Send the first fragment with an IPv6 optional header (which means the fragment header does not need to be included with packets that are not fragments of a larger data block)
- Set the offset in the more fragments header to the first octet in the original data block this packet represents divided by 8; in the example in Figure 5-3, the first packet has an offset of 0, while the second has an offset of 150 (1200/8).
- Set the more fragments bit to 0 if this is the last fragment of the data block, and 1 if there are more fragments to follow.

This size of the total data block IPv6 can carry through fragments is limited by the size of the *offset* field, which is 13 bits long. Hence, IPv6 can carry, at most, 2^{14} octets of data as a series of fragments, or a data block of about 65,536 octets plus one MTU-sized fragment. Anything larger than this would need to be broken up, in some way, by a higher layer protocol before being passed to IPv6 for transport.

Finally, IP must provide for some way to carry packets across a network that uses more than one type of physical layer. This is solved by rewriting the lower layer headers at each hop in the network where multiple media types might be interconnected. Devices that rewrite the lower layer headers in this way were originally called gateways, but are generally called routers now, because they route traffic based on the information contained in the IP header. Packet switching is considered in more detail in Chapter 7, "Packet Switching."

There are some other interesting aspects of the way IPv6 carries data; Figure 5-4 illustrates an IPv6 header to work from.

In Figure 5-4:

- The **version** is set to 6, for IPv6.
- The **traffic class** is divided into two fields, 6 bits for carrying the type of service (or service class), 2 bits for carrying congestion notification. Quality of Service (QoS) is considered in more detail in Chapter 8, "Quality of Service."
- The **flow label** is designed as a hint to tell forwarding devices how to keep packets within a single flow on the same path in an equal cost multipath (ECMP) set of paths.
- The **payload length** indicates the amount of data being carried in the packet in octets.

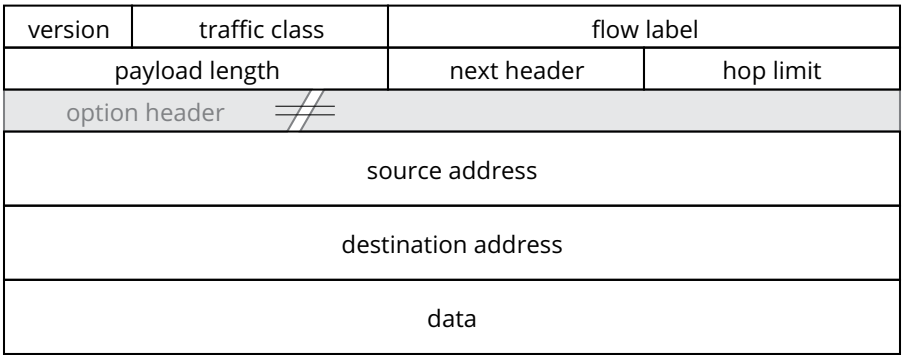


Figure 5-4 *The IPv6 Header Format*

- The **next header** provides information about any additional headers contained in the packet. The IPv6 header can contain information beyond what is contained in the basic header; these optional headers are discussed in more detail in a following section.
- The **hop limit** is the number of times this packet can be "handled" by a network device before being dropped. Any router (or other device) that rewrites the lower layer headers should decrement this number by one in the forwarding process; when the hop limit reaches 0 or 1, the packet should be discarded.

Note

The hop count is used to prevent a packet from looping in a network forever. Each time the packet is forwarded by a network device, the hop count is decremented by one. If the hop count reaches 0, the packet is discarded. If a packet is looping within the network, the hop count (also called a Time to Live, or TTL) will eventually be reduced to 0, and the packet will be dropped.

The IPv6 header is a mixture of variable (Type Length Value [TLV]) and fixed length information. The basic header is made up of fixed length fields, but the *next header* field leaves open the possibility of optional (or extension) headers, some of which are formatted as TLVs. This allows custom hardware (for instance, an Application-Specific Integrated Circuit [ASIC]) to be built to quickly switch packets based on the fixed length fields, while leaving open the possibility of carrying variable length data that might only be processed in software.

Multiplexing

IPv6 enables multiplexing in two different ways:

- By providing a large address space to use in identifying hosts and networks (or, more largely, reachable destinations)
- By providing a space into which the upper layer protocol can place a protocol number, which allows multiple protocols to run on top of IPv6

IPv6 Addressing

The IPv6 address is 128 bits, which means there can be up to 2^{128} addresses—a vast number of addresses, enough to perhaps number every grain of dust on the Earth. The IPv6 address is normally written as a series of hexadecimal numbers, rather than as a series of 128 0s and 1s, as shown in Figure 5-5.

Two points on zeros are worth noting in the IPv6 address format:

- Leading zeros in each section (set off by colons) are omitted.
- A single long string of zeros can be replaced by a double colon once in the address (not twice).

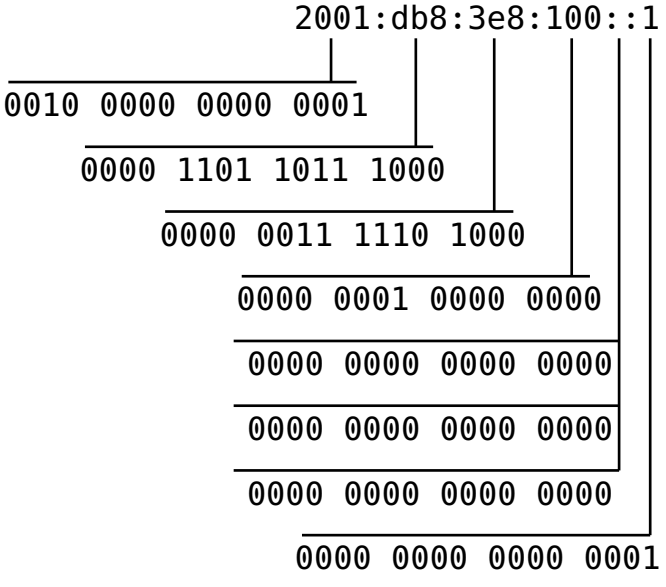


Figure 5-5 *IPv6 Address*

> **Note**
>
> When every address in the network begins with the same set of numbers, *sometimes* only the part that changes will be included to shorten the address, as well. For instance, in a network with 2001:db8:3e8:100::1 and 2001:db8:3e8:101::2, the two addresses may be referred to as 100::1 and 101::2, rather than repeating the entire address. You will need to fill in the remainder of the address from the context, such as a network diagram, or some earlier mention of the address, etc.

Why so many addresses? Because many addresses are never used in any addressing scheme.

First, many addresses are never used because addresses are aggregated. Aggregation is the use of a single prefix (or network, or reachable destination) to represent a larger number of reachable destinations; Figure 5-6 illustrates.

In Figure 5-6:

- Hosts A and B are given 101::1 and 101::2 as their IPv6 addresses. These two hosts are, however, connected to a single broadcast segment (such as Ethernet), and hence share the same interface at C. Even though C has an address on this shared network, it actually advertises the network itself—some engineers find it helpful to think of the *wire itself*—as a reachable destination: 101::/64.
- E receives two reachable destinations, 101::/64 from C and 102::/64 from D. By decreasing the prefix length, it can advertise a single reachable destination that includes both of these two *longer prefix* reachable destinations. E advertises 100::/60.
- G, in turn, receives 100::/60 from E, and 110:/60 from F. Again, this same address space can be described using a single reachable destination, 100::/56, so this is what G advertises.

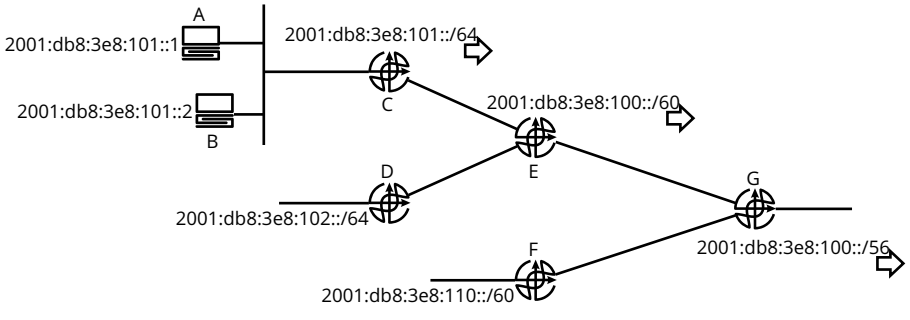


Figure 5-6 *Address Aggregation in IPv6*

How does this aggregation work in the actual address space? Figure 5-7 is used to explain.

The prefix length, which is the number after the slash in a reachable destination, tells you the number of bits that count in determining what is part of the prefix (and hence also what is not). The prefix length is counted from the left to the right. Any set of addresses with the same values in the numbers within the prefix length are considered to be part of the same reachable destination.

- There are 128 bits in the full IPv6 address space, so a /128 represents a single host.

Figure 5-7 *Aggregation and Reachable Destinations in IPv6 Addressing*

- In an address with a 64-bit prefix length (/64), only the left four sections of the IPv6 address are part of the prefix, or the reachable destination; the remainder, the four right sections of the IPv6 address, are assumed to either be host or subnetwork addresses that are "contained" in the prefix.
- In an address with a 60-bit prefix length (/60), the left four sections of the IPv6 address *minus one* hexadecimal digit are considered part of the reachable destination, or the prefix.
- In an address with a 56-bit prefix length (/56), the left four sections of the IPv6 address *minus two* hexadecimal digits are considered part of the reachable destination, or the prefix.

Note

So long as you always change the prefix length in increments of 4 (/4, /8, /12, /16, etc.), the significant digits, or the digits that are part of the prefix, will always move one to the right (as you increase the prefix length) or the left (as you decrease the prefix length).

Aggregation sometimes seems complicated, but it is an essential part of IP.

Some of the address space is consumed in autoconfiguration. While autoconfiguration is not covered in detail here, the interaction between autoconfiguration and IPv6 address assignment is important to consider. Some amount of address space must generally be set aside to ensure no two devices connected to the network will end up with the same identifier. In the case of IPv6, half of the address spaces (everything greater than a /64), within certain ranges of addresses, are set aside in order to form unique per device identifiers.

Third, some addresses are set aside for special use. For instance, in IPv6, the following address spaces are assigned to some special use:

- ::ffff/96 is set aside for IPv4 addresses that are "mapped into" the IPv6 address space.
- fc00::/7 is set aside for unique local addresses (ULAs); packets with these addresses are not intended to be routed on the global Internet, but rather kept within the network of a single organization.
- fe80::/10 is set aside for link local addresses; these addresses are automatically assigned on each interface, and are only used for communicating over a single physical or virtual link.

- ::/0 is set aside as a default route; if a network device does not know of any other way to reach a particular destination, it will forward traffic toward the default route.

Fourth, devices can be assigned multiple addresses. Many engineers tend to think of an address as if it describes a single host or system. In reality, a single address can be used to describe many things, including

- A single host or system
- A single interface on a host or system; a host with multiple interfaces would have multiple addresses
- A set of reachable services on a host or system; for example, a virtual machine or a particular service running on a host may be assigned an address that is different from any of the addresses assigned to the host's interfaces

There is no necessary direct correlation between an address and a physical device, or an address and a physical interface.

Multiplexing Between Processes

The second multiplexing mechanism is allowing multiple protocols to run over the same base layer. This form of multiplexing is provided through protocol numbers; Figure 5-8 illustrates.

The next header field either points to

- The next header in the IPv6 packet, if there is a next header
- A protocol number, if the next header is a transport protocol (such as TCP)

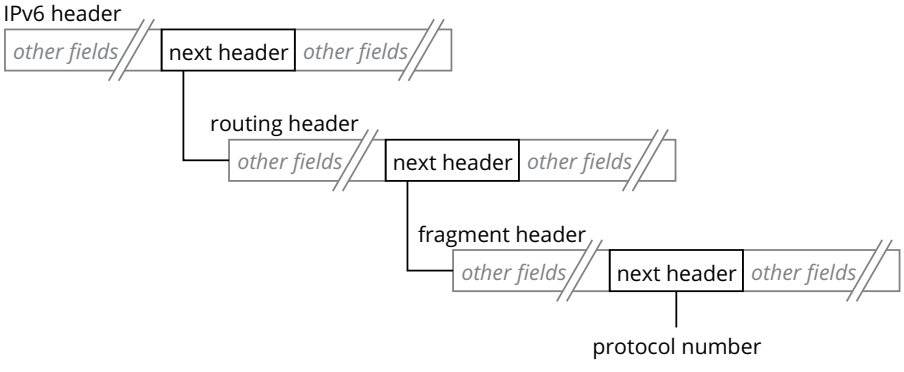


Figure 5-8 *The Protocol Number in IPv6*

These additional headers are called optional or extension headers; some of them are fixed length, and others are TLV based; for instance:

- **Hop-by-hop options:** A set of TLVs describing actions each forwarding device should take
- **Routing:** A set of fixed length route types used to indicate the path the packet should take through the network
- **Fragment:** A fixed length set of fields providing packet fragment information (as described above)
- **Authentication header:** A set of TLVs containing authentication and/or encryption information
- **Jumbogram:** An optional data length field enabling the IPv6 packet to carry up to one byte less than 4GB of data

The next header field is 8 bits long, which means it can carry a number between 0 and 255. Each number in this range is assigned either to a specific kind of option header or a specific higher layer protocol. For instance:

- **0:** The next header is an IPv6 hop-by-hop option.
- **1:** The packet payload is the Internet Control Message Protocol (ICMP).
- **6:** The packet payload is TCP.
- **17:** The packet payload is the User Datagram Protocol (UDP).
- **41:** The packet payload is IPv6.
- **43:** The next header is an IPv6 routing header.
- **44:** The next header is an IPv6 fragment header.
- **50:** The next header is an Encapsulated Security Header (ESH).

The protocol number is used by the receiving host to dispatch the contents of the packet to the correct local process for processing; normally, this means stripping the lower (physical) layer headers off the packet, placing the packet into the input queue for the correct process (such as TCP), and then notifying the operating system the relevant process needs to run.

Transmission Control Protocol

The primary goal of TCP is to provide what appears to be a connection-oriented transport on top of IP. As a higher layer protocol, it relies on the addressing and

multiplexing capabilities of IPv6 to carry information to the correct destination host. Because of this, TCP does not require an address scheme. The focus of TCP is on flow and error control, considered in separate sections below. A short section on TCP port numbers rounds out this discussion of TCP.

Flow Control

TCP uses a sliding window method to control the flow of information across each connection between two hosts; Figure 5-9 illustrates.

In Figure 5-9, assume the initial window size is set to 20. The sequence of events is then

- At *t1,* the sender transmits 10 packets or octets of data (in the case of TCP, it is 10 octets of data).
- At *t2,* the receiver acknowledges these 10 octets, and the window is set to 30. This means the sender is now allowed to send up to 30 more octets of data before waiting for another acknowledgment; in other words, the sender can send up to octet 40 before it must wait for an acknowledgment to send more data.

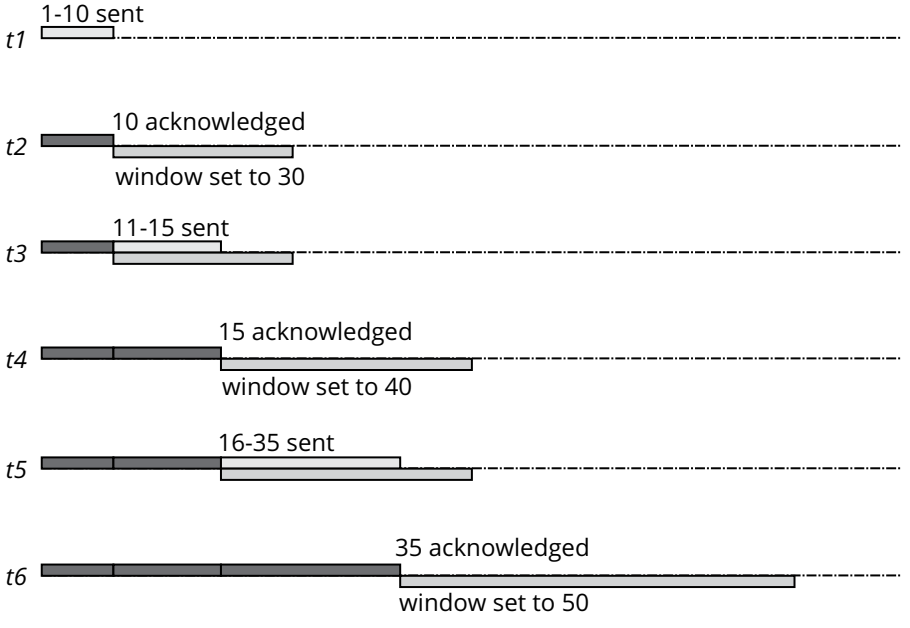


Figure 5-9 *A Sliding Window*

- At *t3,* the sender sends another 5 octets of data, numbers 11–15.
- At *t4,* the receiver acknowledges the receipt of the octets through 15, and the window is set to 40 octets.
- At *t5,* the sender sends about 20 octets of data, numbered 16–35.
- At *t6,* the receiver acknowledges 35 and the window is set to 50.

Several important points to note about this technique are as follows:

- When the receiver acknowledges receiving a particular piece of data, it implicitly also acknowledges receiving everything before this piece of data.
- If the receiver does not send an acknowledgment—say the transmitter sends 16–35 at *t5,* and the receiver does not send an acknowledgment—the sender will wait some period of time and assume the data never arrived, so it will retransmit the data.
- If the receiver acknowledges some of the data the sender has transmitted, but not all of it, the sender assumes some of the data is missing, and retransmits from the point the receiver has acknowledged. For instance, if the sender transmitted 16–35 at *t6,* and the receiver acknowledged 30, the sender should retransmit 30 and forward.
- The window is set at both the sender and the receiver; this is explained in more detail in a following section.

Instead of using octet numbers, TCP assigns each transmission a sequence number; when the receiver acknowledges a specific sequence number, the transmitter assumes the receiver has actually received all the octets of information up to the transmission with the sequence number. For TCP, then, the sequence number acts as a sort of "shorthand" for a set of octets. Figure 5-10 illustrates.

In Figure 5-10:

- At *t1,* the sender bundles octets 1–10 and transmits them, marking them as sequence number 1.
- At *t2,* the receiver acknowledges sequence number 1, implicitly acknowledging the receipt of octets 1–10.
- At *t3,* the sender bundles octets 11–15 together and transmits them, marking them as sequence number 2.
- At *t4,* the receiver acknowledges sequence number 2, implicitly acknowledging the octets sent through 15.

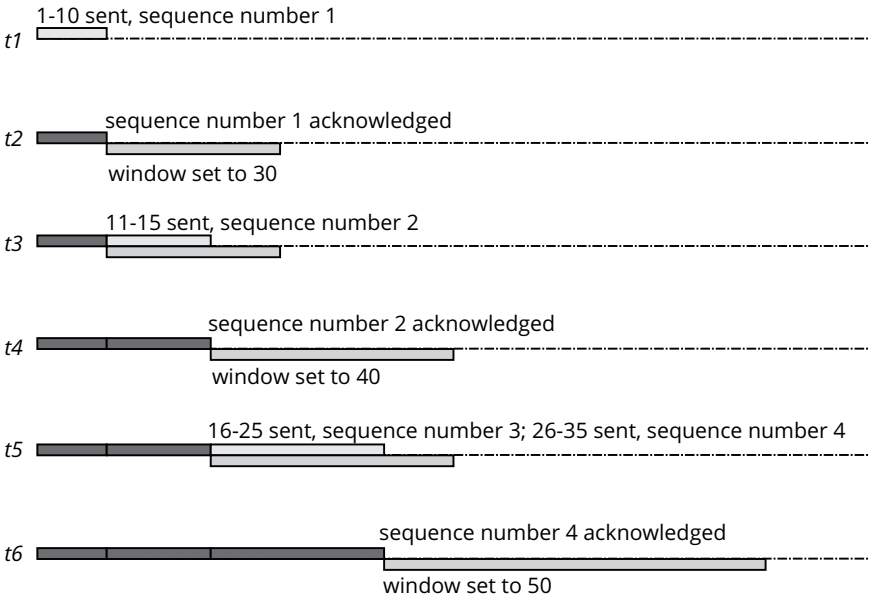


Figure 5-10 *Windowed Flow Control with Serial Numbers*

- At *t5*, assume 10 octets will fit into a single packet; in this case, the sender would send two packets, one containing 16–25, with the sequence number 3, and one containing octets 26–35, with sequence number 4.
- At *t6*, the receiver acknowledges sequence number 4, implicitly acknowledging all the previously transmitted data.

The sections that follow consider various questions in relation to the windowed flow control scheme used by TCP.

What Happens If One Packet of Information Is Missed?

What if the first packet out of a flow of 100 packets is not received? Using the system described in Figure 5-10, the receiver would simply not acknowledge this first packet of information, forcing the sender to retransmit the data sometime later. This is inefficient, however; each dropped packet of information requires a complete resend from that packet forward. TCP implementations use two different ways to allow a single packet to be requested by a receiver.

The *first way* is a triple acknowledgment. If a receiver acknowledges a packet that is earlier than the most recently acknowledged serial number *three times*, the sender assumes the receiver is asking for the packet to be retransmitted. Three repeated

acknowledgments are used to prevent out-of-order packet delivery, or dropped packets, from causing a false retransmit request.

The *second way* is to implement selective acknowledgments (SACK).[15] SACK adds a new field to the TCP acknowledgment that allows a receiver to acknowledge the receipt of a specific set of serial numbers, rather than assuming the acknowledgment of a single serial number acknowledges every lower serial number as well.

How Long Does the Transmitter Wait Before Retransmitting?

The first way in which a sender can detect a packet has been lost is through the Retransmit Time Out (RTO), which is calculated as a function of the Round Trip Time (RTT or rtt). The rtt is the time interval between the transmission of a packet by a sender and the receipt of an acknowledgment from the receiver. The rtt measures the delay through the network from the transmitter to the receiver, the processing time at the receiver, and the delay through the network from the receiver to the transmitter. Note the rtt can vary depending on the path each packet takes through the network, local conditions at the time the packet is switched, etc.

The RTO is normally calculated as a weighted average in which older rtts have less impact than more recent measured rtts.

An alternative mechanism used in most TCP implementations is **fast retransmit.** In fast retransmit, the receiver adds one to the expected sequence number in any acknowledgment. For instance, if a sender transmits sequence 10, the receiver acknowledges sequence 11, *even though it has not yet received sequence 11*. In this case, the sequence number in the acknowledgment acknowledges the receipt of data and indicates what sequence number it is expecting the sender to transmit next.

If the transmitter receives an acknowledgment with a sequence number that is one larger than the last acknowledged sequence number *three times in a row,* it will assume the packets following have been dropped.

There are, therefore, two *types* of packet loss in TCP when fast start is implemented. The first is a standard timeout, which occurs when the sender transmits a packet, and does not receive an acknowledgment before the RTO expires. This is called an RTO failure. The second is called a fast retransmit failure. These two conditions are often handled differently.

How Is the Window Size Chosen?

There are a number of different considerations in choosing a window size, but the dominant factor is often gaining the highest possible performance while avoiding link congestion. In fact, TCP congestion control is probably the primary form of

15. Floyd et al., *TCP Selective Acknowledgment Options*.

congestion control actually deployed in the global Internet. To understand TCP congestion control, it is best to begin with some definitions:

- **Receive Window (RWND):** The amount of data the receiver is willing to receive; this window is normally set based on the receiver's buffer size, or some other resource available at the receiver. This is the window size advertised in the TCP header.
- **Congestion Window (CWND):** The amount of data the transmitter is willing to send before receiving an acknowledgment. This window is not advertised in the TCP header; the receiver does not know the size of the CWND.
- **Slow Start Threshold (SST):** The CWND at which the sender considers the connection at its maximum packet rate without congestion occurring on the network. The SST is initially set by the implementation, and changed in the case of packet loss depending on the congestion avoidance mechanism being used.

Most TCP implementations begin sessions with a Slow Start algorithm.[16] In this phase, the CWND starts at 1, 2, or 10. For each segment for which an acknowledgment is received, the size of CWND is increased by 1. Given such acknowledgments should take not much longer than a single rtt, slow start should cause the window to double each rtt. The window will continue increasing at this rate until either a packet is lost (the receiver fails to acknowledge a packet), CWND reaches RWND, or CWND reaches SST. Once any of these three conditions occur, the sender moves to congestion avoidance mode.

Note

How does increasing CWND by 1 for each ACL received double the window each rtt? The thinking is this: When the window size is 1, you should receive one segment per rtt. When you increase the window size to 2, you should receive 2 segments in each rtt; to 4, you should receive 4, etc. As the receiver is acknowledging each segment separately, and increasing the window by 1 each time it acknowledges a segment, it should acknowledge 1 segment in the first rtt, and set the window to 2; 2 segments in the second rtt, adding 2 to the window, to set the window to 4; 4 segments in the third rtt, adding 4 to the window, to set the window size to 8, etc.

16. Blanton, Paxson, and Allman, *TCP Congestion Control*.

In congestion avoidance mode, CWND is increased once each rtt, which means the size of the window stops growing exponentially and instead grows linearly. CWND will continue growing either until the receiver fails to acknowledge a packet (TCP assumes this means a packet has been lost or dropped), or until CWND reaches RWND. There are two broadly deployed ways in which a TCP implementation can respond to the loss of a packet, called *Tahoe* and *Reno*.

Note

There are actually many different variations of Tahoe and Reno; only the very basic implementations are considered here. There are also many different methods for reacting to a packet loss while the connection is in congestion avoidance mode; the "Further Reading" section contains information on where to find out about some of these other methods.

If the implementation is using Tahoe, and the packet loss is discovered through a fast retransmit, it will set SST to half of the current CWND, set CWND to its original value, and begin slow start again. This means the sender will transmit 1, 2, or 10 sequence numbers again, increasing CWND for each sequence number acknowledged. As in the beginning of the slow start process, this has the effect of doubling CWND each rtt. Once CWND reaches SST, TCP will move back into congestion avoidance mode.

If the implementation is using Reno, and the packet loss is discovered through a fast retransmit, it will set SST *and* CWND to half the current CWND, and continue operating in congestion avoidance mode.

In either implementation, if packet loss is discovered because the receiver does not send an acknowledgment within the RTO, the CWND is set to 1, and slow start is used to ramp the connection speed back up.

Error Control

TCP provides two forms of error detection and control:

- The protocol itself, along with the windowing mechanism, ensures data is delivered to the application in order and without any missing information.
- The one's complement checksum included in the TCP header is considered weaker than a Cyclic Redundancy Check (CRC) and many other forms of error detection. This error check serves to complement, rather than replace, the error correction provided by protocols lower and higher in the stack.

If a receiver detects a checksum error, it can use any of the mechanisms described here to request the sender retransmit the data—simply not acknowledging the receipt

of the data, requesting a retransmit through SACK, actively *not* acknowledging the receipt of the data through fast retransmit, or by sending a triple acknowledgment for the specific segment containing the corrupted data.

TCP Port Numbers

TCP does not directly manage any kind of multiplexing; however, it does provide port numbers that applications and protocols above TCP in the protocol stack can use to multiplex. While these port numbers are carried *in* TCP, they are generally *opaque to* TCP; TCP does not attach any meaning to these port numbers other than using them to dispatch information to the correct application on the receiving host.

TCP port numbers are divided into two broad classes: well known and ephemeral. Well-known ports are defined as a part of an upper layer protocol specification; these ports are the "default" ports for these applications. For instance, a service supporting the Simple Mail Transfer Protocol (SMTP) can generally be found by connecting to a host using TCP on port number 25. A service supporting the Hypertext Transport Protocol (HTTP) can normally be found by connecting to a host using TCP on port 80. These services do not *necessarily* need to use these port numbers; most servers can be configured to use some port number other than the one designated in the protocol specification. For instance, web servers not intended for general (or public) use may use some other TCP port, such as 8080.

Ephemeral ports are significant only to the local host and normally assigned from a pool of available port numbers on the local host. Ephemeral ports are most often used as source ports for TCP connections; for instance, a host connecting to a service at port 80 on a server will use an ephemeral port as its source TCP port. So long as any particular host uses a given ephemeral port number only once for any TCP connection, each TCP session on any network can be uniquely identified through the source address, source port, destination address, destination port, and the number of the protocol running on top of TCP.

TCP Session Setup

TCP uses a three-way handshake to set up a session:

1. The client sends a synchronization (SYN) to the server. This packet is a normal TCP packet, but with a SYN bit set in the TCP header, and indicates the sender is requesting a session to be set up with the receiver. This packet is normally sent to a well-known port number, or some prearranged port number that the client knows a server will be listening on at a particular IP address. This packet includes the client's initial sequence number.

2. The server sends an acknowledgment for the SYN, a SYN-ACK. This packet acknowledges the sequence number provided by the client, plus one, and includes the server's initial sequence number as the sequence number for this packet.
3. The client sends an acknowledgment (ACK) including the server's initial sequence number plus one.

This process is used to ensure two-way communication exists between the client and the server before beginning to transfer data. The initial sequence number chosen by the sender and receiver is randomized in most implementations to prevent a third-party attacker from guessing what sequence number will be used and taking over the TCP session in its initial stages of formation.[17]

QUIC

In 2012, Jim Roskind designed a new transport protocol with the primary intent of increasing the speed at which data can be transferred over relatively stable high-speed networks. Specifically:

- Reducing the three-way handshake to a single packet startup (a zero-way handshake)
- Reducing the number of retransmitted packets required to transfer data
- Reducing head-of-line blocking across multiple data streams within a single TCP stream caused by packet loss

Each of these is considered in the sections that follow.

Reducing the Startup Handshake

The rtt cannot, generally, be changed, because it is normally bounded by the physical distance and link speed between the sender and receiver. One of the best ways to reduce total data transfer time, then, is to simply reduce the number of round trips required between the sender and receiver to transfer a given stream or block of data. QUIC's startup is designed to reduce the number of round trips required to set up a

17. Gont and Bellovin, *Defending against Sequence Number Attacks*.

new connection from the three-way handshake of TCP to a 0 round trip time startup process.

To do this, QUIC uses a series of cryptographic keys and hashes (see Chapter 10, "Transport Security," for more information); the process is

1. The client sends the server a hello (CHLO) containing a proof demand, which is a list of certificate types the client will accept to verify the server's identity; a set of certificates the client has access to; and a hash of the certificate the client intends to use in this connection. One specific field, the source address token (STK) will be left blank, because no communication has occurred with this server before.
2. The server will use this information to create an STK based on the information provided in the client's initial hello and the client's source IP address. The server sends a reject (REJ), which contains this STK.

Once the client has the STK, it includes this in future hello packets. If the STK matches the previously used STK from this IP address, the server will accept the hello.

Note

This IP address/STK pair can be stolen, and hence the source IP address can be spoofed by an attacker with access to any communication with this pair included. This is a known problem in QUIC, addressed in the QUIC documentation pointed to in the "Further Reading" section at the end of the chapter.

In comparison, TCP requires at least one-and-a-half rtts to set up a new session: the SYN, the SYN-ACK, and then the following ACK. How much time does moving to a single rtt connection time save? It depends on the implementation of the client and server applications, of course. However, many web pages and mobile device apps must connect to many different servers (perhaps hundreds) to build a single web page or application screen. If each of these connections is reduced from one-and-a-half rtts to a single rtt, there could be a significant performance impact.

Reducing Retransmissions

QUIC uses a number of different mechanisms to reduce the number of retransmitted packets:

- Including Forward Error Correction (FEC) in all packets; this allows the receiver to (often) rebuild corrupted information rather than request the information to be resent.
- Using negative acknowledgments (NACKs) rather than SACK or the triple ACK mechanism to request retransmission of specific sequence numbers; this prevents ambiguity between a request for a retransmission and network conditions that cause multiple acknowledgments to be sent.
- Using fast acknowledgments, as described previously for TCP.
- Using the CUBIC congestion avoidance window control.

The CUBIC congestion avoidance mechanism is the most interesting of these. CUBIC attempts to perform a binary search between the last window size before a packet drop and some lower window size calculated using a **m**ultiplicative factor. When a packet loss is detected (either through an RTO timeout or through a NACK), the maximum window size (WMAX) is set to the current window size, and a new minimum window size (WMIN) is calculated.

The sender's window is set to WMIN and then quickly increased to a window size halfway between WMIN and WMAX. Once the window reaches this halfway point, the window size is increased very slowly in what is called probing, until the next packet drop is encountered. This process allows CUBIC to find the maximum transmission rate just below the point where the network begins dropping packets fairly quickly.

Reducing Head of Line Blocking

A "single transaction" across the Internet is often not a "single transaction," but rather a large collection of transactions across a number of different servers. To build a single web page, for instance, hundreds of elements, such as images, scripts, Cascading Style Sheet (CSS) elements, and Hypertext Markup Language (HTML) files need to be transferred from the server to the client. There are two ways these files can be transferred: in serial or in parallel. Figure 5-11 illustrates.

In Figure 5-11, three options are illustrated to transfer multiple elements from a server to a client:

- In the **serialized** option, the elements are transferred one at a time across a single session. This is the slowest of the three possible options, as the entire page must be built element by element, with smaller elements waiting on larger ones to transfer before they can be displayed.

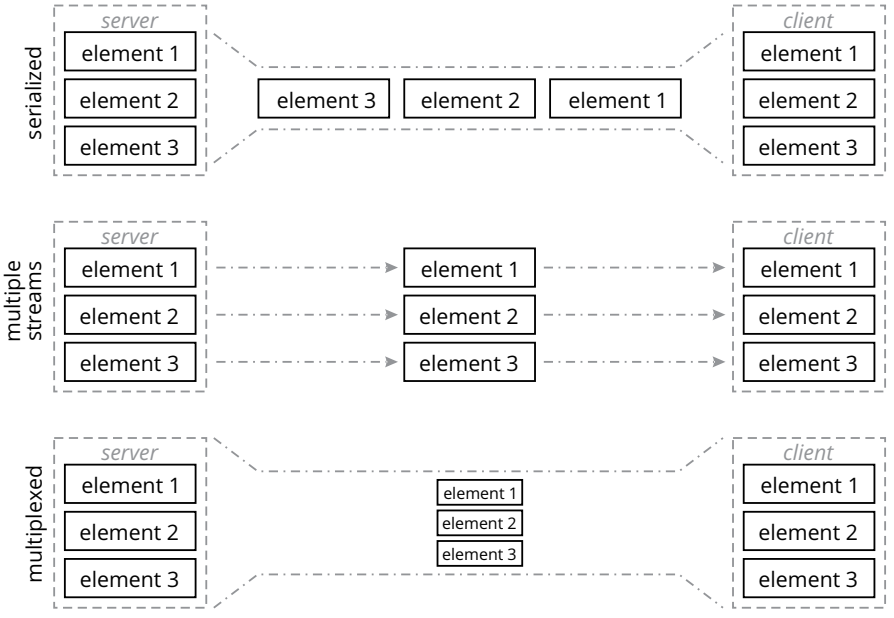


Figure 5–11 *Multiple Element Transfer Options*

- In the **multiple streams** option, each element is transferred over a separate connection (such as a TCP session). This is much faster, but it requires multiple connections to be built, which can negatively impact the client and server resources.
- In the **multiplexed** option, each element is transferred separately across a single connection. This allows each element to be transferred at its own rate, but with the resource overhead of the multiple streams option.

Some form of multiplexed transfer mechanism tends to provide the fastest transfer rate with the most efficient use of resources, but how should this multiplexing be implemented? The Hypertext Transfer Protocol version 2 (HTTPv2) allows a web server to multiplex multiple elements across a single HTTP session; since HTTP runs on top of TCP, this means a single TCP stream can be used to transfer multiple web page elements in parallel. However, a single dropped packet at the TCP level means every parallel transfer within the HTTP stream must be paused while TCP recovers (this is a form of fate sharing).

QUIC solves this problem by allowing multiple HTTPv2 streams to reside within a single QUIC connection. This reduces the transport overhead at the client and server, while providing optimal delivery of the web page elements.

Path MTU Discovery

One of the core issues of the argument between Asynchronous Transfer Mode (ATM) and the Internet Protocol (IP) was the fixed cell size. While IP networks rely on variable length packets, ATM, in order to facilitate faster switching speeds, and in order to interoperate better with the many different Time Division Multiplexing (TDM) physical layers, specified fixed length cells. IPv4, in particular, not only provides for a variable length packet, but fragmentation in flight. Figure 5-12 illustrates.

In Figure 5-12, if A sends a packet toward E, what size should it make the packet? The only link A really knows about is the link between itself and B, which is marked having a 1,500 octet Maximum Transmission Unit (MTU) size. If A sends a 1,500 octet packet, however, the packet will not be able to pass through the [C,D] link. There are two ways to solve this problem.

The first is for C to fragment the packet into two smaller packets. This is possible in IPv4; C can determine the packet will not fit on the next link over which the packet should be forwarded, and break the packet into two smaller packets. There are a number of problems with this solution, of course. For instance, the process of fragmenting a packet requires a lot more work on the part of C, possibly even moving the packet out of the hardware switching path into the software switching path.

The second is for A to never send a packet larger than the minimum MTU along the entire path to E. To do this, A must discover the minimum MTU along the path, and it must be able to fragment the information sent from upper layer protocols into multiple packets before transmission. IPv6 chooses this latter option, relying on Path MTU (PMTU) discovery to find the minimum MTU along a path (assuming PMTU actually works, a fairly bad assumption in large public networks), and allowing the IPv6 process at A to fragment information from upper layer protocols into multiple packets, which are then reassembled into the original upper layer data block at the receiver.

This solution, however, also seems to be problematic. In recent work with the Domain Name System (DNS), researchers have discovered that some 37%

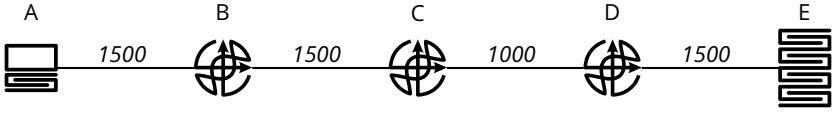


Figure 5–12 *Packet Fragmentation Example*

of all DNS resolvers will drop fragmented IPv6 packets.[18] Why would this be so? The easiest way to understand is to consider the structure of a fragmented packet, and the nature of DoS and DDoS attacks.

When a packet is transmitted, a header is placed on the packet indicating the receiving service (a socket or protocol number of some kind), as well as information about the transmitting service. This information is important to filtering a packet based on various security policies, particularly if the security policy is "only allow session initiation packets into the network unless the packet belongs to an existing session." In other words, a typical stateful filter protecting a server will have some basic rules it follows:

- If the packet initiates a new session, forward it and build a new session record.
- If the packet is part of an existing session, forward it and reset the session timer.
- If the packet is not part of an existing session, drop it.
- Every now and again, clean out old sessions.

While it is possible to forge a packet that appears to be from an existing session, it is not very easy; various nonces and other techniques are deployed to discourage this sort of behavior. But fragmenting a packet removes the header from the second half of the packet, effectively meaning the second packet in a fragmented pair can only be attached to a particular session, or flow, by tracing down the part of the packet that has the full header.

How can a router or middlebox do such a thing? It must somehow keep a copy of each packet fragment with a header someplace in memory, so the packet with the header can be referenced to process any future fragments. How long must it keep these fragments with headers? There is actually no way to tell. Ultimately, it is easier to simply drop any fragments than to maintain the state required to process them.

The result? It appears even source-based fragmentation is not very useful at the IP layer.

This should bring to mind one of the founding principles of the Internet Protocol suite: the *end-to-end principle*. The end-to-end principle states

18. Huston, "Dealing with IPv6 Fragmentation in the DNS."

the network should not modify traffic in flight between two end devices; or rather the network should operate as a black box connecting two devices, never changing the data as it is received from the end host.

Does this mean all filtering of traffic should be banned on the public Internet, imposing the end-to-end rule in earnest, leaving all security to the end hosts? This does seem to be the flavor of the original IPv6 discussions around stateful packet filters. This does not, however, seem like the most realistic option available; the stronger defense is not a single perfect wall, but rather a series of less than perfect walls. Defense in depth will beat a single firewall every time.

Another alternative is to accept another bit of reality often forgotten in the network engineering world: abstractions leak. The end-to-end principle describes a perfectly abstracted system capable of carrying traffic from one host to another, and a perfectly abstracted set of hosts between which traffic is being carried. But all nontrivial abstractions leak; the MTU and fragmentation problem is just a leakage of state from the network into the host, and a system on the host trying to abstract that leakage into the application sending traffic over the network. In this kind of situation, it might be best to simply admit the leakage and officially push the information up the stack so the application can make a better decision about how to send traffic.

But this leads to another interesting question to ponder: is the stateful filtering described here betraying the end-to-end principle? The answer depends on whether you consider the upper layer protocol shipping the data to be the end point, or the system the application is running on (hence, including the IP stack itself), the end point. Either way, this bit of ambiguity has plagued the Internet from the earliest of days, although the network engineering world has not always thought seriously about the difference between the two points of view.

ICMP

While the transport protocols, such as TCP and QUIC, tend to receive the most attention among the middle tier of protocols, there are a number of other protocols that are just as important for the operation of an IP-based network. Among these is ICMP, which can be said to provide metadata about the network itself. ICMP is a simple protocol that is used to request specific state information, or for network

devices to send information about why a particular packet is being dropped at some point in the network. Specifically:

- ICMP can be used to send an echo request or echo reply. This functionality is used to ping a particular destination address, which can be used to determine if the address is reachable without consuming too many resources at the receiver.
- ICMP can be used to send a notification about a packet being dropped because it is too large to be transmitted across a link (the packet is too big).
- ICMP can be used to send a notification that a packet has been dropped because its Time to Live (TTL) has reached 0 (the packet has expired in transit).

The packet too big response can be used to find the Maximum Transmission Unit (MTU) across a network; the sender can transmit a large packet and wait to see if some device in the network sends a *packet too big* notification through ICMP. If such a notification arrives, the sender can try progressively smaller packets to determine the largest packet that can be transmitted end-to-end across the network.

The expired in transit response can be used to trace the route from a source to a destination in a network (this is called trace route). A sender can transmit a packet to a particular destination using any transport layer protocol (including TCP, UDP, or QUIC), but with a TTL of 1. The first hop network device should decrement the TTL and send an ICMP expired in transit notification back to the sender. Sending a series of packets, each with a TTL one larger than the previous one, each device along the path can be induced to transmit an ICMP expired in transit notification to the sender, revealing the entire path of the packet.

Final Thoughts

Upper layer transport protocols manage the same problems as lower layer transport protocols—error control, flow control, transport, and marshaling—only end to end rather than device to device. Even so, while many of the solutions are similar or the same, many other solutions are radically different. This chapter has considered four different upper layer transport protocols, two of which occupy the same "space" in a protocol stack—TCP and QUIC—and two of which occupy completely different spaces in a protocol stack—IP and ICMP. While there are other solutions to the problems presented, the solutions presented by these four protocols cover most of the widely deployed solutions to these problems.

The next chapter moves you from understanding how information is transported across a network into the realm of how the layers considered in this and the previous chapter interact. These interlayer problems relate to the interaction surfaces considered in the State/Optimization/Surface model of network complexity you will find useful in analyzing a large number of network problems.

Further Reading

Armitage, Grenville. "Summary of Five New TCP Congestion Control Algorithms Project." *The FreeBSD Forums*. Accessed July 5, 2017. https://forums.FreeBSD.org/threads/22396/.

Blanton, Ethan, Dr. Vern Paxson, and Mark Allman. *TCP Congestion Control*. Request for Comments 5681. RFC Editor, 2009. doi:10.17487/RFC5681.

Chu, H. K. Jerry, and Vivek Kashyap. *Transmission of IP over InfiniBand (IPoIB)*. Request for Comments 4391. RFC Editor, 2006. doi:10.17487/RFC4391.

Cole, Robert G., Dr. David H. Shur, and Curtis Villamizar. *IP over ATM: A Framework Document*. Request for Comments 1932. RFC Editor, 1996. doi:10.17487/RFC1932.

Deering, Dr. Steve E., and Robert M. Hinden. "Internet Protocol, Version 6 (IPv6) Specification." Internet-Draft. Internet Engineering Task Force, May 2017. https://datatracker.ietf.org/doc/html/draft-ietf-6man-rfc2460bis-13.

Desanti, Claudio, Robert Nixon, and Craig Carlson. *Transmission of IPv6, IPv4, and Address Resolution Protocol (ARP) Packets over Fibre Channel*. Request for Comments 4338. RFC Editor, 2006. doi:10.17487/RFC4338.

"DoD Standard Internet Protocol." IETF, January 1980. https://tools.ietf.org/html/rfc760.

Fairhurst, Gorry, Marie-Jose Montpetit, Bernhard Collini-Nocker, Hilmar Linder, and Horst D. Clausen. *A Framework for Transmission of IP Datagrams over MPEG-2 Networks*. Request for Comments 4259. RFC Editor, 2005. doi:10.17487/RFC4259.

Floyd, Sally, Jamshid Mahdavi, Matt Mathis, and Dr. Allyn Romanow. *TCP Selective Acknowledgment Options*. Request for Comments 2018. RFC Editor, 1996. doi:10.17487/RFC2018.

Gont, Fernando, and Steven Bellovin. *Defending against Sequence Number Attacks*. Request for Comments 6528. RFC Editor, 2012. doi:10.17487/RFC6528.

Gupta, Mukesh, and Alex Conta. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. Request for Comments 4443. RFC Editor, 2006. doi:10.17487/RFC4443.

Ha, Sangtae, Injong Rhee, and Lisong Xu. "CUBIC: A New TCP-Friendly High-Speed TCP Variant." *ACM SIGOPS Operating System Review* 42, no. 5 (July 2008): 64–74.

Huston, Geoff. "Dealing with IPv6 Fragmentation in the DNS." *APNIC Blog*, August 22, 2017. https://blog.apnic.net/2017/08/22/dealing-ipv6-fragmentation-dns/.

Internet Control Message Protocol. Request for Comments 792. RFC Editor, 1981. doi:10.17487/RFC0792.

"Internet Protocol." IETF, September 1981. https://tools.ietf.org/html/rfc791.

IPv4 Exhaustion, June 9, 2010. https://commons.wikimedia.org/wiki/File:Ipv4-exhaust.svg.

Jacobson, V. "Congestion Avoidance and Control." In *Symposium Proceedings on Communications Architectures and Protocols*, 314–29. SIGCOMM '88. New York, NY, USA: ACM, 1988. doi:10.1145/52324.52356.

Jamal, Habibullah, and Kiran Sultan. "Performance Analysis of TCP Congestion Control Algorithms." *International Journal of Computers and Communications* 2, no. 1 (2008): 30–38.

Johansson, Peter G. *IPv4 over IEEE 1394*. Request for Comments 2734. RFC Editor, 1999. doi:10.17487/RFC2734.

Katz, Dave. *Transmission of IP and ARP over FDDI Networks*. Request for Comments 1390. RFC Editor, 1993. doi:10.17487/RFC1390.

Lawrence, Joe L., and David M. Piscitello. *The Transmission of IP Datagrams over the SMDS Service*. Request for Comments 1209. RFC Editor, 1991. doi:10.17487/RFC1209.

Luciani, Dr. James V., Dr. Bala Rajagopalan, and Daniel O. Awduche. *IP over Optical Networks: A Framework*. Request for Comments 3717. RFC Editor, 2004. doi:10.17487/RFC3717.

Mathis, Matt, Nandita Dukkipati, and Yuchung Cheng. *Proportional Rate Reduction for TCP*. Request for Comments 6937. RFC Editor, 2013. doi:10.17487/RFC6937.

Panabaker, Ruston, Simon Wegerif, and Dan Zigmond. *The Transmission of IP Over the Vertical Blanking Interval of a Television Signal*. Request for Comments 2728. RFC Editor, 1999. doi:10.17487/RFC2728.

Partridge, Dr. Craig, Mark Allman, and Sally Floyd. *Increasing TCP's Initial Window*. Request for Comments 3390. RFC Editor, 2002. doi:10.17487/RFC3390.

Postel, J. "Comments on Internet Protocol and TCP," August 15, 1977. https://www.rfc-editor.org/ien/ien2.txt.

———. "Draft Internetwork Protocol Specification, Version 2," February 1978. https://www.rfc-editor.org/ien/ien28.pdf.

———. "Internetwork Protocol Specification, Version 4," June 1978. https://www.rfc-editor.org/ien/ien41.pdf.

———. "Internetwork Protocol Specification, Version 4," September 1978. https://www.rfc-editor.org/ien/ien41.pdf.

"QUIC, a Multiplexed Stream Transport over UDP—The Chromium Projects." Accessed July 5, 2017. https://www.chromium.org/quic.

Riegel, Max, Sangjin Jeong, and HongSeok Jeon. *Transmission of IP over Ethernet over IEEE 802.16 Networks*. Request for Comments 5692. RFC Editor, 2009. doi:10.17487/RFC5692.

Stevens, W. Richard. *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*. Request for Comments 2001. RFC Editor, 1997. doi:10.17487/RFC2001.

Varada, Srihari V. *IP Version 6 over PPP*. Request for Comments 5072. RFC Editor, 2007. doi:10.17487/RFC5072.

Waitzman, David. *Standard for the Transmission of IP Datagrams on Avian Carriers*. Request for Comments 1149. RFC Editor, 1990. doi:10.17487/RFC1149.

Review Questions

1. The choice of using a /64 for the host address space is often considered controversial. What do you think are the positive and negative aspects of this specific choice?
2. The Internet has been "running out of IPv4 address space" for many years. One of the reactions to this lack of address space has been the widespread deployment of Network Address Translators (NATs). The following questions relate to NATs.
   a. What is the difference between a NAT and a Port Address Translator (PAT)?
   b. Why do PATs create a problem for FTP (for instance)? How is this normally solved?

   c. Throughout the development of IPv6, there was a general movement against the deployment of NATs and PATs; can you think of or find the reasons engineers objected to the use of NAT and PAT on the global Internet?

3. The fragmentation of packets by routers and other network devices was removed from the IPv6 specification, although it was allowed in the IPv4 specifications. What are the tradeoffs in removing this capability? What complexity does fragmentation add to network devices, and what complexity does removing it from the network devices add to end hosts?

4. How can an implementation of TCP differentiate between well-known and ephemeral ports? Does it need to?

5. From a security perspective, what might be the advantages and disadvantages of allowing network devices and hosts to respond to ICMP, versus not allowing them to?

6. Explain the aggregation of IPv6 addresses in terms of nibbles (4 bits) rather than in bits, as is done in the chapter.

7. Why is the prefix length called the *prefix length*? What is the history behind this term?

8. Compare the prefix length to the older method for determining the point where the network address stops and the host bits, or the "bits the network device does not care about," begin, the subnet mask. Which do you think is easier to use?

This page intentionally left blank

Chapter 6

Interlayer Discovery

Learning Objectives

After reading this chapter, you should be able to:

- Understand the four ways in which a device can discover the mapping between identifiers used in different protocols at different layers
- Understand port numbers
- Understand the basic operation of the Domain Name System (DNS)
- Understand the basic operation of the Dynamic Host Configuration Protocol (DHCP)
- Understand the Address Resolution Protocol (ARP)
- Understand Neighbor Discovery and Stateless Address Autoconfiguration
- Understand the default gateway

In a layered and/or modularized system, there must be some way to relate *services* or *entities* in one layer to services and entities in another. Figure 6-1 illustrates the problem.

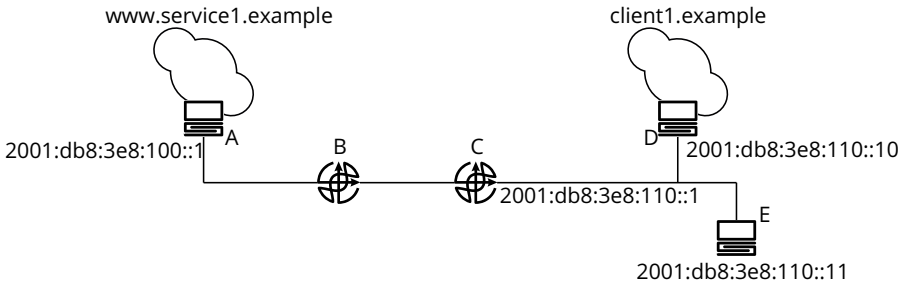


Figure 6-1 *Interlayer Discovery Problems*

In Figure 6-1:

- How can A, D, and E discover the IP address they should be using for their interfaces?
- How can D discover the Media Access Control (MAC), physical, or lower layer protocol address it should use to send packets to E?
- How can client1.example, which is running on D, discover the Internet Protocol (IP) address it should use to reach www.service1.example?
- How can D and E discover what address they should send traffic to if it is not on the same wire or segment?

Each of these problems represents a different part of interlayer discovery. While these problems may seem unrelated, they actually represent the same set of problems, with a narrow set of available solutions, at different layers of a network or protocol stack. This chapter will consider a range of possible solutions for these problems, including examples of each solution.

This chapter will end with a section on the default gateway problem; while this is not strictly an interlayer discovery problem, it is still important to understanding how an IP network operates.

Interlayer Discovery Solutions

The main reason the interlayer discovery problem space appears to be a large set of unrelated problems, rather than a single problem, is that it is spread across many different layers; each set of layers in a network protocol stack needs to be able to discover which service or entity at "this" layer relates to which service or entity at some lower layer. Another way to describe this set of problems is the ability to map an

identifier at one layer to an identifier at another layer—identifier mapping. As there are at least three pairs of protocols in most widely deployed protocol stacks (and potentially, or arguably, eight), a wide variety of solutions must be deployed to solve the same set of interlayer discovery problems in different places. Two definitions will be helpful in understanding the range of solutions, and actual deployed protocols and systems in this space:

- An **identifier** is a set of numbers or letters (such as a string) that uniquely identify an entity.
- A device, whether real or virtual, which appears to be a single destination from the point of view of the network will be called an **entity** when considering generic problems and solutions, and **hosts** or **services** when considering specific solutions.

There are four different ways to solve the interlayer discovery and address assignment problems:

- Using well-known and/or manually configured identifiers
- Storing the information in a *mapping database* that services can access to map between different kinds of identifiers
- Advertising a mapping between two identifiers in a protocol
- Calculating one kind of identifier from another

These solutions not only apply to *discovery,* but also identifier *assignment*. When a host is connected to a network, or a service is spun up, it must somehow determine how it should identify itself—for instance, what Internet Protocol version 6 (IPv6) address it should use when connecting to the local network. The solutions available for solving this problem are the *same four solutions*.

These four solutions will be considered in the following sections.

Well-Known and/or Manually Configured Identifiers

The solution chosen often depends on the scope of the identifiers, the sheer number of identifiers that need to be assigned, and the rate at which the identifiers change. If

- The identifiers are widely used, especially in protocol implementations, and the network will simply not work without some agreement on the interlayer mappings, and…

- The number of mappings between identifiers is relatively small, and…
- The identifiers are generally stable—in particular, they are never changed in a way that requires existing, deployed implementations to be modified in order to allow the network to continue functioning, then…

The easiest solution is to manually maintain a mapping table of some kind.

For instance, the Transmission Control Protocol (TCP) carries a number of higher layer transport protocols. The problem of relating individual carried protocols to port numbers is a global interlayer discovery problem: every implementation of TCP deployed in a real network must be able to agree on what services are reachable on specific port numbers for the network to "work." The range of interlayer mappings, however, is very small, a few thousand port numbers need to be mapped to services, and fairly static (new protocols or services are not often added). This specific problem, then, is easy to solve through a manually managed mapping table.

The mapping table for TCP port numbers is maintained by the Internet Assigned Numbers Authority (IANA), at the direction of the Internet Engineering Task Force (IETF); a part of this table is shown in Figure 6-2.[1]

In Figure 6-2, the *echo* service is assigned port 7; this service is used to provide the *ping* functionality described at the end of Chapter 5, "Higher Layer Data Transports."

Mapping Database and Protocol

If the number of entries in the table becomes large enough, the number of people involved in maintaining the table becomes large enough, or the information is dynamic enough that it needs to be learned at the time the mapping is required, rather than when a piece of software is deployed, it makes sense to build and distribute a database dynamically. Such a system should include protocols to synchronize database partitions to present a consistent view to external queries, and protocols hosts and services can use to query the database with one identifier to discover the matching identifier from a different layer of the network.

Dynamic mapping databases may accept input through manual configuration or automated processes (such as a discovery process that gathers information about the state of the network and stores the resulting information in the dynamic database). They may also either be distributed, which means copies or portions of the database

1. This chart is taken from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

Service Name	Port Number	Transport Protocol
	0	tcp
	0	udp
tcpmux	1	tcp
tcpmux	1	udp
compressnet	2	tcp
compressnet	2	udp
compressnet	3	tcp
compressnet	3	udp
	4	tcp
	4	udp
rje	5	tcp
rje	5	udp
	6	tcp
	6	udp
echo	7	tcp
echo	7	udp
	8	tcp
	8	udp
discard	9	tcp
discard	9	udp
discard	9	sctp
discard	9	dccp
	10	tcp
	10	udp

Figure 6-2 *IANA TCP Port Mapping Table*

are stored on a number of different hosts or servers, or centralized, which means the database is stored on a small number of hosts or servers.

The Domain Name System (DNS) is described as an example of an identity mapping service based on a dynamic, distributed database. The Dynamic Host Configuration Protocol (DHCP) is described as an example of a similar system used primarily for the assignment of addresses.

Advertising Identifier Mappings in a Protocol

If the scope of the mapping problem can be contained, but the number of identity pairs is large, or can change rapidly, then creating a single protocol that allows entities to request mapping information from a device directly can be an optimal solution. For instance, in Figure 6-1, D could ask E directly what its local MAC (or physical) address is.

The Internet Protocol version 4 (IPv4) Address Resolution Protocol (ARP) is a good example of this kind of solution, as is the IPv6 Neighbor Discovery (ND) protocol. These examples are considered in more detail in later sections.

Calculating One Identifier from the Other

In some cases, it is possible to calculate an address or identifier at one layer from the address or identifier in another layer. Few systems use this technique for mapping addresses; most systems that use this technique do so in order to *assign* an address. One example of this type of system is Stateless Address Autoconfiguration (SLAAC), an IPv6 protocol hosts can use to determine what IPv6 address should be assigned to an interface, which is considered in more detail as part of the IPv6 ND discussion later in the chapter.

Another example of using a lower layer address to calculate an upper layer address is in the formation of end-system addresses in the International Organization for Standardization (ISO) suite of protocols, such as Intermediate System to Intermediate System (IS-IS). This example is considered in more detail in Chapter 16, "Link State and Path Vector Control Planes."

Interlayer Discovery Examples

Four examples of protocols providing interlayer discovery and address assignment are considered in the following sections.

The Domain Name System

DNS maps between human-readable character strings, such as the name service1.example used in Figure 6-1, to IP addresses. Figure 6-3 illustrates the basic operation of the DNS system.

In Figure 6-3, assuming there are no caches of any kind (so the entire process is illustrated):

1. A host, A, attempts to connect to www.service1.example. The host's operating system examines its local configuration for the address of the DNS server it should query to discover where this service is located, and finds the address of the recursive server. The host operating system's DNS application sends a DNS query to this address.

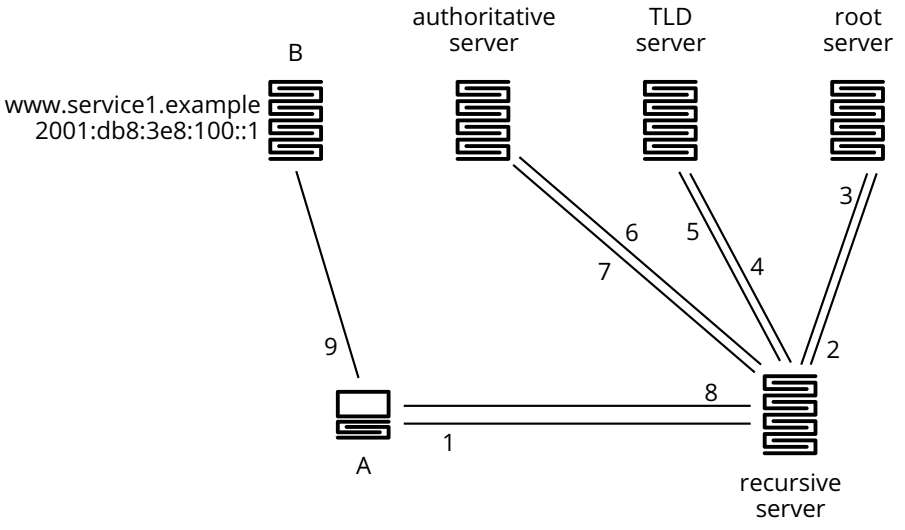


Figure 6-3 *An Overview of the Operation of the DNS System*

2. The recursive server receives this query and—given there are no caches—examines the domain name for which an address is being requested. The recursive server notes the right-hand portion of the domain name is *example*, so it asks a root server where to find information on the *example* domain.

3. The root server returns the address of the server containing information about the top-level domain (TLD) *example*.

4. The recursive server now requests information about which server to contact about *service1.example*. The recursive server proceeds through the domain name one section at a time, using information discovered about the section of the name to the right to discover which server to ask about the information to the left. This process is called *recursing* through the domain name; hence the server is called a recursive server.

5. The TLD server returns the address of the *authoritative server* for service1.example. If information about the location of a service has been cached from a prior request, it is returned as a nonauthoritative answer; if the actual server configured to hold the information about a domain replies, its answer is *authoritative*.

6. The recursive server requests information about www.service1.example from the authoritative server.
7. The authoritative server responds with the IP address of server B.
8. The recursive server now responds to the host, A, with the correct information to reach the requested service.
9. The host, A, contacts the server on which www.service1.example is running on the IP address 2001:db8:3e8:100::1.

This process may appear to be very drawn out; for instance, why not just keep all the information on the root server to save a lot of steps? This would violate the basic idea of DNS, however, which is to keep information about each domain in the control of the domain owner as much as possible. Further, this would make the building and maintenance of the root servers very expensive, as they would need to be capable of holding millions of records and answer hundreds of millions of queries for DNS information each day. The separation of information allows each owner to control his data and enables the DNS system to scale.

Normally, the information returned through a DNS query process is cached by each server along the way, so the mapping does not need to be requested each time the host needs to reach a new server.

How are these DNS tables maintained? Usually through the manual work of domain- and top-level domain owners, as well as edge providers all across the world. DNS does not automatically discover the name of each entity attached to the network and what each one's address is.

DNS pairs a manually maintained database, with the work spread out among many different pairs of hands, with a protocol used to query the database; hence DNS falls into the *mapping database with a protocol* class of solutions. How does a host know what DNS server to query? This information is either manually configured or learned through a discovery protocol such as IPv6 ND or DHCP.

DHCP

When a host (or some other device) first connects to a network, how does it know which IPv6 address (or set of IPv6 addresses) to assign to the local interface? One solution to this problem is for the host to send a query to some database to discover what addresses it should use, such as DHCPv6. To understand DHCPv6, it is important to begin with the concept of a link local address in IPv6. In the discussion on the size of the IPv6 address space in Chapter 5, "Higher Layer Data Transports," fe80::/10 was called out as being reserved for link local addressing. To form a link

local address, a device running IPv6 combines the fe80:: prefix with the MAC (or physical) address, which is often formatted as an EUI-48 address, and sometimes as an EUI-64 address (see Chapter 4, "Lower Layer Transports," for information on EUI addresses). For instance:

- A device has an interface with the EUI-48 address 01-23-45-67-89-ab.
- This interface is connected to an IPv6 network.
- The device can assign fe80::123:4567:89ab as a link local address and use this address to communicate to other devices on this segment only.

This is an example of *calculating one identifier from another.* Once the link local address has been formed, DHCP6 is one method that can be used to obtain a unique address within the network (or globally, depending on the configuration of the network). DHCPv6 uses the User Datagram Protocol (UDP) for its lower layer transport. Figure 6-4 illustrates.

In Figure 6-4:

1. The host that has just connected to the network, A, sends a *solicit* message. This message is sourced from the link local address and sent to the multicast address ff02::1:2, UDP ports 547 (for the server) and 546 (for the client), so every device connected to the same physical wire will receive the message. This message will

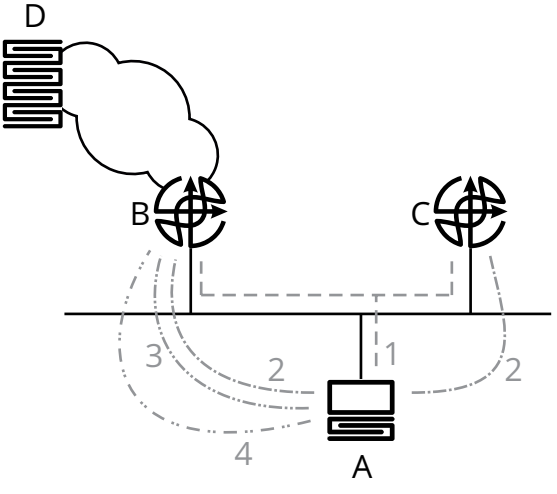


Figure 6-4 *DHCPv6 Operation*

include a DHCP Unique Identifier (DUID), which the client forms,[2] and the server uses to ensure it is consistently communicating with the same device.

2. B and C, both of which are configured to act as DHCPv6 servers, respond with an *advertise* message. This message is a unicast packet directed at A itself, using the link local address from which A sources the solicit message.
3. Host A chooses one of the two servers from which to request an address. The host sends a *request* to the multicast address ff02::1:2, asking B to provide it with an address (or a pool of addresses), information on which DNS server to use, etc.
4. The server, running on B, then responds with a *reply* to the link local address A initially formed; this verifies B has allocated the resources from its local pool and allows A to start using them.

What happens if no device on the segment is configured as a DHCPv6 server? For instance, in Figure 6-4, what if D is the only available DHCPv6 server because DHCPv6 is not running on B or C? In this case, a router (or even some other host or device) can act as a *DHCPv6 relay*. The DHCPv6 packets that A transmits will be received by the relay, encapsulated, and transmitted to the DHCPv6 server for processing.

Note

The process described here is called stateful DHCP and is normally triggered when the *Managed* bit is set in the *router advertisement*. DHCPv6 can also work with SLAAC, described later in the "IPv6 Neighbor Discovery" section, to provide information SLAAC does not provide in the stateless DHCPv6 mode. This mode is normally used when the *Other* bit is set in the router advertisement. The IETF draft *DHCPv6/SLAAC Interaction Problems on Address and DNS Configuration* describes this interaction and problems in the interaction between these two mechanisms.[3]

3. Liu et al., "DHCPv6/SLAAC Interaction Problems on Address and DNS Configuration."

In cases where the network administrator knows all IPv6 addresses will be configured through DHCPv6, and only one DHCPv6 server will be available on each

2. Johnson and Narten, *Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)*.

segment, the advertise and request messages can be skipped by enabling DHCPv6 *rapid commit*.

IPv4 Address Resolution Protocol

Although IPv6 is the focus of this book, there are some instances where IPv4 provides a useful example of a solution; the IPv4 Address Resolution Protocol (ARP) is one such case. ARP is a very simple protocol used to solve interlayer discovery without relying on a server of any type. Figure 6-5 will be used to explain the operation of ARP.

Assume A would like to send a packet to C. Knowing C's IPv4 address, 203.0.113.12, is not enough for A to properly form a packet to place on the wire toward C. To properly build a packet, A must also know

- Whether or not C is on the same wire as A
- The MAC, or physical, address of C

Without two pieces of information, A does not know how to encapsulate the packet on the wire, so C will actually receive the packet, and B will ignore it. How can A discover this information? The first question, whether or not C is on the same

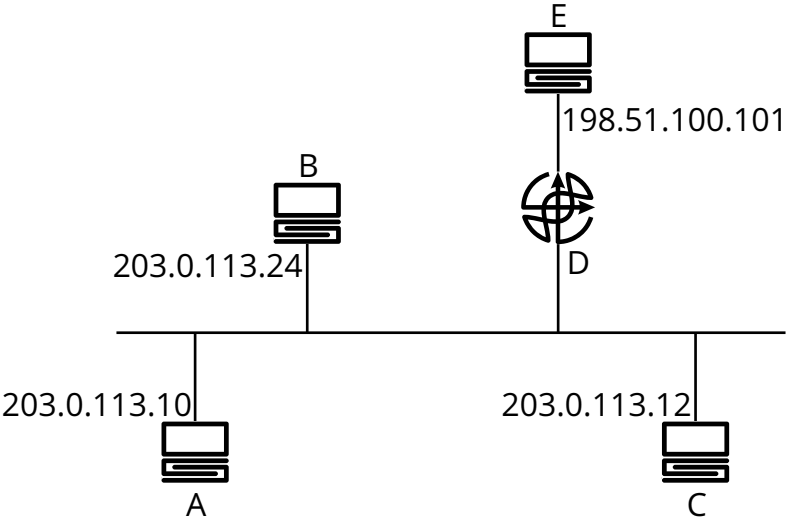


Figure 6-5 *Address Resolution Protocol Example*

wire as A, can be answered by considering the local interface IP address, the destination IP address, and the subnet mask. This is considered in more detail later in this chapter.

ARP solves the second problem, matching the destination IP address to the destination MAC address, with the following process:

1. Host A sends a broadcast packet to every device on the wire containing the IPv4 address, but not the MAC address. This is an *ARP request;* it is A's request for the MAC address corresponding to 203.0.113.12.
2. B and D receive this packet, but do not respond, because none of their local interfaces have the address 203.0.113.12.
3. Host C receives this packet and responds, again using a unicast packet, to the request. This *ARP reply* contains both the IPv4 address and the matching MAC address, giving A the information needed to build packets toward C.

When A receives this reply, it will insert the mapping between 203.0.113.12 and the MAC address contained in the reply in a local ARP cache. This information will be stored until it times out; the rules for timing out an ARP cache entry vary between implementations and can often be manually configured. How long to cache an ARP entry is a balance between not repeating the same information too often on the network, in the case where the IPv4-to-MAC address mapping does not change very often, and keeping up with any changes in the location of a device, in the case where a particular IPv4 address may move between hosts.

Any device receiving an ARP reply can accept the packet and cache the information it contains. For instance, B, on receiving the ARP reply from C, can insert the mapping between 203.0.113.12 and C's MAC address into its ARP cache. In fact, this property of ARP is often used to speed up the discovery of devices when they are attached to a network. There is nothing in the ARP specification that requires a host to wait for an ARP request to send an ARP reply. When a device connects to a network, it can simply send an ARP reply with the correct mapping information to make the initial connection process to other hosts on the same wire faster; this is called a *gratuitous ARP.*

Gratuitous ARPs are also useful for *Duplicate Address Detection (DAD);* if a host receives an ARP reply with an IPv4 address it is using, it will report a duplicate IPv4 address. Some implementations will also send out a series of gratuitous ARPs in this case, in order to prevent the address from being used, or force the other host to also report the duplicate address.

What happens if Host A requests an address using ARP that is not on the same segment, such as 198.51.100.101 in Figure 6-5? There are two different possibilities to this situation:

- If D is configured to answer as a proxy ARP, it can respond to the ARP request with the MAC address connected to the segment. A will then cache this response, sending any traffic destined to E to the MAC address of D, which can then forward this traffic on to E. Most widely deployed implementations do not enable proxy ARP by default.
- A could send the traffic to its default gateway, which is a locally connected router that should know the path to any destination on the network.

IPv4 ARP is an example of a protocol that maps interlayer identifiers by including both identifiers in a single protocol.

IPv6 Neighbor Discovery

IPv6 replaces the simpler ARP protocol with a series of Internet Control Message Protocol (ICMP) v6 messages. Five kinds of ICMPv6 messages are defined:

- Type 133, Router Solicitation
- Type 134, Router Advertisement
- Type 135, Neighbor Solicitation
- Type 136, Neighbor Advertisement
- Type 137, Redirect

Figure 6-6 is used to explain the operation of IPv6 ND.

To understand the operation of IPv6 ND, it is best to follow a single host as it is connected to a new network. Host A in Figure 6-6 is used as an example.

- A will begin by forming a link local address, as described previously; assume A chooses fe80::AAAA as its link local address.
- A now uses this link local address as a source address and sends a router solicitation to a link local multicast address (the all nodes multicast address); this is an ICMPv6 message type 133.

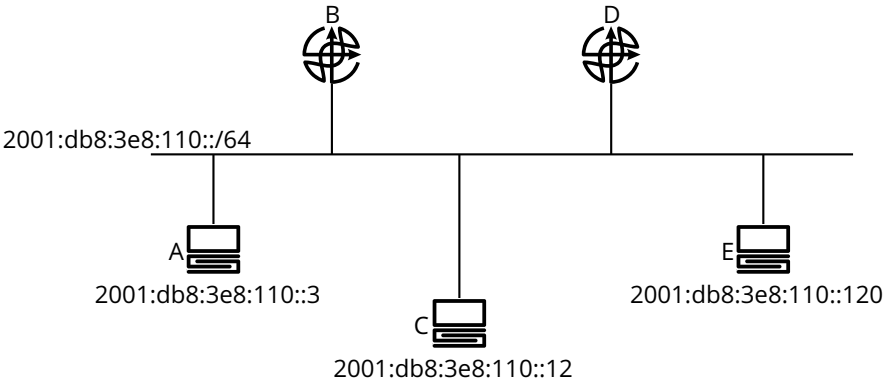


Figure 6-6 *IPv6 Neighbor and Router Discovery*

- B and D receive this router solicitation and respond with a router advertisement, which is an ICMPv6 message type 134. This unicast packet is transmitted to the link local address A used as the source address, fe80::AAAA.
  - The router advertisement contains information on how the newly connected host should determine its local configuration information in the form of several flags.
  - The *M* flag indicates the host should request an address through DHCPv6, because this is a *managed* link.
  - The *O* flag indicates the host can retrieve information *other* than the address it should use via DHCPv6. For instance, the DNS server the host should use to resolve DNS names should be retrieved using DHCPv6.
- If the *O* flag is set, and not the *M* flag, A must determine its own interface IPv6 address. To do this, it determines the set of IPv6 prefixes in use on this segment by examining the *prefix information* field in the router advertisement. It chooses one of these prefixes and forms an IPv6 address using the same process it used to form a link local address: it adds a local MAC (EUI-48 or EUI-64) address to the indicated prefix. This process is called SLAAC.
- The host must now make certain it has not chosen an address some other host on the same network is using; it must perform DAD. To perform a duplicate address detection:
  - The host sends a series of neighbor solicitation messages using the just-formed IPv6 address and asking for the corresponding MAC (physical) address. These are ICMPv6 type 135 messages transmitted from the link local address already assigned to the interface.

- If the host receives a neighbor advertisement or neighbor solicitation using the same IPv6 address, it assumes the locally formed address is a duplicate; in this case, it will form a new address using a different local MAC address and try again.
- If the host does not receive a response, nor another host's neighbor solicitation using the same address, it assumes the address is unique and assigns the newly formed address to the interface.

Resolving False Positives in Duplicate Address Detection

The DAD process, as described here, can result in false positives. Specifically, if some other device on the wire loops the original neighbor solicitation packets back to A, it will believe these are from another host claiming the same address, and hence will declare a duplicate and try to form a new address. If the device constantly loops back any neighbor solicitation A sends, A will never be able to form an address using SLAAC.

To solve this, RFC7527 outlines an enhanced DAD process. In this process, A would calculate a nonce, or rather a randomly selected series of numbers, and include it in the neighbor solicitation used to check for a duplicate address. This nonce is included through the Secure Neighbor Discovery (SEND) extensions to IPv6 outlined in RFC3971.

If A receives a neighbor solicitation with the same nonce it used to send neighbor solicitations during DAD, it will form a new nonce and try again. If it occurs a second time, the host will assume the packets are being looped and ignore any further neighbor solicitations with its own nonce in them. If the received neighbor solicitations have a different nonce than the one the local host has chosen, the host will assume there is, in fact, another host that has chosen the same IPv6 address and will then form a new IPv6 address.

Once it has an address to transmit data from, A now needs one more piece of information before sending information to another host on the same segment—the MAC address of the receiving host. If A, for instance, wants to send a packet to C, it will begin by sending a multicast neighbor solicitation message to C asking for its MAC address; this is an ICMPv6 message type 135. When C receives this message, it will respond with the correct MAC address to send traffic for the requested IPv6 address; this is an ICMPv6 message type 136.

While the preceding process describes router advertisements being sent in response to a router solicitation, each router will send periodic router advertisements on each attached interface. The router advertisement contains a *lifetime* field, indicating how long the router advertisement is valid.

The Default Gateway Problem

How can a host know whether to try to send a packet to a host over the segment it is connected to, or to send the packet to a router for further processing? If a host should send packets to a router for further processing, how can it know which router (if there is more than one) to send the traffic to? These two problems, together, make up the default gateway problem.

For IPv4, the problem is fairly easy to solve using the prefix and prefix length. Figure 6-7 illustrates.

IPv4 implementations assume *any host within the same IPv4 subnet must be physically connected to the same wire*. How can the implementation tell the difference? The subnet mask is another form of the prefix length, which indicates where

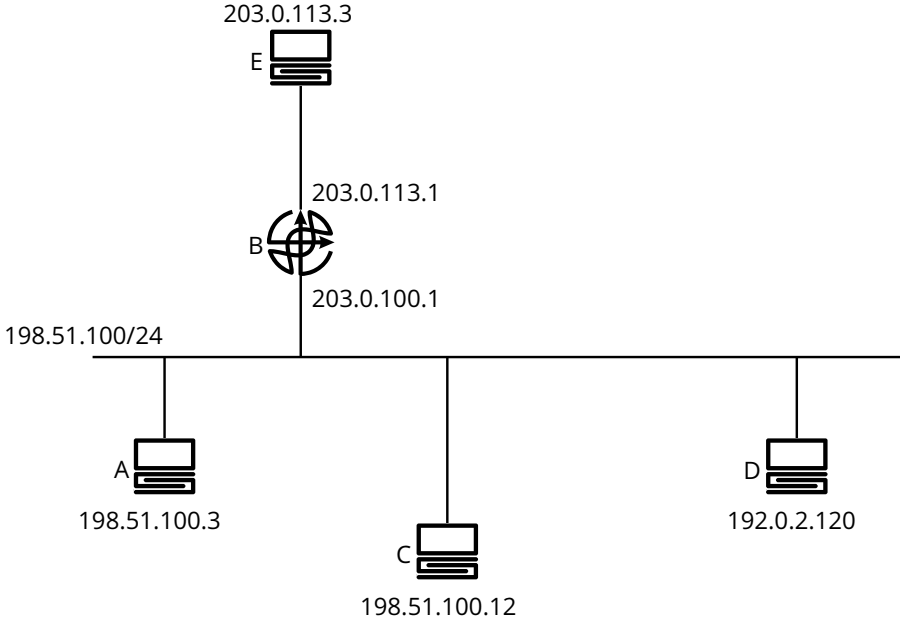


Figure 6-7 *IPv4 Default Gateway Usage*

the network address ends and the host address begins. In this case, assume the prefix length is 24 bits, or the network address is a /24. The *24* tells you how many bits are set in the subnet mask:

```
24 bits = 11111111.11111111.11111111.0000000
```

Since IPv4 uses a "dotted decimal" notation, this can also be written as 255.255.255.0. To discover whether or not C is on the same wire as A, A will

1. Logically AND the subnet mask with the local interface address
2. Logically AND the subnet mask with the destination address
3. Compare the two results; if they match, the destination host is on the same wire as the local interface

Figure 6-8 illustrates.

There are four IPv4 addresses in Figure 6-8; assume A needs to send packets to C, D, and E. If A knows the prefix length of the local segment is 24 bits either through manual configuration or through DHCPv4, then it can simply look at the 24 most significant bits of each address, compare it to the 24 most significant bits of its own address, and determine whether the destination is on segment or not. Twenty-four bits of an IPv4 address produces a nice break between the third and fourth section of the address (each section of an IPv4 address represents 8 bits of address space, for a total of 32 bits of address space).

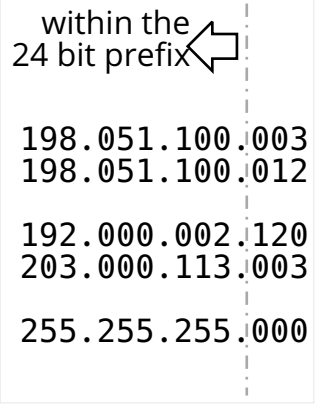


Figure 6-8 *Using the Prefix Length to Determine What Is On and Off Segment*

Any two addresses with the same left three sections as A has, called the network address, are on the same segment; any address that does not is not on segment. In this case, the network address for A and C match, so A will believe C is on the same segment, and hence will send packets to C directly, rather than sending them to a router. For any destination A believes is off segment, it will send packets to the final destination's IPv4 address, but to the default gateway's MAC address. This means the router acting as the default gateway will accept the packet and switch it based on the destination IPv4 address (packet switching is considered more fully in Chapter 7, "Packet Switching"). How is the default gateway chosen? It is either manually configured or included in a DHCPv4 option.

What about D? Because the network portions of the addresses don't match, A will believe D is off segment. In this case, A will send any traffic for D to its default gateway, which is B. When B receives these packets, it will realize A and D are reachable through the same interface (based on its routing table—building routing tables is considered in Part II, "The Control Plane"), so it will send an ICMP redirect to A telling it to send traffic toward D directly, rather than through B.

IPv6 presents a more complex set of problems to solve when considering which default gateway to use, because IPv6 assumes a single device may have many IPv6 addresses assigned to a particular interface. Figure 6-9 illustrates.

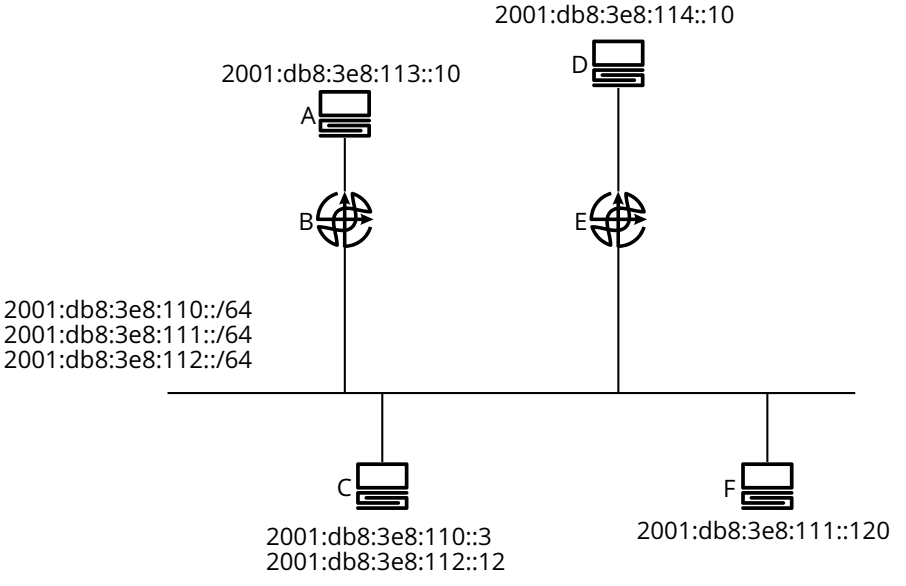


Figure 6-9 *IPv6 and On Link/Off Link Determination*

In Figure 6-9, assume the network administrator has configured the following policies:

- No host may connect to A unless it has an address in the 2001:db8:3e8:110::/64 range of addresses.
- No host may connect to D unless it has an address in the 2001:db8:3e8:112::/64 range of addresses.

Note

You would never build policies like this in the real world; this is a contrived situation to illustrate a problem set in a minimally sized network. A much more real problem of this same type would involve unicast Reverse Path Forwarding (uRPF).

To make these policies work, the administrator has assigned 110::3 and 112::12 to host C and 111::120 to host F. This might look odd, but it is perfectly legal for a single segment to have multiple IPv6 subnets assigned in IPv6; it is also perfectly legal to have a single device with multiple addresses. In fact, in IPv6, there are many situations where a single device may have a range of addresses assigned.

From the perspective of the prefix lengths, however, no two addresses assigned to C or F are on the same subnet. Because of this, IPv6 does not rely on the prefix length to determine what is on segment and what is not. Instead, IPv6 implementations keep a table of all connected hosts, using neighbor solicitations to discover what is on segment and what is not. When a host wants to send traffic off the local segment, it sends the traffic to one of the routers it has learned about through router advertisements. If a router receives a packet that it knows another router on the segment has a better route to (because the routers have routing tables that tell them which path to take to any particular destination), the router will send an ICMPv6 redirect message telling the host to use some other first hop router to reach the destination.

Final Thoughts

This chapter has provided an overview of a very difficult problem, and a number of complex solutions—the Domain Name System, the Dynamic Host Configuration Protocol, the Address Resolution Protocol, and Neighbor Discovery—are far more complex than the high-level overviews provided here. The deployment and operation

of DNS servers and the maintenance of the DNS system are an entire career field within network engineering, for instance.

Even so, all of these complex solutions represent just one of four ways to solve the difficult problems of mapping the identifiers used at one layer into the identifiers used at another layer, or the discovery of identifiers in order to facilitate communication. The contrast between the four basic solutions and the diverse protocols implementing those solutions is a solid example of the premise of this book: if you understand the problem space, and you understand the available solutions, then it becomes possible to ask the right questions of a solution to understand how it works.

Once the identifiers have been discovered, and the data to be transported has been marshaled, it is time to switch packets through the network; this is the topic of the next chapter.

Further Reading

Asati, Rajiv, Hemant Singh, Wes Beebee, Carlos Pignataro, Eli Dart, and Wesley George. *Enhanced Duplicate Address Detection*. Request for Comments 7527. RFC Editor, 2015. doi:10.17487/RFC7527.

Baker, Fred, and Brian E. Carpenter. *First-Hop Router Selection by Hosts in a Multi-Prefix Network*. Request for Comments 8028. RFC Editor, 2016. doi:10.17487/RFC8028.

Beebee, Wes, Hemant Singh, and Erik Nordmark. *IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes*. Request for Comments 5942. RFC Editor, 2010. doi:10.17487/RFC5942.

Droms, Ralph. *DNS Configuration Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*. Request for Comments 3646. RFC Editor, 2003. doi:10.17487/RFC3646.

———. *Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6*. Request for Comments 3736. RFC Editor, 2004. doi:10.17487/RFC3736.

Gont, Fernando. *Security Implications of IPv6 Fragmentation with IPv6 Neighbor Discovery*. Request for Comments 6980. RFC Editor, 2013. doi:10.17487/RFC6980.

Johnson, Jarrod, and Dr. Thomas Narten. *Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)*. Request for Comments 6355. RFC Editor, 2011. doi:10.17487/RFC6355.

Kempf, James, Jari Arkko, Brian Zill, and Pekka Nikander. *SEcure Neighbor Discovery (SEND)*. Request for Comments 3971. RFC Editor, 2005. doi:10.17487/RFC3971.

Liu, Bing, Sheng Jiang, Xiangyang Gong, Wendong Wang, and Enno Rey. "DHCPv6/SLAAC Interaction Problems on Address and DNS Configuration." Internet-Draft. Internet Engineering Task Force, August 2016. https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-dhcpv6-slaac-problem-07.

Mrugalski, Tomek, Marcin Siodelski, Bernie Volz, Andrew Yourtchenko, Michael Richardson, Sheng Jiang, Ted Lemon, and Timothy Winters. "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) bis." Internet-Draft. Internet Engineering Task Force, June 2017. https://datatracker.ietf.org/doc/html/draft-ietf-dhc-rfc3315bis-09.

Narten, Dr. Thomas, Tatsuya Jinmei, and Dr. Susan Thomson. *IPv6 Stateless Address Autoconfiguration*. Request for Comments 4862. RFC Editor, 2007. doi:10.17487/RFC4862.

Nordmark, Erik, and Igor Gashinsky. *Neighbor Unreachability Detection Is Too Impatient*. Request for Comments 7048. RFC Editor, 2014. doi:10.17487/RFC7048.

Simpson, William A., Dr. Thomas Narten, Erik Nordmark, and Hesham Soliman. *Neighbor Discovery for IP Version 6 (IPv6)*. Request for Comments 4861. RFC Editor, 2007. doi:10.17487/RFC4861.

Troan, Ole, and Ralph Droms. *IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) Version 6*. Request for Comments 3633. RFC Editor, 2003. doi:10.17487/RFC3633.

Zeng, Shengyou, John Jason Brzozowski, Kim Kinnear, and Bernie Volz. *DHCPv6 Leasequery*. Request for Comments 5007. RFC Editor, 2007. doi:10.17487/RFC5007.

Review Questions

1. Consider each of the four ways to solve the interlayer discovery and mapping problem discussed in the chapter. Build a chart describing the state and surface interactions for each one, and what the optimization tradeoffs might be.

2. Describe the process the IETF uses for maintaining number registries. Does this seem like a complex system or a simple one? Does it seem as though it would be effective in ensuring identifier uniqueness?

3. Consider that there must be millions, or perhaps hundreds of millions, of DNS queries each day. How many DNS root servers are there? Given these two numbers, how do you think the DNS system is scaled to support the entire global Internet?
4. Is it possible to convert a globally reachable IP address to a DNS name (to map in the opposite direction from what is described in the chapter)? Can you think of one example where this would be useful?
5. The "larger" DNS system also contains a mapping system from DNS names to human-readable information about domain ownership called whois. What protocol does it use to communicate, where is the information stored, and what kinds of information are available through this system?
6. Explain what DNS glue records are and what they are used for.
7. From the perspective of state, optimization, and surface, what are the tradeoffs between a mechanism like DHCP and one like SLAAC? Consider not only the ease with which addresses can be assigned, but any security and control issues that might arise with each one.
8. Why would most implementations not enable proxy ARP by default? What is the risk in enabling proxy ARP?
9. How does the neighbor discovery protocol End System to Intermediate System (ES-IS) compare to IPv6 ND?
10. Consider how IPv6 Router Discovery works in relation to the default gateway problem.

Chapter 7

Packet Switching

Learning Objectives

After reading this chapter, you should be able to understand:

- The four steps required to switch a packet through a network device
- How the receive and transmit rings are used in the process of forwarding a packet
- The basic process of switching a packet, including how the forwarding tables are built
- How routing is different from switching and the advantages of routing
- The concept of equal cost multipath
- The concept of link aggregation
- The concept of a bus and the concept of a crossbar fabric

Network devices are inserted into networks to solve a number of problems, including connecting different kinds of media and scaling a network by only carrying packets where they need to go. Routers and switches are, however, complex devices in their own right; engineers can build an entire career by specializing in solving just a small set of the problems encountered in carrying packets through a network device. Figure 7-1 is used to discuss an overview of the problem space.

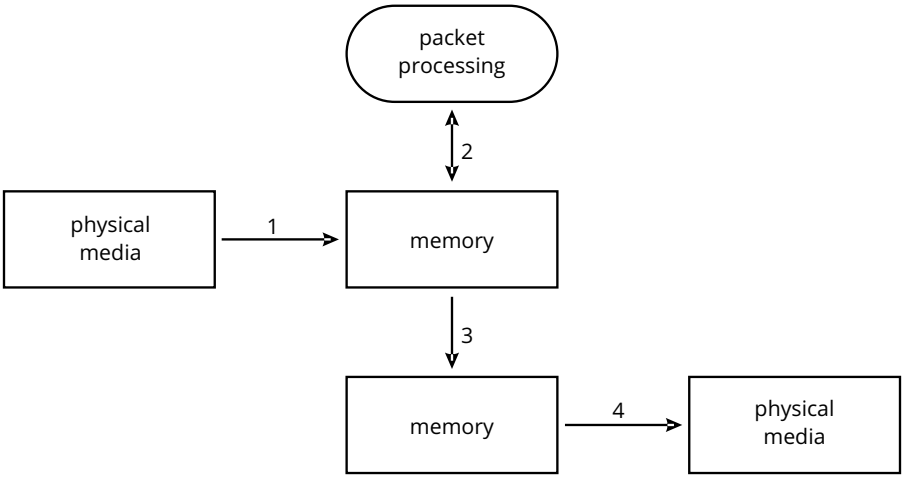


Figure 7-1 *Moving a Packet Through a Network Device*

In Figure 7-1, there are four distinct steps:

1. The packet needs to be copied off the physical media and into memory within the device; this is sometimes called clocking the packet off the wire.
2. The packet needs to be processed, which generally means determining the correct outbound interface and modifying the packet in any way necessary. For instance, in a router, the lower layer header is stripped off and replaced with a new one; in a stateful packet filter, the packet may be dropped based on internal state; etc.
3. The packet needs to be copied from the inbound to the outbound interface. This often involves a trip across an internal fabric, or bus. Some systems skip this step by using a single memory pool for both inbound and outbound interfaces; these are called shared memory systems (one thing about network engineering you will notice is the names of things either tend to be too clever or too obvious).
4. The packet needs to be copied back onto the outbound physical media; this is sometimes called clocking the packet onto the wire.

Note

Smaller systems, particularly those focused on fast, consistent packet switching, will often use shared memory to transfer a packet from one interface to another. The time required to copy a packet in memory is often larger than the speed at which the interfaces operate; shared memory systems avoid this in memory copying of packets.

The problem space discussed in the sections that follow, then, consist of this:

How are packets which need to be forwarded by the network device carried from the inbound to the outbound physical media, and how are packets exposed to processing along this path?

Each of the following sections discusses one part of the solution to this problem.

Physical Media to Memory

The first step in processing a packet through a network device is to copy the packet off the wire and into memory. Figure 7-2 is used to illustrate the process.

There are two steps in Figure 7-2:

Step 1. The physical media chipset (the *PHY chip)* will copy each time (or logical) slot from the physical media, which represents a single bit of data, into a memory location. This memory location is actually mapped into a receive ring, which is a set of memory locations (packet buffer) set aside for the sole purpose of receiving packets being clocked off the wire. The receive ring, and all packet buffer memory, is normally carved out of a single kind of memory accessible by (shared by) all the switching components on the receiving end of the line card or device.

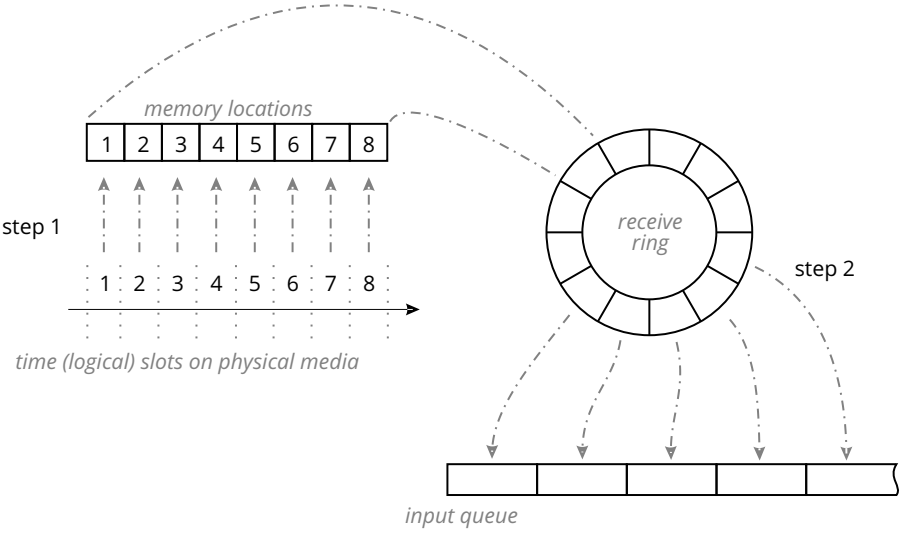


Figure 7-2 *Clocking a Packet into Memory*

Note

A ring buffer is used based on a single pointer, which is incremented each time a new packet is inserted into the buffer. For instance, in the ring shown in Figure 7-2, the pointer would begin at slot 1 and increment through the slots as packets are copied into the ring buffer. If the pointer reaches slot 7, and a new packet arrives, the packet will be copied into slot 1—regardless of whether or not the contents of slot 1 have been processed.

In packet switching, the most time-consuming and difficult task is copying packets from one location to another; this is avoided as much as possible through the use of pointers. Rather than moving a packet in memory, a pointer to the memory location is passed from process to process within the switching path.

Step 2. Once the packet is clocked into memory, some local processor is interrupted. During this interrupt, the local processor will remove the pointer to the packet buffer containing a packet from the receive ring and place a pointer to an empty packet buffer onto the receive ring. The pointer is placed on a separate list called the input queue.

Processing the Packet

Once the packet is in the input queue, it can be processed. Processing can be seen as a chain of events, rather than a single event; Figure 7-3 illustrates.

Some processing needs to take place before the packet is switched, such as Network Address Translation, because it changes some information about the packet used in the actual switching process. Other processing can take place after the switch.

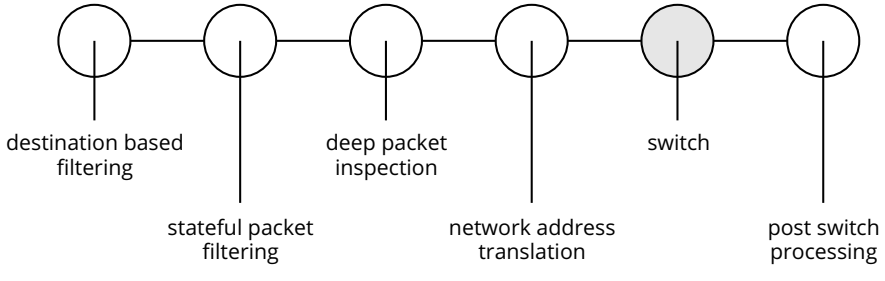


Figure 7-3 *Packet Switching Process*

Switching

Switching a packet is a somewhat simple operation:

1. The switching process looks up the destination Media Access Control (MAC), or physical device, address in a forwarding table (in switches this is sometimes called the bridge learning table, or just the bridge table).
2. The outbound interface is determined based on the information in this table.
3. The packet is moved from the input queue to the output queue.

The packet is not modified in any way during the switching process; it is copied from the input queue to the output queue.

> **Note**
>
> How is the forwarding table built? By a control plane. Part II of this book considers control planes in some detail.

Routing

Routing is a more complex process than switching; Figure 7-4 illustrates.

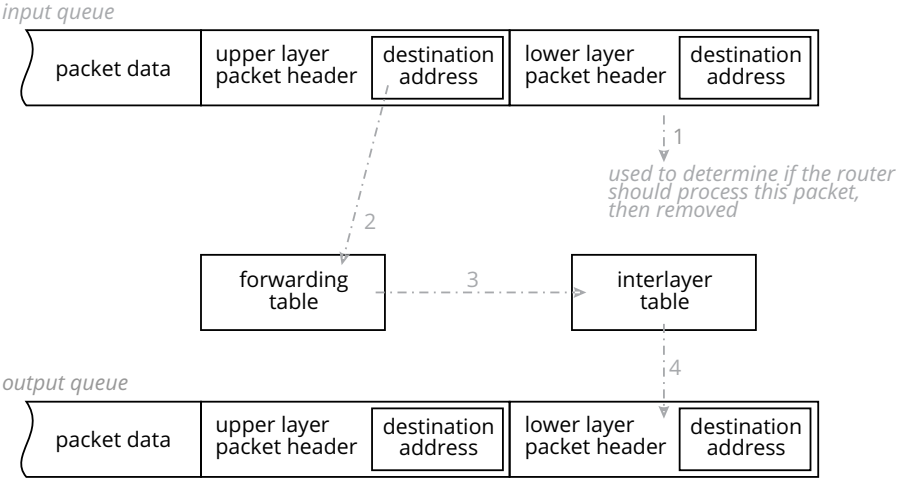


Figure 7-4 *Routing a Packet*

In Figure 7-4, the packet begins on the input queue. The switching processor then

1. Removes (or ignores) the lower layer header (for instance, the Ethernet framing on the packet). This information is used to determine whether or not the router should receive the packet, but is not used during the actual switching process.
2. Looks up the destination address (and potentially other information) in the forwarding table. The forwarding table relates the destination of the packet to the next hop of the packet. The next hop can either be the next router in the path toward the destination or the destination itself.
3. The switching processor then examines an interlayer discovery table (such as those considered in Chapter 6, "Interlayer Discovery"), to determine the correct physical address to which to send the packet to bring the packet one hop closer to the destination.
4. A new lower layer header is built using this new lower layer destination address and copied onto the packet. Normally, the lower layer destination address is cached locally, along with the entire lower layer header. The entire header is rewritten in a process called the MAC header rewrite.

The entire packet is now moved from the input queue to the output queue.

Why Route?

Because routing is a more complex process than switching, why route? Figure 7-5 will be used to illustrate.

There are at least three specific reasons to route, rather than switch, in a network. Using the network in Figure 7-5 as an example:

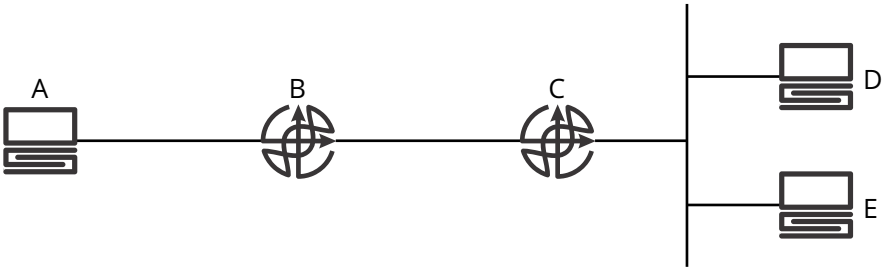


Figure 7-5 *Why Route?*

- If the [B,C] link is a different kind of physical media than the two links connecting to hosts, with different encoding, headers, addressing, etc., then routing will allow A and D to communicate without worrying about these differences in the link types. This could be overcome in a purely switched network through header translation, but header translation doesn't really take any less work than routing in the switching path, so there is little point in *not* routing to solve this problem. Another solution might be for every physical media type to agree on a single addressing and packet format, but given the constant advances in physical media, and the many different kinds of physical media, this seems like an unlikely solution.
- If the entire network were switched, B would need to know full reachability information for D and E; specifically, D and E would need to know the physical or lower layer addresses for each device connected to the host segment beyond C. This might not be a big problem in a smaller network, but in larger networks, with hundreds of thousands of nodes, or the global Internet, this will not scale—there is simply too much state to manage. It is possible to aggregate reachability information with lower layer addressing, but it is more difficult than using a higher layer address assigned based on the device's topological attachment point, rather than an address assigned at the factory that uniquely identifies the interface chipset.
- If D sends a broadcast to "all devices on segment," A will receive the broadcast if B and C are switches, but not if B and C are routers. Broadcast packets cannot be eliminated, as they are an essential part of just about every transport protocol, but in purely switched networks, broadcasts present a very hard-to-solve scaling problem. Broadcasts are blocked (or rather consumed) at a router.

Note

In the commercial networking world, the terms *routing* and *switching* are often used interchangeably. The reason for this is primarily marketing history; *routing* always originally meant "switched in software," while *switching* always meant "switched in hardware." As packet switching engines capable of rewriting a MAC header in hardware became available, they were called "Layer 3 switches," which was eventually shortened to just *switch*. Most data center "switches," for instance, are actually routers, as they do perform a MAC header rewrite on forwarded packets. If someone calls a piece of equipment a switch, then it is best to clarify whether it is a Layer 3 switch (properly a router) or a Layer 2 switch (properly a switch).

> **Note**
>
> The terms *link* and *connection* are used interchangeably here; a link is a physical or virtual wired or wireless connection between two devices.

Equal Cost Multipath

In some network designs, engineers will introduce parallel links between two network nodes. If you assume these parallel links are equal in bandwidth, latency, and so on, they are said to be equal cost. In this scenario, the links are said to be equal cost multipath (ECMP).

In networking, there are two variants seen frequently on production networks. They behave similarly but are different in how the links are grouped and managed by the network operating system.

Link Aggregation

Link aggregation schemes take multiple physical links and bundle them into a single virtual link. For purposes of routing protocols and loop prevention algorithms such as spanning tree, a virtual link is treated as if it were a single physical link.

Link aggregation is used to increase bandwidth between network nodes without having to replace slower physical links with faster ones. For instance, two 10Gbps links could be aggregated into a single 20Gbps link, thus doubling the potential bandwidth between the two nodes, as shown in Figure 7-6. The word *potential* was chosen carefully, as aggregated links do not, in practice, scale linearly.

The problem link aggregation faces is determining which packets should be sent down which member of the bundle. Intuitively, this might not seem like a problem. After all, it would seem to make sense to use the link bundle in a round-robin fashion. The initial frame would be sent down the first member of the bundle, the second frame down the second member, and so on, eventually wrapping back around to the first link bundle member. In this way, the link should be used perfectly evenly, and bandwidth should scale linearly.

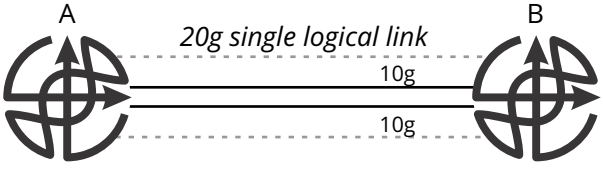


Figure 7-6 *Link Aggregation*

There are a very few real-life implementations where aggregated links are used on a round-robin basis like this because they run the risk of delivering out-of-order packets. Assume Ethernet frame one is sent down link member one, and frame two is sent down link member two immediately after. For whatever reason, frame two gets to the other end before frame one. The packets that these frames contain will be delivered to the receiving hosts out of order—packet two before packet one. This is a problem because a computational burden is now placed on the host to reorder the packets so the entire datagram can be properly reassembled.

Therefore, most vendors implement flow hashing to ensure the entirety of a traffic flow uses the same bundle member. In this way, there is no risk of a host receiving packets out of order, as they will be sent sequentially across the same link member.

Hash Algorithms

A hash is a simple concept that is actually quite difficult to implement in a useful way: a hash takes a string of numbers of any size and returns a fixed length number, or hash, that (more or less) uniquely represents the original string. The simple-to-implement part is this: one rather naïve hash is to simply add the numbers in a set of numbers until you reach a single digit, calling the result the hash. For instance:

```
23523
2 + 3 + 5 + 2 + 3 == 15
1 + 5 == 6
```

Hence, the number 23523 can be *represented* as 6. One curious property of the hash is there is no way to determine, from the hash, what the original number was—this is one of the essential observations of many uses for the hash. If I share a number with some third party, and that party then shares it with you, you can ask me for the hash of the number (without telling me what the actual number is!), and you can verify the number is the same by verifying the hash I give you matches the one you calculate.

The hash here is naïve because it is too easy to obtain a *collision*. In other words, there are many different sets of numbers that will result in a hash of 6 given the same process, such as 222, 33, 111111, and (probably) an almost infinite number of others. In some situations, collisions are extremely undesirable. For instance, if you are trying to store pairs of numbers, where you look up the first number to find the second (an indexing problem), then you want to minimize collisions. Once you have computed the hash of the number you are

using as the index, you do not want to have the result point to a hash bucket with a lot of entries, as each entry in the bucket needs to be searched to find the index. In the extreme case, every number will index into a single hash bucket, resulting in the hash being completely ineffective as a sorting tool.

In other cases, such as the load-sharing example given here, it is more important to make certain the hash spreads entries out among the available buckets as evenly as possible. You want to make certain each bucket contains about the same number of entries, as each bucket represents a single link, and you want the links to each receive the traffic of about the same number of destinations.

Flow hashing works by performing a mathematical operation on two or more static components of a flow, such as source and destination MAC addresses, source and destination Internet Protocol (IP) addresses, or Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) port numbers to compute a link member the flow will use. Because the characteristics of the flow are static, the hashing algorithm results in an identical computation for each frame or packet in a traffic flow, guaranteeing the same link will be used for the life of the flow.

While flow hashing solves the out-of-order packet problem, it introduces a new problem. Not all flows are the same size. Some flows use a high amount of bandwidth, such as those used for file transfers, backups, or storage; these are sometimes called *elephant flows*. Other flows are quite small, such as those used to load a web page or communicate using voice over IP; these are sometimes called *mouse flows*. Because flows are different sizes, some link members might be running at capacity, while others are underutilized.

This mismatch in utilization brings us back around to the point about linear scaling. If frames were load-balanced across an aggregated link bundle perfectly evenly, then adding new links to the bundle would evenly multiply capacity. However, hashing algorithms combined with the unpredictable volume of traffic flows mean bundled links will not be evenly loaded.

The job of the network engineer is to understand the type of traffic flowing through the aggregated bundle and choose an available hashing algorithm that will result in the most even load distribution. For instance, some considerations might be

- Are many hosts in the same broadcast domain communicating with one another across the aggregated link? Hashing against the MAC addresses found in the Ethernet frame header is a possible solution, because the MAC addresses will be varied.

- Are a small number of hosts communicating to a single server across the aggregated link? There might not be enough variety of either MAC addresses or IP addresses in this scenario. Instead, hashing against TCP or UDP port numbers might result in the greatest variety and subsequent traffic distribution across the aggregated links.

Link Aggregation Control Protocol

When bundling links together, you must consider the network devices on either end of the link and take special care to allow the link bundle to be formed while maintaining a loop-free topology. The most common way of addressing this issue is by using industry standard Link Aggregation Control Protocol (LACP), codified as Institute of Electrical and Electronic Engineers (IEEE) standard 802.3ad.

On links designated by a network engineer, LACP advertises its intent to form an aggregated link to the other side. The other side, also running LACP, accepts this advertisement if the announced parameters are valid, and forms the link. Once the link bundle has been formed, the aggregated link is placed into a forwarding state. Network operators can then query LACP for status on the aggregated link and the state of link members.

LACP is also aware when a member of the link bundle goes down, as control packets no longer flow across the failed link. This capability is useful, as it allows the LACP process to notify the network operating system to recalculate its flow hashes. Without LACP, it might take the network operating system a longer time to become aware of the failed link, causing traffic to be hashed to a link member that is no longer a valid path.

Other link aggregation control protocols exist. It is also possible in some scenarios to create link bundles manually without the protection of a control protocol; however, LACP dominates as the standard in use by networking vendors as well as host operating systems and hypervisor vendors for link aggregation.

Multichassis Link Aggregation

Multichassis Link Aggregation (MLAG) is a feature offered by some network vendors allowing a single aggregated link bundle to span two or more network switches. To facilitate this, a vendor's special control protocol will run between the MLAG member switches, making multiple network switches act as if they are one switch as far as LACP, Spanning Tree Protocol (STP), and any other protocols are concerned.

The usual justification for MLAG is physical redundancy, where a network engineer requires a lower layer (such as Ethernet) adjacency between network devices (instead of a routed connection), and also requires the link bundle to remain up if

the remote side of the link fails. Spreading the link bundle between two or more switches allows this requirement to be met. Figure 7-7 illustrates.

While many networks operate some flavor of MLAG in production, many others have shied away from the technology, at least partially because MLAG is proprietary; there is no such thing as multivendor MLAG. Better network design trends away from widely dispersed switched domains, a scenario that benefits from MLAG. Instead, network design is trending toward constrained switched domains interconnected through routing, obviating the need for MLAG technologies.

Routed Parallel Links

Routed control planes, called routing protocols (see the chapters in Part II of this book for more information on routing and loop-free path calculation), sometimes compute a set of multiple paths through a network with equal costs. In the case of routing, multiple links with the same cost may not even connect a single pair of devices; Figure 7-8 illustrates.

In Figure 7-8, there are three paths:

- [A,B,D] with a total cost of 10
- [A,D] with a total cost of 10
- [A,C,D] with a total cost of 10

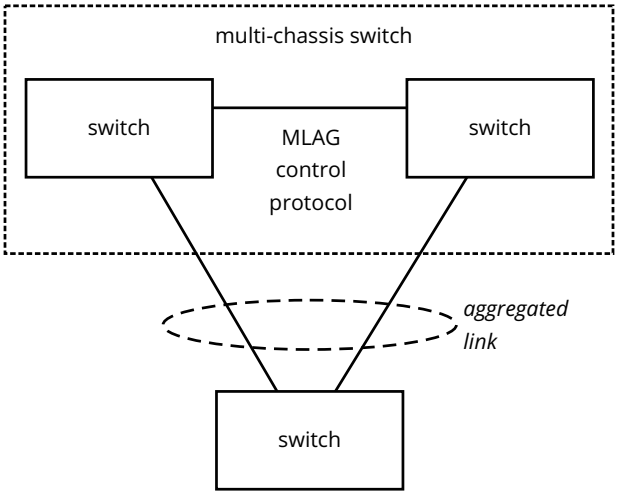


Figure 7-7 *Multichassis Link Aggregation*

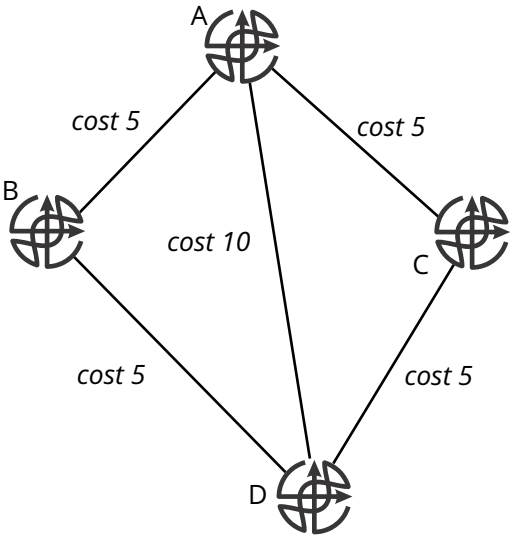


Figure 7-8 *Routed ECMP*

Because these three paths have the same cost, they may all three be installed in the local forwarding table at A and D. Router A, for instance, may forward traffic over any one of these three links toward D. When a router has multiple options to reach the same destination, how does it decide which physical path to take?

As with lower layer ECMP, the answer is hashing. Routed ECMP hashing can be performed on a variety of fields. Common fields to hash against include source or destination IP addresses and source or destination port numbers. The hashing results in a consistent path being selected for the duration of an L3 flow. Only in the case of a link failure would the flow need to be rehashed and a new forwarding link chosen.

Packet Processing Engines

The steps involved in routing a single packet may seem very simple—look up the destination in a table, build (or retrieve) a MAC header rewrite, rewrite the MAC header, and then place the packet on the correct queue for an outbound interface. As simple as this might be, it still takes time to process a single packet. Figure 7-9 illustrates three different *paths* through which a packet may be switched in a network device.

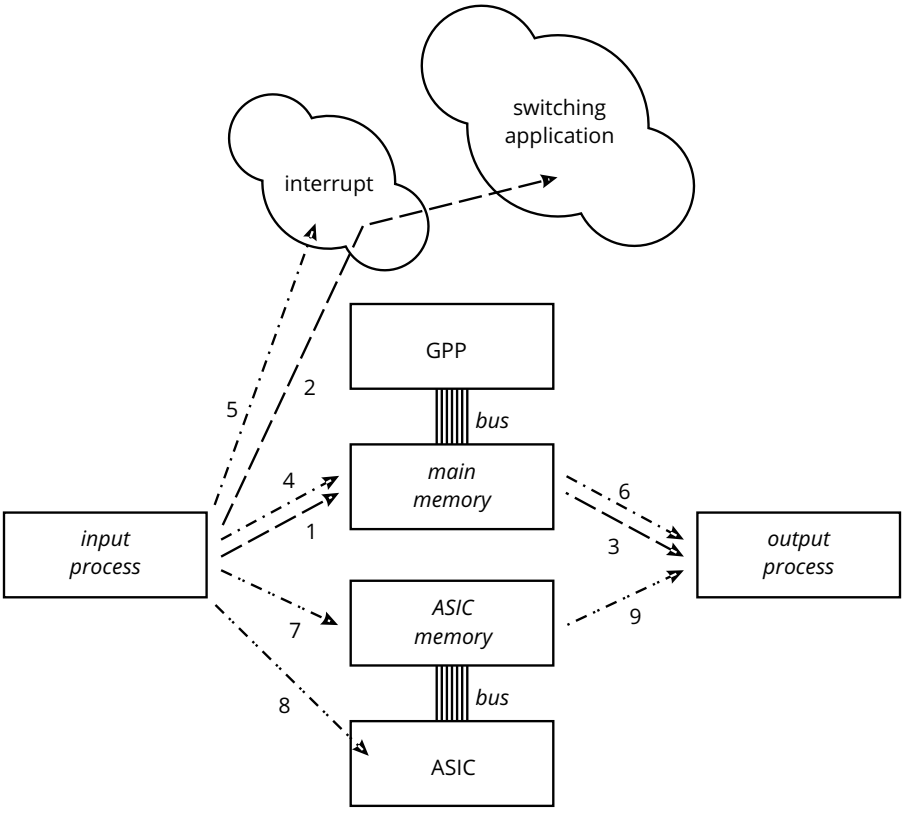


Figure 7-9 *Switching Paths*

Figure 7-9 illustrates three different switching paths through a device; these are not the *only possible* switching paths, but they are the most common ones. The first path processes packets through a software application running on a general-purpose processor (GPP), and consists of three steps:

1. The packet is copied off the physical media into main memory, as described in the sections above.
2. The physical signal processor, the PHY chip, sends a signal to the GPP (probably, but not necessarily, the main processor in the network device), called an interrupt.
   a. The interrupt causes the processor to stop other tasks (this is why it is called an interrupt) and run a small piece of code that will schedule another process, the switching application, to run later.

   b. When the switching application runs, it will do the appropriate lookups and make the appropriate modifications to the packet.

3. Once the packet has been switched, it is copied out of main memory by the outbound processor, as described in the following sections.

Switching a packet through a process in this way is often called process switching (for obvious reasons), or sometimes the slow path. No matter how fast the GPP is, to reach full line rate switching on higher-speed interfaces requires a lot of tuning—to the point of being almost impossible. The second switching path shown in Figure 7-9 was designed to process packets more quickly:

4. The packet is copied off the physical media into main memory, as described in the previous sections.
5. The PHY chip interrupts the GPP; the interrupt handler code, rather than calling another process, actually processes the packet.
6. Once the packet has been switched, the packet is copied from main memory into the output process, as described in the text that follows.

This process is often called interrupt context switching, for obvious reasons; many processors can support switching packets fast enough to carry packets between low and moderate rate interfaces in this mode. The switching code itself must be highly optimized, of course, because switching the packet causes the processor to stop executing any other tasks (such as processing a routing protocol update). This was originally—and is still sometimes—called the fast switching path.

For truly high-speed applications, the process of switching packets must be offloaded from the main processor, or any kind of GPP, and onto a specialized processor designed for the specific task of processing packets. Sometimes these processors are called Network Processing Units (NPUs), much like a processor designed to handle just graphics is called a Graphics Processing Unit (GPU). These specialized processors are a subset of a broader class of processors called Application-Specific Integrated Circuits (ASICs), and are often just called ASICs by engineers. Switching a packet through an ASIC is shown as steps 7 through 9 in Figure 7-9:

7. The packet is copied off the physical media into the ASIC's memory, as described in the previous sections.

8. The PHY chip interrupts the ASIC; the ASIC handles the interrupt by switching the packet.
9. Once the packet has been switched, the packet is copied from the ASIC's memory into the output process, as described next.

Many specialized packet processing ASICs have a number of interesting features, including

- Internal memory structures (registers) configured specifically to handle the various kinds of addresses used in networks
- Specialized instruction sets designed to handle various packet processing requirements, such as examining the inner headers being carried in a packet, and rewriting the MAC header
- Specialized memory structures and instruction sets designed to store and look up destination addresses to speed packet processing
- The ability to recycle a packet through the packet pipeline in order to perform operations that cannot be supported in a single pass, such as deep packet inspection or specialized filtering tasks

Across the Bus

In smaller network devices with just one network process (the ASIC or NPU, as described previously), moving a packet from the input queue to the output queue is simple. The input and output interfaces both share a common pool of packet memory, so a pointer to the packet can be moved from one queue to the other.

To reach higher port counts and larger-scale devices—particularly chassis devices—there must be an internal bus, or fabric, that connects the input and output packet processing engines. One common type of fabric used to interconnect packet processing engines within a network device is a crossbar fabric; Figure 7-10 illustrates.

The size and structure of the crossbar fabric are dependent on the number of ports connected. If there are more ports in the switch than feasible to connect via a single crossbar fabric, then the switch will use multiple crossbar fabrics. A common topology for this kind of fabric is a multistage Clos connecting the ingress and egress crossbar fabrics together. You might think of this as a crossbar of crossbars.

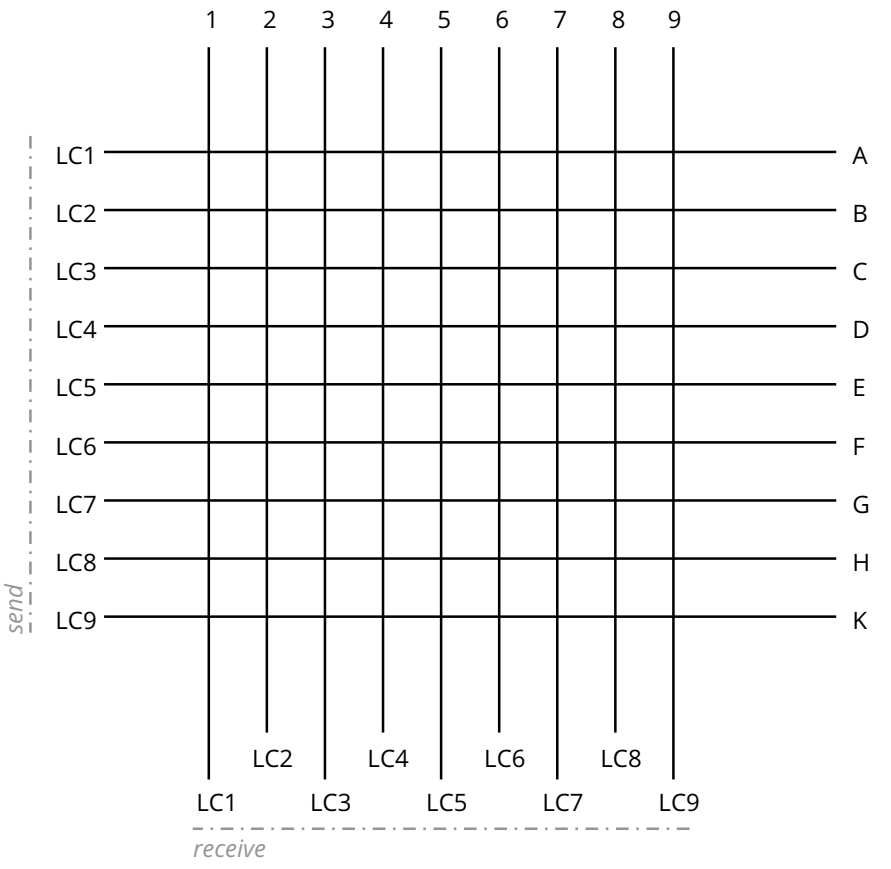


Figure 7-10 *A Crossbar Fabric*

> **Note**
>
> Spine and leaf fabrics, which are a form of Clos, are considered in Chapter 25, "Disaggregation, Hyperconvergence, and the Changing Network."

A crossbar fabric requires a sense of time (or rather a fixed time slot) and a scheduler to work. At each interval of time, one output (send) port is connected to one input (receive) port, so that during this time period the sender can transmit a packet, frame, or set of packets to the receiver. The scheduler "connects" the correct cross

points on the crossbar fabric for transmissions to take place during the correct time period. For instance:

- Line card 1 (LC1) would like to send a packet to LC3.
- LC3 would like to send a packet to LC5.

During the next time cycle, the scheduler can connect row A to column 1 ("make" the connection at A3) and connect row C to column 5 ("make" the connection at C5) so a communication channel is set up between these pairs of line cards.

Crossbars and Contention

What happens if two transmitters want to send a packet to a single receiver? For instance, if during one period of time both LC1 and LC2 want to send a packet to LC9 across the crossbar fabric? This is called contention, and is a situation that must be handled by the fabric scheduler. Which of the two ingress ports should be allowed to send their traffic to the egress port? And where are the ingress traffic queues in the meantime?

One option is for the packets to be stored in an input queue; switches that use this technique are called input-queued switches. These kinds of switches suffer from head-of-line (HOL) blocking. HOL blocking is what happens when the packet at the head of the line, waiting to be forwarded across the fabric, blocks the other packets queued up behind it.

Another option is for the switch to leverage multiple virtual output queues (VOQs) per input port.

VOQs give a crossbar fabric multiple places to stash ingress packets while they are waiting to be delivered to their egress ports. In many switch designs, one VOQ exists per output port for which input traffic is destined. Therefore, an input port can have several packets queued in several different VOQs, assuming several different egress ports.

Each of these VOQs is eligible to be serviced during a single clock cycle. This means HOL blocking is eliminated, because several different packets from the same input queue can be passed through the crossbar fabric at the same time. Rather than a single queue existing for an input port, there are several different queues. Think of it as additional checkout lines being opened at the grocery store.

Even with VOQs, the potential remains for contention across the crossbar fabric. The most common example is where two or more ingress packets need to leave the

switch via the same egress port at the same time, or more precisely, on the same clock cycle. An egress port can only send one packet per clock cycle.

Determining which ingress queue will get to deliver traffic to the egress port first is an algorithm determined by the switch manufacturer to make the maximum use of the hardware. iSLIP is one scheduling algorithm used by switches to solve this problem.

An Overview of the iSLIP Algorithm

The iSLIP algorithm arbitrates crossbar fabric contention, scheduling traffic so the network device achieves nonblocking throughput. For the purposes of this discussion, it is helpful to scrutinize iSLIP in its simplest form by reviewing what happens when the iSLIP algorithm executes once.

There are three crucial events that take place during an iSLIP execution:

1. **Request.** All input points (ingress) on the crossbar fabric with queued traffic ask their output points (egress) if they can send.
2. **Grant.** Each output point that received a request must determine which input point will be allowed to send. If there is a single request, then a grant is awarded with no further deliberation. However, if there are multiple requests, the output point must determine which input point can send. This is done via round-robin, where one request is awarded a grant, a subsequent request is awarded a grant during the next execution of iSLIP, and so on in a circular fashion. When the decision has been made for this particular execution of iSLIP, each output point sends its grant message, effectively signaling permission to send, to the appropriate input point.
3. **Accept.** An input point considers the grant messages it has received from output points, choosing a grant in round-robin fashion. Upon selection, the input notifies the output that the grant has been accepted. If and only if the output point is notified the grant was accepted will the output point move on to the next request. If there is no accept message received, then the output point will attempt to service the previous request during the next execution of iSLIP.

Understanding the request, grant, and accept process gives us insight into how packets can be delivered simultaneously through a crossbar fabric without colliding. However, if you ponder a complex set of inputs, VOQs, and outputs, you might

realize a single iSLIP run doesn't schedule as many packets for delivery as it could have after only a single execution.

Certainly, some inputs were granted outputs and some packets can be forwarded, but it is possible some outputs were never matched with a waiting input. In other words, if you limit iSLIP to a single execution per clock cycle, we'd be leaving available egress bandwidth unused.

Therefore, the normal practice is to run iSLIP through multiple iterations. The result is the number of input-to-output matches is maximized. More packets can be sent across the crossbar fabric at a time. How many times does iSLIP need to run to maximize the number of packets that can be switched through the crossbar fabric in a clock cycle? Research suggests that for the traffic patterns prevalent on most networks, running iSLIP four times matches inputs and outputs across the crossbar fabric the best. Executing iSLIP more than four times does not result in a meaningfully larger number of matches. In other words, there is nothing to be gained running iSLIP five, six, or ten times in most network environments.

Moving Beyond iSLIP

This discussion has assumed, so far, that the traffic flowing through the crossbar fabric was all of equal importance. However, in modern data centers, certain traffic classes are prioritized over the other. For instance, Fibre Channel over Ethernet (FCoE) storage frames need to traverse the fabric in a lossless manner, while a TCP session falling into a scavenger QoS class does not.

Does iSLIP handle traffic with different priorities, granting some requests before others? Yes, but in a modified form of the algorithm we've looked at. Variants to iSLIP include Prioritized, Threshold, and Weighted iSLIP.

Beyond iSLIP, used here merely as a convenient example of contention management, vendors will write their own algorithms to suit their own crossbar fabric's hardware capabilities. For example, this section only considered an input-queued crossbar fabric, but many crossbar fabrics offer output-queuing on the egress side of the crossbar as well.

Memory to Physical Media

Once the packet is carried across the bus to the outbound line card, or the pointer on the packet buffer is moved from the input queue to the output queue, there is still work for the network device to do. Figure 7-11 illustrates.

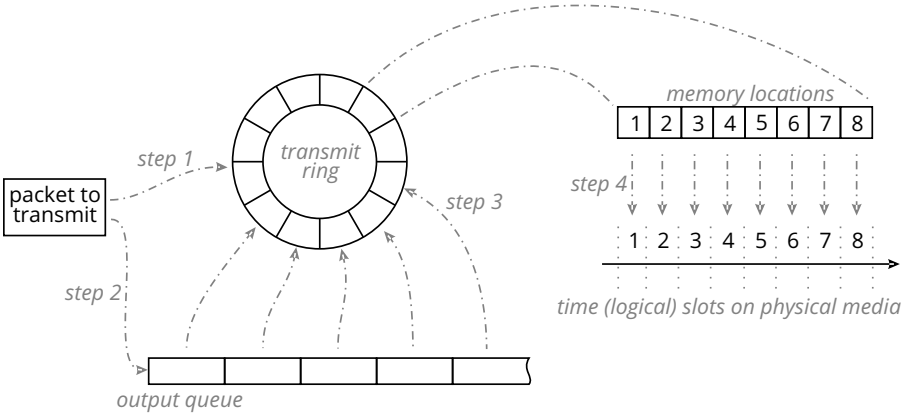


Figure 7-11 *Clocking a Packet Back onto the Wire*

Note the ring shown in Figure 7-11 is the transmit ring, rather than the receive ring. There are four steps in Figure 7-11:

Step 1. The packet is passed to the transmit side of the router for forwarding. There may be post switch processing that needs to be done here, depending on the platform and specific features; these are not shown in this illustration. An attempt will first be made to place the packet directly on the transmit ring, where it can be transmitted. If the ring already has a packet on it, or if the ring is full (depending on the implementation), the packet will not be placed on the transmit ring. If the packet is placed on the transmit ring, step 2 is skipped (which means the packet will not be processed using any outbound Quality of Service [QoS] rules). Otherwise, the packet is placed on the output queue, where it will await being transferred to the transmit ring.

Step 2. If the packet cannot be placed on the transmit ring, it will be placed on the output queue for holding for some later time.

Step 3. Periodically, the transmit code will move packets from the output queue to the transmit ring. The order in which packets are taken from the output queue will depend on the QoS configuration; see Chapter 8, "Quality of Service," for more information on how QoS is applied to queues in various situations.

Step 4. At some point after the packet has been moved to the transmit ring, the transmit PHY chip, which reads each bit from the packet buffer, encodes it into the proper format for the outbound physical media type and copies the packet onto the wire.

Final Thoughts on Packet Switching

The details of packet switching might seem mired in minutiae. After all, does it matter exactly how a packet or frame moves between two devices? Is it really all that critical to comprehend serialization and deserialization, equal cost multipath, crossbar fabric contention, transmit rings, and the like?

In a certain sense, these details don't matter to the average network engineer. When a network device is doing its job moving data through it, the actual processes followed by the switch to get that job done are trivialities. "It just works."

However, switching internals often factor greatly into network design. For example, consider port-to-port latency. In some high-traffic networks, the amount of time it takes for a switch to move a frame from ingress port to egress port makes a difference in overall application performance. In modern switches, port-to-port latency is measured in single microseconds or hundreds of nanoseconds. If one switch gets the job done in 1 microsecond, while another can do it in 400 nanoseconds, that can impact a hardware choice.

Another consideration is troubleshooting. What happens when a network device does not appear to be forwarding all of the packets it receives, i.e. there is more ingress than egress? Small amounts of packet loss in a network fabric are troublesome to track down. Understanding a network device's internal packet switching process shines a great deal of light on where the breakdown might be happening.

Therefore, don't dismiss packet switching as "too close to the wires" to be relevant to the aspiring networker. Rather, embrace a knowledge of packet switching for the deep insights into overall network performance that it supplies.

Further Reading

"1.5. Basics of How Operating Systems Work." *Operating Systems Study Guide*. Accessed April 22, 2017. http://faculty.salina.k-state.edu/tim/ossg/Introduction/OSworking.html.

Bollapragada, Vijay, Russ White, and Curtis Murphy. *Inside Cisco IOS Software Architecture*. Indianapolis, IN: Cisco Press, 2000.

BSTJ 18: 1. January 1939: Crossbar Dial Telephone Switching System. (Scudder, F. J.; Reynolds, J. N.), 1939. http://archive.org/details/bstj18-1-76.

"Cisco Nexus 5548P Switch Architecture." *Cisco*. Accessed July 29, 2017. http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5548p-switch/white_paper_c11-622479.html.

"Fast Ethernet | Integrating 100mbps into Existing 10mbps Networks." *Savvius*. Accessed April 22, 2017. https://www.savvius.com/resources/compendium/fast_ethernet/overview.

Heineman, George T., Gary Pollice, and Stanley Selkow. *Algorithms in a Nutshell: A Practical Guide*. 2nd edition. O'Reilly Media, 2016.

Inniss, Daryl, and Roy Rubenstein. *Silicon Photonics: Fueling the Next Information Revolution*. 1st edition. Morgan Kaufmann, 2016.

"Intel Ethernet Switch Family Hash Efficiency." Intel, April 2009. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ethernet-switch-hash-efficiency-paper.pdf.

"Interrupt." *Wikipedia*, Accessed February 3, 2017. https://en.wikipedia.org/w/index.php?title=Interrupt&oldid=763436239.

Kloth, Axel K. *Advanced Router Architectures*. Boca Raton, FL: CRC Press, 2005.

Konheim, Alan G. *Hashing in Computer Science: Fifty Years of Slicing and Dicing*. 1st edition. Wiley-Interscience, 2011.

Lekkas, Panos. *Network Processors: Architectures, Protocols and Platforms*. 1st edition. New York: McGraw-Hill Education, 2003.

Meiners, Chad R., Alex X. Liu, and Eric Torng. *Hardware Based Packet Classification for High Speed Internet Routers*. 2010 edition. New York: Springer, 2010.

Noubir, Guevara. "Signal Encoding Techniques." Accessed April 22, 2017. http://www.ccs.neu.edu/home/noubir/Courses/CS6710/S12/slides/signals-encoding.pdf.

Stringfield, Nakia, Russ White, and Stacia McKee. *Cisco Express Forwarding*. 1st edition. Indianapolis, IN: Cisco Press, 2007.

Thakur, Dinesh. "Encoding Techniques and Codec." *Computer Notes*. Accessed April 22, 2017. http://ecomputernotes.com/computernetworkingnotes/communication-networks/encoding-techniques-and-codec.

"Understanding IEEE 802.3ad Link Aggregation—Technical Documentation—Support—Juniper Networks," March 26, 2013. https://www.juniper.net/documentation/en_US/junose14.2/topics/concept/802.3ad-link-aggregation-understanding.html.

Review Questions

1. What happens if one end of a link is configured as a bundle and the other end is not? Specifically, what happens if one device thinks STP is running and the other does not?

2. Why is flow hashing typically used as opposed to round-robin as a forwarding algorithm in ECMP?
3. What is the purpose of a multistage fabric? Provide an example.
4. Briefly summarize techniques found in crossbar fabrics to mitigate contention.
5. The iSLIP algorithm has steps of Request, Grant, and Accept. In a single sentence for each, explain what happens in each step.
6. How many times does iSLIP need to run before it is no longer effective in improving input-to-output matches?
7. How many packets can be placed on the transmit ring at a time?
8. Why not make the transmit and receive rings large enough to prevent any packet from ever being overwritten because the packets being held in the ring buffer are not processed quickly enough? What are the tradeoffs in terms of switching speed through the switch, memory utilization, and other factors?
9. Research and describe the impact of a broadcast storm in a network. How does routing prevent broadcast storms?
10. What are some advantages of using MLAG to build very large, flat networks without routing? What are some disadvantages?

Chapter 8

Quality of Service

Learning Objectives

After reading this chapter, you should understand:

- Why Quality of Service is necessary in a network, even if the network has plenty of bandwidth
- How traffic is classified and why classification is normally done as few times as possible
- The relationship between Quality of Service, Class of Service, and Type of Service
- How ToS markings are carried in a packet
- What a QoS trust boundary is
- ToS markings, and the translation of these markings at network boundaries
- Jitter and its impact on applications
- Fairness in queueing

On an ordinary day, the highway was wide enough to accommodate travelers. There were enough lanes. The speed limit was set to move traffic through the area quickly. The volume of cars was not excessive. Vehicles on this highway moved along effectively, moving down the road without having to jostle for position, stand on the brakes, weave in between lanes, or otherwise negotiate excessive traffic. That is, on an ordinary day.

This was not an ordinary day. On this day, the president was coming to town. The president was making a speech, and many people wanted to hear this speech. As the hour got closer to the president's speech, the ordinarily effective highway saw an increase in traffic. At first, this was not a concern. The highway rarely ran at capacity, and so an increase in traffic was manageable. Granted, there were more vehicles on the road, and they were running closer together. But this didn't cause any problems.

As the day wore on, and the time for the president's speech became quite close, the traffic had increased yet again. Now, there were problems. The highway was no longer able to carry the volume of traffic trying to run across it. Vehicles merging onto the highway found themselves stuck in lines at the on-ramp. Other vehicles were trapped on the highway, moving, albeit very slowly. Some vehicles gave up on using the highway, turning around and heading back home, hoping to catch the president's speech on television or via live stream.

The president's cavalcade of vehicles drove from the regional airport to the site of the speech. Their vehicles, too, were impacted by the congested highway. However, the presidential parade had more of something than any other vehicles on the road had—importance. To indicate their importance, they put on their emergency lights. Police escort vehicles, presidential protection detail, limousines, and threat response trucks all lit up in flashing red and blue.

The struggling highway traffic moved aside as the president's vehicles surged forward, heading down a priority lane to the site of the speech. Not everyone was going to make it to the speech, but the president couldn't be victimized by the traffic. No matter how overloaded the highway was, the president had to get through. The president was the one making the speech.

Defining the Problem Space

Network engineers frequently face the problem of too much traffic for too small of a link. In particular, in almost every path through a network, one link restricts the entire path, much as one intersection or one road restricts the flow of traffic. Figure 8-1 illustrates.

In Figure 8-1, A is communicating with G, and B is communicating with E. If each of these pairs of devices are using close to the available bandwidth on their local links ([A,C], [B,C], [F,G], and D,E]), assuming all the links are the same speed, the [C,D] link will be overwhelmed with traffic, becoming a choke point in the network.

When a link is congested, such as the [C,D] link in Figure 8-1, there is more traffic to be sent down the link than the link has capacity to carry. During times of congestion, a network device such as a router or switch must determine which traffic should be forwarded, which should be dropped, and in what order packets should

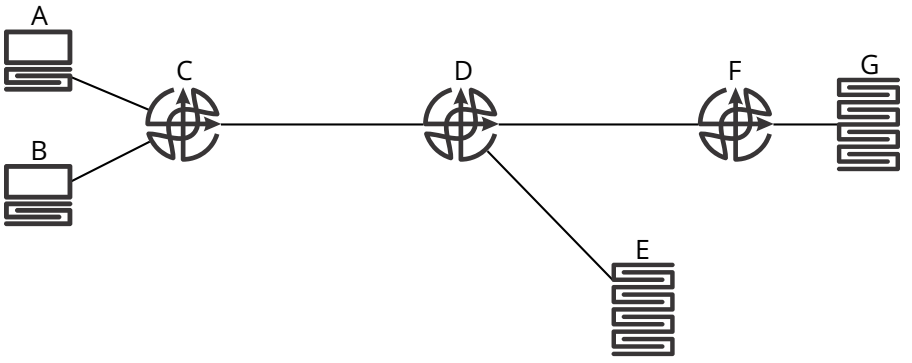


Figure 8-1 *Congestion Choke Point in Network Paths*

be forwarded. Various prioritization schemes have been constructed to address this challenge.

Managing link congestion by prioritizing some traffic classes over others comes under the broad heading of Quality of Service (QoS). The perception of QoS among network engineers is troubled for many reasons. For instance, many implementations, even recent ones, tend to be not as well thought out as they could be, especially in the way they are configured and maintained. Further, early schemes did not always work well, and QoS can often add to the problems in a network, rather than relieve them, and tends to be very difficult to troubleshoot.

For these reasons, and because the configuration required to implement prioritization schemes tends toward the arcane, QoS is often considered a dark art. To successfully implement a QoS strategy, you must classify traffic, define a queueing strategy for various traffic classes, and install the strategy consistently across all network devices that might experience link congestion.

While it is possible to become buried in the many different features and functions of QoS schemes and implementations, the result should always be the same. The president must deliver a timely speech.

Why Not Just Size Links Large Enough?

After thinking through the value proposition of QoS, an obvious reaction is to wonder why network engineers don't simply size links large enough to avoid congestion. After all, if links were large enough, congestion would disappear. If congestion disappeared, then the need to prioritize one traffic type over another would disappear. All traffic would be delivered, and all of these pesky problems rooted in insufficient bandwidth would be obviated. Indeed, overprovisioning is perhaps the best QoS of all.

Sadly, the overprovisioning strategy is not always an available option. Even if it were, the very largest links available can't overcome certain traffic patterns. Some applications will use as much bandwidth as available when transferring data, creating a point of congestion for other applications sharing the link. Others will transmit in micro-bursts, overwhelming network resources for a short time, and some transport mechanisms—such as the Transmission Control Protocol (TCP)—will intentionally congest a path occasionally in order to determine the best rate at which to send data. While a larger link can reduce the amount of time a congestion condition exists, in certain scenarios, there is no such thing as having enough bandwidth to meet all demands.

Most networks are built on a model of oversubscription, where some larger amount of aggregated bandwidth is shared at certain bottlenecks. For example, a Top of Rack (ToR) switch in a busy data center might have 48x10GbE ports facing hosts, but only have 4x40GbE ports facing the rest of the data center. This results in an oversubscription ratio of 480:160, which reduces to 3:1. Implicitly, the 160Gbps of data center facing bandwidth is a potential bottleneck—a congestion point—for the 480Gbps of host facing bandwidth. And yet, a 3:1 oversubscription ratio is common in data center switching designs. Why?

The ultimate answer is often money. It is often possible to design a network in which the edge ports match the available bandwidth. For instance, in the data center fabric given above, it is almost certainly possible to add enough link capacity to provide 480Gbps from the ToR into the fabric, but the cost may well be prohibitive. The network engineer needs to consider not only the costs of the port and fiber optics, but also the cost of additional power, and the cost of the additional cooling required to control the environment once the necessary additional devices have been added, and even the costs of additional rack space and floor weight.

Spending money to provide a higher fabric bandwidth may also be hard to justify if the network or fabric is rarely congested. Some congestion events are not frequent enough to justify an expensive network upgrade. Would a city spend millions or billions of dollars in transportation infrastructure improvements to ease traffic once a year when a politician comes to visit? No. Instead, other adjustments are made to handle the traffic problem.

For example, companies might most keenly experience this constraint in wide area networking, where links are leased from service providers (SPs). SPs make their money, in part, by connecting disparate geographies together for organizations that cannot afford to build out and operate long-distance fiber-optic cables on their own. These long-haul links normally offer much lower bandwidth than the shorter, local, links on a single campus, or even within a single building. A high-speed link within a campus or data center can easily overwhelm slower long-haul links.

Organizations will size long-haul (such as intersite, or even intercontinental) links as large as reasonably possible, but again, the key to keep in mind is money.

Long-haul links provided by SPs to other organizations are a costly, usually significant and oft scrutinized budget item. The more bandwidth being leased, the higher the costs tend to be. The result is a massive oversubscription, where WAN links are greatly bandwidth constrained when compared to the speeds available on a campus or inside a data center.

In a world of oversubscription and consequent congestion points, as well as temporary traffic patterns that need careful management, QoS traffic prioritization schemes will always be required.

Classification

QoS prioritization schemes act on different traffic classes, but what is a traffic class, and how is it defined?

Traffic classes represent aggregated groups of traffic. Data streams from applications requiring similar handling or presenting similar traffic patterns to the network are placed into groups and managed by a QoS policy (or Class of Service, CoS). This grouping is crucial, as it would be ponderous to define unique QoS policies for a potentially infinite number of applications. As a matter of practicality, network engineers will typically group traffic into four classes. More classes are certainly possible, and such schemes do exist in production networks. However, the management of the classification system and policy actions becomes increasingly tedious as the number of classes grows beyond four.

It is possible for each packet to be assigned to a particular CoS based on the source address, destination address, source port, destination port, size of the packet, and other factors. Assuming each application has its own profile, or set of characteristics, each application can be placed into a specific CoS, and acted on local QoS policy. The problem with this method of traffic classification is the classification is only locally significant—the classification action is relevant only to the device performing the classification.

Classifying packets in this way requires a lot of time, and processing each packet will take a lot of processing power. Because of this, it is still best not to repeat this processing at every device through which the packet passes. Instead, it is better to classify the traffic once, mark the packet at this single point, and act on this marking at every subsequent hop in the network.

> **Note**
>
> Even though packets and frames are distinct in networking, the term *packets* will be used in this chapter.

Various marking schemes have been designed and standardized, such as the 8-bit Type of Service (ToS) field included in the Internet Protocol version 4 (IPv4) header. Version 6 of this same protocol (IPv6) includes an 8-bit Traffic Class field serving a similar purpose. Ethernet frames use a 3-bit field as part of the 802.1p specification. Figure 8-2 illustrates the IPv4 ToS field.

In networking best practice, traffic classification should result in one action and one action only—marking. When a packet has been marked, the assigned value can be preserved and acted upon throughout the packet's entire journey through the network path. Classification and subsequent marking should be a "one-and-done" event in the life of a packet.

QoS best practice is to mark traffic as closely to the source as possible. Ideally, traffic will be marked at the point of ingress to the network. For example, traffic flowing into a network switch from a personal computer, phone, server, IoT device, etc. will be marked, and the mark will serve as the traffic classifier on the packet's journey through the network.

An alternate scheme to the ingress network device classifying and marking traffic is for the application itself to mark its own traffic. In other words, the packet is sent out with the ToS byte already populated. This brings up the problem of trust. Should an application be allowed to rank its own importance? In the worst-case scenario, all applications would selfishly mark their packets with values indicating the highest possible importance. If every packet is marked as being highly important, then in actuality, no packet is highly important. For one packet to be more important than any other, there must be differentiation. Traffic classes must have distinct levels of importance for QoS prioritization schemes to have any meaning.

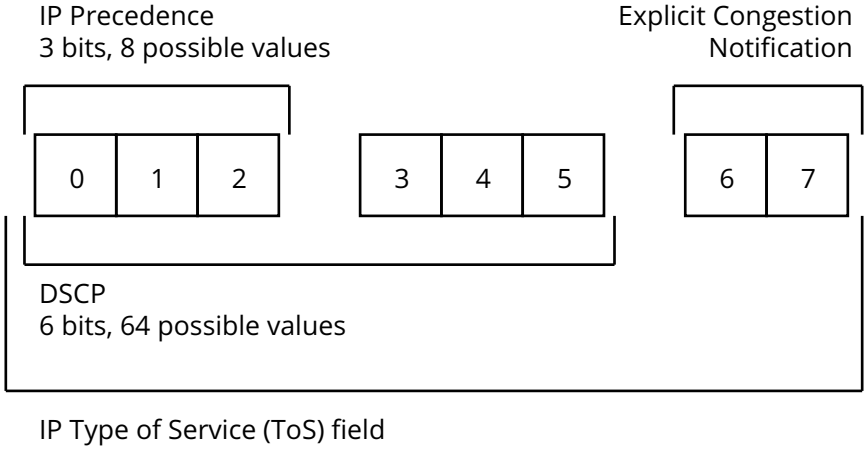


Figure 8-2 *Ethernet DSCP and IP ToS Fields*

To maintain control over traffic classification, all networks implementing QoS have trust boundaries. Trust boundaries allow the network to avoid a situation where all applications have marked themselves as important. Imagine what would happen on a congested road if every vehicle had flashing emergency lights—the truly important vehicles would not stand out.

In networking, some applications and devices are trusted to mark their own traffic. For example, IP phones are typically trusted to mark their streaming voice and control protocol traffic appropriately, meaning the marks that IP phones apply to their traffic are accepted by the ingress network device. Other endpoints or applications might be untrusted, meaning the packet's ToS byte is erased or rewritten on ingress. By default, most network switches erase the marks sent to them unless configured to trust specific devices. For instance, makers placed in a packet by a server are often trusted, while markings set by an end host are not. Figure 8-3 illustrates a trust boundary.

In Figure 8-3, packets being transmitted by B are marked with AF41. As these packets are originating from a host within the QoS trust domain, the markings remain as they pass through D. Packets originating from A are marked with EF; however, since A is outside the QoS trust domain, this marking is stripped at D. Packets within the trust domain originating at A are seen as unmarked from a QoS perspective. The physical layer and upper layer protocol markings may, or may not, be related. For instance, the upper layer markings may be copied into the lower layer markings, or the lower layer markings may be carried through the network, or the lower layer markings may be stripped. There are many different possible implementations, so you should be careful to understand the way the markings are being handled across layers, as well as at each hop.

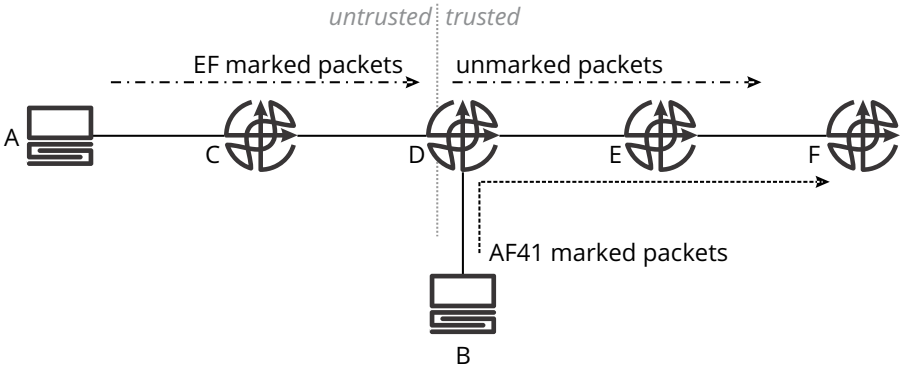


Figure 8-3 *QoS Trust Boundary*

Although network operators can use any values they choose in the ToS byte to create distinct traffic classes, it is often best to stick with some standard, such as the values defined by IETF RFC standards. These standards were defined to give network engineers a logical scheme to appropriately distinguish many different traffic classes.

Two of these "Per Hop Behavior" schemes appear in RFC2597, Assured Forwarding (AF), and RFC3246, Expedited Forwarding (EF), with various other RFCs updating or clarifying the content of these foundational documents. Both of these RFCs define traffic marking schemes, including the exact bit values that should populate the ToS byte or Traffic Class byte of an IP header to indicate a specific type of traffic. These are known as Differentiated Service Code Points, or DSCP values.

For example, RFC2597's assured forwarding scheme defines 12 values in a bitwise hierarchical scheme to populate the eight bits found in the ToS byte field. The first three bits are used to identify a class while the second three bits identify a drop precedence. The final two bits are unused. Table 8-1 illustrates the code markings for several AF classes.

Table 8-1 *Assured Forwarding Class of Service Quality of Service Markings*

	Class 1 (001)		Class 2 (010)		Class 3 (011)		Class 4 (100)	
Low Drop	001 010	AF11	010 010	AF21	011 010	AF31	100 010	AF41
Medium Drop	001 100	AF12	010 100	AF22	011 100	AF32	100 100	AF42
High Drop	001 110	AF13	010 110	AF23	011 110	AF33	100 110	AF43

Table 8-1 shows the DSCP bit value for AF11, traffic of Class 1 with a low drop precedence, is 001 010, where "001" indicates Class 1, and "010" indicates the drop precedence. Examining the table more deeply reveals the binary pattern selected by the RFC authors. All Class 1 traffic is marked with 001 in the first three bits, all Class 2 with 010 in the first three bits, etc. All Low Drop Precedence traffic is marked with 010 in the second three bits, all Medium Drop Precedence traffic with 100 in the second three bits, etc.

The Assured Forwarding scheme is shown in Table 8-2 to illustrate. It is not meant to be a definitive list of code points used in QoS traffic classification. For example, the Class Selector scheme described in RFC2474 exists for backward compatibility with the IP Precedence marking scheme. IP Precedence used only the first three bits of the ToS byte, for a total of eight possible classes. The Class Selector uses eight values as well, populating the first three bits of the six-bit DSCP field with significant values (matching the legacy IP Precedence scheme), and the last three bits with zeros. Table 8-2 shows these class selectors.

Table 8-2 *Class Selectors from RFC2474*

CS0	000 000
CS1	001 000
CS2	010 000
CS3	011 000
CS4	100 000
CS5	101 000
CS6	110 000
CS7	111 000

RFC3246 defines the latency, loss, and jitter requirements of traffic that must be forwarded expeditiously, along with a single new code point—EF, which is assigned binary value 101 110 (decimal 46).

The quantity and variety of formally defined DSCP values might seem overwhelming. The combined definitions of AF, CS, and EF alone result in formal definitions for 21 different classes out of a possible 64 using the six bits of the DSCP field. Are network engineers expected to use all of these values in their QoS prioritization schemes? Should traffic be broken down with such fine granularity for effective QoS?

In practice, most QoS schemes limit themselves to between four and eight traffic classes. The different classes allow for each group to be treated uniquely during times of congestion. For example, one traffic class might be shaped to meet a specific bandwidth threshold. Another traffic class might be prioritized above all other traffic. Yet another might be defined as business-critical, or traffic that is more important than most but less important than some. Network protocol traffic critical for infrastructure stability could be treated as very high priority. A scavenger traffic class might be near the bottom of the priority list, receiving slightly more attention than unmarked traffic.

A scheme incorporating these values is likely to be a mix of code points defined in the various RFCs and could vary somewhat from organization to organization. Generally accepted values include EF for critical traffic with a timeliness requirement such as VoIP, and CS6 for network control traffic such as routing and first hop redundancy protocols. Unmarked traffic (i.e., a DSCP value of 0) is delivered on a best-effort basis, with no guarantee of service level made (this would generally be considered the scavenger class, as above).

Preserving Classification

An interesting problem mentioned in both RFC2597 and RFC3246 is the issue of mark preservation when a marked packet is tunneled. When a packet is tunneled, the

original packet is wrapped—or encapsulated—inside of a new IP packet. The ToS byte value is inside the IP header of the now-encapsulated packet. Uh oh. What just happened to the carefully crafted traffic classification scheme? The answer is network devices engage in *ToS reflection* when tunneling. Figure 8-4 shows the reflection process.

When a packet is tunneled, the ToS byte value in the encapsulated packet is copied (or reflected) in the IP header of the tunnel packet. This preserves the traffic classification of the tunneled application.

A similar challenge comes when sending marked traffic from a network domain you control into one you do not. The most common example is sending marked traffic from your local area network into the network of your service provider, traversing its wide area network. Service providers, as a part of the contract to provide connectivity, often provide differentiated levels of service as well. However, for them to be able to provide differentiated service, traffic must be marked in a way they can recognize. Their marking scheme is unlikely to be the same as your marking scheme, considering the sheer number of possible marking schemes possible.

A couple of solutions to this dilemma present themselves:

- **DSCP mutation:** In this scenario, the network device on the border between the LAN and the WAN translates the mark from the original value assigned on the LAN into a new value the SP will honor. The translation is performed in accordance with a table configured by a network engineer.
- **DSCP translation:** It is not uncommon for SPs to observe only the first three bits of the ToS byte, hearkening back to the days of IP Precedence defined all the way back in RFC791.

In the second solution, the network engineer is faced with creating a modern DSCP marking scheme using six bits, even though the SP will pay attention to just

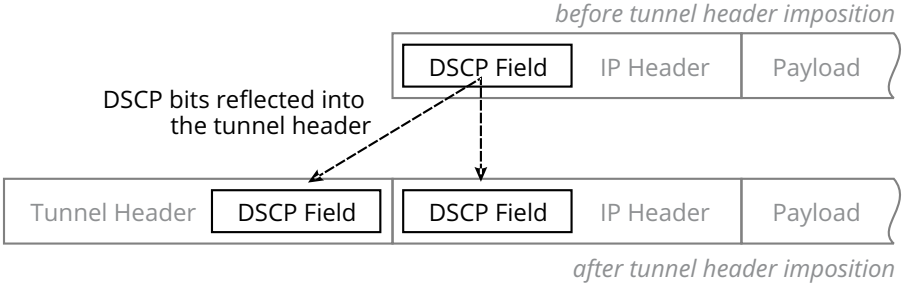


Figure 8-4 *DSCP Bit Reflection between the Inner and Outer Header*

the first three. The challenge is to maintain differentiation. For example, consider the scheme illustrated in Table 8-3; this scheme will not resolve the issue.

Table 8-3 *Translating DSCP to IP Precedence*

DSCP (LAN, 6 bits)	PRECEDENCE (SP WAN, 3 bits)
EF / 101 110	101
CS5 / 101 000	101
AF23 / 010 110	010
AF13 / 001 110	001
AF12 / 001 100	001
AF11 / 001 010	001

In this table, six unique DSCP values have been defined for use on the local area network. However, these six unique values are reduced to only three unique values if only the first three bits are honored by the service provider. This means some traffic that might have enjoyed differentiated treatment before entering the provider's network will now be lumped into the same bucket. In the example, EF and CS5, formerly unique, fall into the same class when they leave the border router, as the initial three bits of EF and CS5 are both 101. The same goes for AF11, AF12, and AF13—three formerly distinct traffic classes that will now be treated identically while traversing the SP WAN, as they all share the same initial 001 value in the initial three bits.

A way to solve this problem is to create a DSCP marking scheme that will maintain uniqueness in the first three bits as demonstrated in Table 8-4. This might require a reduction in the overall number of traffic classes, however. Limiting the scheme to the first three bits to define classes will reduce the total number of classes to maximum of six.

Table 8-4 *Translating DSCP to IP Precedence*

DSCP (LAN, 6 bits)	PRECEDENCE (SP WAN, 3 bits)
EF / 101 110	101
AF41 / 100 010	100
AF31 / 011 010	011
AF21 / 010 010	010
AF11 / 001 010	001
CS0 / 000 000	000

Table 8-4 shows a marking scheme using a mix of EF, AF, and Class Selector values especially chosen to preserve uniqueness in the first three bits.

The Unmarked Internet

So far, this discussion assumes network devices will honor the marks found in an IP packet. Certainly, this is true in privately owned networks and on leased networks where the terms of trust have been negotiated with a service provider. But what happens on the global Internet? Do network devices servicing public Internet traffic observe and honor DSCP values, and prioritize some traffic over other traffic during times of congestion? From the perspective of Internet consumers, the answer is no. The public Internet is a best effort transport. There are no guarantees of even traffic delivery, let alone traffic prioritization.

Even so, the global Internet is increasingly being used as a wide area transport for traffic carried between private facilities. Cheap broadband Internet service sometimes offers more bandwidth at a lower cost than private WAN circuits leased from a service provider. The tradeoff for this lower cost is a lower level of service, often substantially lower. Cheap Internet circuits are cheap because they do not offer service level guarantees, at least not ones meaningful enough to inspire confidence in the timely delivery of traffic (if at all). While it is possible to mark traffic destined for the Internet, the ISP will not pay attention to the marks. When the Internet is being used as a WAN transport, how then can a QoS policy be effectively applied to traffic?

Creating a Quality of Service over the public Internet requires a rethinking of QoS prioritization schemes. To the private network operator, the public Internet is a black box. The private operator has no control over the public routers between the edges of the private WAN. It is not possible for the private operator to prioritize certain traffic over other traffic on a congested public Internet link without control over the intermediate, public Internet router.

The solution to providing Quality of Service over the public Internet is multipartite:

- Control over traffic happens at the private network edge, before the traffic enters the public Internet's black box. This is the last point at which the private network operator has device control.
- QoS policy is enforced primarily through *path selection* and secondarily via congestion management.

Note

See Chapter 17, "Policy in the Control Plane," for more information on using traffic engineering to manage QoS problems.

Implicit in the notion of path selection is the existence of more than one path to select from. In the emerging Software-Defined Wide Area Network (SD-WAN) model, two or more WAN circuits are treated as a bandwidth pool. In the pool, the individual circuit used to carry traffic at any given time is decided on a moment-by-moment basis as the network devices at the edge of the pool perform quality tests along each available circuit or path. Depending on a path's characteristics at any point in time, traffic may be sent down one path or another.

Which traffic is sent down which path? SD-WAN offers granular traffic classification capabilities beyond the human-manageable four to eight classes defined by DSCP marks imposed on the ToS byte. SD-WAN path selection policy can be defined on an application-by-application basis, with nuanced forwarding decisions made. This is distinct from the idea of marking as close to the source as possible, and then making forwarding decisions during congestion times based on the mark. Rather, SD-WAN compares real-time path characteristics with the policy-defined needs of applications classified in real time, and then makes a real-time path selection decision.

The result should be an application user experience similar to a wholly owned private WAN with a QoS prioritization scheme managing congestion. The mechanisms used to achieve this similar result are substantially different, however. The functionality of SD-WAN hinges on the ability to detect and quickly reroute traffic flows around a problem, as opposed to managing a congestion problem once it has happened. SD-WAN technologies do not replace QoS; rather they provide an "over the top" option for situations where QoS is not supported on the underlying network.

Congestion Management

Classification, by itself, does not result in a specific forwarding posture on the part of a network device. Rather, classifying traffic is the first necessary step in creating a framework for differentiated forwarding behavior. In other words, the packets have been classified and differentiated, but that is all. Pointing out differences is not the same as taking differentiated actions on those classes.

Our discussion of QoS now moves into the realm of *policy*. How are congested interfaces managed? When packets are waiting for delivery, how does a network device decide which packets are sent first? The decision points are based primarily around how well the user experience can tolerate packet jitter, latency, and loss. A variety of problems and QoS tools present themselves to address these issues.

Timeliness: Low-Latency Queueing

Network interfaces forward packets as quickly as possible. When traffic is flowing at less than or equal to the bandwidth of the egress interface, traffic is delivered, one packet at a time, without drama. When an interface can keep up with the demands being placed on it, there is no *congestion*. Without congestion, there is no concern about differentiated traffic types. The marks on the individual packets might be observed for statistical purposes, but there is no QoS policy that needs to be applied. Traffic arrives at the egress interface and is delivered.

As described in the description of the switching path through a router in Chapter 7, "Packet Switching," packets are delivered to a transmit ring after being switched. The outbound interface's physical processor removes packets from this ring and clocked onto the physical network medium. What happens if there are more packets to be transmitted than the link can support? In this case, the packets are placed in a queue, the *output queue,* rather than on the transmit ring. The QoS policies configured on the router are actually implemented in the process of removing packets from the output queue onto the transmit ring for transmission. When packets are being placed on the output queue, rather than the transmit ring, the interface is said to be congested.

By default, congested network interfaces deliver packets on a first-in, first-out (FIFO) basis. FIFO does not make a policy decision based on differentiated traffic classes; rather FIFO simply services buffered packets in order, as quickly as the egress interface will allow. For many applications, FIFO is not a bad way to go about dequeueing packets. For instance, there might be little real-world impact if a Hypertext Transfer Protocol (HTTP, the protocol used to carry World Wide Web information) packet from one web server is transmitted before one from a different web server.

For other traffic classes, there is a great deal of concern about *timeliness*. As opposed to FIFO, some packets should be moved to the head of the queue and sent as quickly as possible to avoid *delay* and an impact to the end user experience. One impact is in the form of a packet arriving too late to be useful. Another impact is in the form of a packet not arriving at all. It is worth considering each of these scenarios and then some helpful QoS tools for each.

Voice over IP (VoIP) traffic must be both delivered and delivered on time. When considering voice traffic, think of any real-time voice chatting performed over the Internet using an application such as Skype. Most of the time, the call quality is decent. You can hear the other person. That person can hear you. The conversation flows normally. You might as well be in the same room with the other person, even if he is across the country.

On occasion, VoIP call quality might drop. You might hear a series of subsecond stutters in the person's voice, where the speed of vocal delivery is irregular. In this case, you are experiencing *jitter*, which means packets are not arriving consistently in time. Overly long interpacket gaps result in an audible stuttering effect. While no packets were lost, they weren't delivered along the network path in a timely fashion. Somewhere along the path, the packets were delayed long enough to introduce audible artifacts. Figure 8-5 illustrates jitter in packet transmission.

VoIP call quality can also suffer from packet loss, where packets in the network path were dropped along the way. While there are many potential reasons for packet loss in network paths, the scenario considered here is tail drop, where so much traffic has arrived beyond the egress interface's capability to keep up that there is no room left in the buffer to queue up additional excess. The latest traffic arrivals are discarded as a result; this drop is called tail drop.

When VoIP traffic is being tail dropped, the listener hears the result of the loss. There are gaps where the speaker's voice is completely missing. Dropped packets could come through as silence, as the last bit of received sound being looped as a way to fill the gap, an extended hiss, or other digital noise. Figure 8-6 illustrates dropped packets across a router or switch.

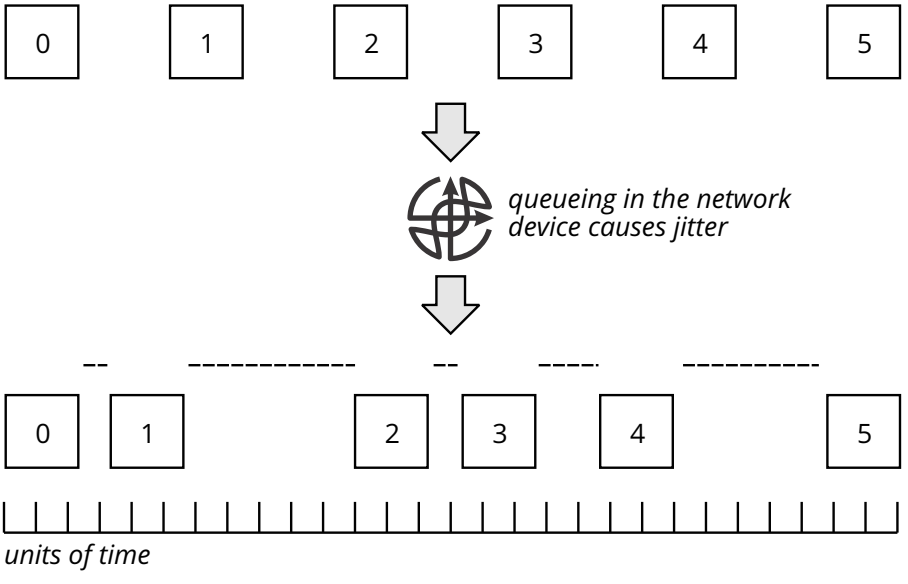


Figure 8-5 *Jitter in Packet Delivery in a Network*

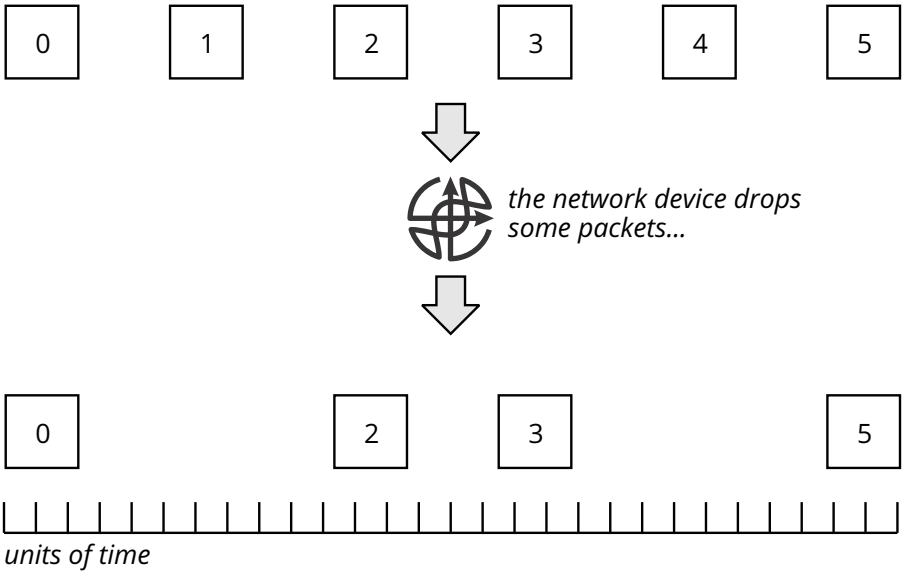


Figure 8-6 *Dropped Packets across a Router or Switch*

To deliver consistent call quality, even in the face of a congested network path, a QoS prioritization scheme must be applied. This scheme must meet the following criteria.

- **VoIP traffic must be delivered:** Lost VoIP packets result in an audible drop in the conversation.
- **VoIP traffic must be delivered on time:** Delayed or jittery VoIP packets result in audible stutters.
- **VoIP traffic must not starve other traffic classes of bandwidth:** As important as VoIP is, well-written QoS policies will balance timely delivery of voice packets with the need for other traffic classes to also use the link.

A common scheme deployed to prioritize traffic sensitive to loss and jitter is low-latency queueing (LLQ). No IETF RFC defines LLQ; rather, network equipment vendors invented LLQ as a tool in the QoS policy toolbox to prioritize traffic requiring low delay, jitter, and loss—such as voice.

There are two key elements to LLQ.

- Traffic serviced by LLQ is transmitted as quickly as possible to avoid delay, minimizing jitter.

- Traffic serviced by LLQ is not allowed to exceed a specified amount of bandwidth (generally recommended to be no more than 30% of the available bandwidth). Traffic exceeding the bandwidth limit is dropped rather than transmitted. This technique avoids starving other traffic classes.

Implied in this scheme is a compromise for traffic classes services by the LLQ. The traffic will be serviced as quickly as possible, effectively moving it to the head of the queue as soon as it shows up at a congested interface. The catch is that there is a limit on just how much traffic in this class will be treated in this way. That limit is imposed by a network engineer composing the QoS policy.

By way of illustration, assume a WAN link with 1,024Kbps of available bandwidth. This link connects the headquarters office to the service provider WAN cloud, which also connects several remote offices back to HQ. This is a busy WAN link, carrying VoIP traffic between offices, as well as web application traffic and backup traffic from time to time. Furthermore, assume the VoIP system is encoding voice traffic with a codec requiring 64Kbps per conversation.

In theory, this 1,024Kbps link could accommodate 16 × 64Kbps simultaneous VoIP conversations. However, this would leave no room for the other traffic types that are present. This is a busy WAN link! In the writing of the QoS policy, a decision must be made. Just how many voice conversations will be allowed by the LLQ to avoid starving the remaining traffic of bandwidth? A choice could be made to limit the LLQ to only 512Kbps of bandwidth, which would be adequate to handle eight simultaneous conversations, leaving the rest of the WAN link for other traffic classes.

Assuming the link is congested, the situation the link must be in for the QoS policy to be effective, what would happen to the ninth VoIP conversation? This question is actually a naive one, because it assumes each conversation is being handled separately by the QoS policy. In fact, the QoS policy treats all traffic being serviced by the LLQ as one large group of packets. Once the ninth VoIP conversation joins, there will be 576Kbps' worth of traffic to be serviced by an LLQ that only has 512Kbps allocated. To find the amount of dropped traffic, subtract the total traffic set aside for the LLQ from the total traffic offered: 576Kbps – 512Kbps = 64Kbps' worth of LLQ traffic will be dropped to conform to the bandwidth cap. The dropped 64Kbps will come from the LLQ traffic class as a whole, impacting all of the VoIP conversations. If a tenth, eleventh, and twelfth VoIP conversation were to join the LLQ, the problem would become more severe. In this case, 64Kbps × 4 = 256Kbps' worth of nonconforming traffic that would discarded from the LLQ, causing even more loss from all of the VoIP conversations.

As this example shows, managing congestion requires knowledge of the application mix, peak load times, bandwidth demands, and network architecture options available. Only when all points are considered can a solution meeting business

objectives be put in place. For instance, assume 1,024Kbps is the largest you can make the long-haul link due to cost constraints. You could raise the LLQ bandwidth limitation to 768Kbps to accommodate 12 conversations at 64Kbps each. However, this would leave only 256Kbps for other traffic, which perhaps is not enough to meet your business needs for other applications.

In this case, it might be possible to coordinate with the voice system administrator to use a voice codec requiring less bandwidth. If a new codec requiring only 16Kbps of bandwidth per call is deployed instead of the original 64Kbps, 32 VoIP conversations could be forwarded without loss through an LLQ allocated 512Kbps of bandwidth. The compromise? Voice quality. The human voice encoded at 64Kbps will sound more clear and natural when compared to one encoded at 16Kbps. It may also be better to encode at 16Kbps so fewer packets are dropped, and hence the overall quality is better. Which solution to apply will depend on the specific situation.

It is possible for more traffic than specified by the LLQ bandwidth cap to pass through the interface. If the bandwidth cap for traffic serviced by the LLQ is set at a maximum of 512Kbps, it is possible for more than 512Kbps' worth of traffic in the class to pass through the interface. This programmed behavior exhibits itself only if the interface is uncongested. In the original example, where a 64Kbps codec is being used, transmitting 10 conversations at 64Kbps over the link will result in 640Kbps' worth of voice traffic traversing the 1,024Kbps capacity link (1,024Kbps – 640Kbps = 384Kbps left). As long as all other traffic classes stay below 384Kbps total bandwidth utilization, then the link will remain congestion-free. If the link is not congested, then QoS policies do not take effect. If the QoS policy is not in effect, then the LLQ bandwidth cap of 512Kbps does not impact the 640Kbps of aggregated voice traffic.

In this discussion of LLQ, the context has been that of voice traffic, but be aware that LLQ can be applied to any sort of traffic desired. However, in networks where VoIP is present, VoIP tends to be the only traffic serviced by LLQ. For networks where VoIP traffic is not present, LLQ becomes an interesting tool to guarantee timely, low delay and jitter delivery of other sorts of application traffic. However, LLQ is not the only tool available to the QoS policy writer. Several other tools are also useful.

Fairness: Class-Based Weighted Fair Queueing

When timing is of less concern than actual delivery, traffic can often be managed by the technique of class-based weighted fair queueing (CBWFQ). In CBWFQ, participating traffic classes are serviced in accordance with the policy assigned to them. For example, traffic marked as AF41 might be guaranteed a minimum amount of bandwidth. Traffic marked as AF21 might also be guaranteed a minimum amount of bandwidth, perhaps less than the amount given to AF41 traffic. Unmarked traffic might get whatever bandwidth is left over.

CBWFQ has the notion of fairness, where various traffic classes have a chance to be delivered across the congested link. CBWFQ ensures the packets in the queue are being serviced in a fair manner, in accordance with the QoS policy. All traffic classes with bandwidth assigned to them will have packets sent along.

For example, assume a link of 1,024Kbps in capacity. Traffic class AF41 has been guaranteed a minimum of 256Kbps. Class AF31 has been guaranteed a minimum of 128Kbps. Class AF21 has been guaranteed a minimum of 128Kbps. This gives us a ratio of 2:1:1 among those three classes. The remaining 512Kbps is unallocated, meaning it is available for use by other traffic. Including the unallocated amount, the full ratio is 256:128:128:512, which reduces to 2:1:1:4.

To decide which packet is sent next, the queue is serviced in accordance with the CBWFQ policy. This example carves up the 1,024Kbps of bandwidth into four portions, with a ratio of 2:1:1:4. For simplicity's sake, assume the congested interface will service the packets in the queue in eight clock cycles:

1. **Clock cycle 1.** An AF41 packet will be sent.
2. **Clock cycle 2.** Another AF41 packet will be sent.
3. **Clock cycle 3.** An AF31 packet will be sent.
4. **Clock cycle 4.** An AF21 packet will be sent.
5. **Clock cycles 5–8.** Packets with other classifications as well as unclassified packets will be sent.

This example assumes there are packets representing each of the four classes sitting in the buffer, queued to be sent. However, the situation is not always so straightforward. What happens when there are no packets from a particular traffic class to be sent, even though there is room in the guaranteed minimum bandwidth allocation?

Guaranteed bandwidth minimums are not reservations. If the traffic class assigned the guaranteed minimum does not require the full allocation, other traffic classes could use the bandwidth. Neither are guaranteed bandwidth minimums hard limits. If the amount of traffic for a specific class exceeds the guaranteed minimum and bandwidth is available, traffic for the class will flow at a faster rate.

Thus, what happens could look more like this:

1. **Clock cycle 1.** An AF41 packet is sent.
2. **Clock cycle 2.** There is no AF41 packet to be sent, so an AF31 packet is sent instead.
3. **Clock cycle 3.** Another AF31 packet is sent.
4. **Clock cycle 4.** There is no AF21 packet to be sent, so an unclassified packet is sent.

5. **Clock cycles 5–7.** Packets with other classifications as well as unclassified packets are sent.
6. **Clock cycle 8.** There are no more otherwise classified or unclassified packets to be sent, so yet another AF31 packet is sent.

As a result, unused bandwidth is divided up among the classes with excess traffic.

Overcongestion

CBWFQ does not increase throughput of a congested link. Rather, the algorithm is about carefully controlled *sharing* of the overstressed link in a way reflecting the relative importance of various traffic classes. The result of CBWFQ sharing is traffic being delivered via the congested link, but at a reduced rate when compared to the same link at an uncongested time.

The distinction between "sharing an overstressed link" and "creating bandwidth from nothing" cannot be overstated. A common misconception about QoS is, despite points of congestion in a network path, user experience will remain identical. This just is not the case. QoS tools like CBWFQ are, for the most part, about making the best of a bad situation. In picking which traffic is forwarded when, QoS is also choosing which traffic to drop; there are "winners" and "losers" among the flows transmitted across the network.

LLQ is a notable exception because traffic serviced by an LLQ is assumed to be so absolutely critical that it will be serviced to the exclusion of other traffic, up to the bandwidth limitation assigned. LLQ seeks to preserve user experience.

Other QoS Congestion Management Tools

Traffic shaping is a way to gracefully cap traffic classes to a specific rate. For example, traffic marked as AF21 might be shaped to 512Kbps. Shaping is graceful; it allows for nominal bursts above the defined limit before dropping packets. This allows TCP to adjust more easily to the required rate. When the throughput of a shaped traffic class is graphed, the result shows a ramp-up to the speed limit, and then a flat, consistent transfer speed for the duration of the flow. Traffic shaping is most often applied to traffic classes populated by *elephant flows*.

Elephant flows are long-lived traffic flows used to move large amounts of data between two endpoints as quickly as possible. Elephant flows have the ability to fill network bottlenecks with their own traffic, squashing smaller flows. A common QoS strategy is to shape the traffic rate of elephant flows so it will leave the bottleneck link with enough bandwidth to effectively service other traffic classes.

Policing is similar to traffic shaping but treats excess (nonconforming) traffic more harshly. Rather than allowing a small burst above the defined bandwidth cap like shaping does before dropping, policing drops excess traffic immediately. When facing a policer, impacted traffic ramps up to the bandwidth limit, exceeds, and is dropped. This drop behavior causes TCP to start the ramp-up process over again. The resulting graph looks like a sawtooth. Policing can be used to accomplish other tasks, such as re-marking nonconforming traffic to a lower priority DSCP value, rather than dropping.

Queue Management

Buffering packets to deal with a congested interface seems like a lovely idea. Indeed, buffers are necessary to handle traffic arriving too fast due to bursts or interface speed mismatches—moving from a high-speed LAN to lower-speed WAN, for instance. Thus far, this discussion of QoS has been focused on classifying, prioritizing, and then forwarding packets queued in those buffers in accordance with a policy. Sizing buffers as large as possible might seem like a good thing. In theory, if a buffer is large enough to queue up packets overwhelming a link, all packets will eventually be delivered. However, both large buffers and full buffers introduce problems to be dealt with.

When packets are in a buffer, they are being delayed. Some number of microseconds or even milliseconds are being added to the packet's journey between source and destination while they sit in a buffer waiting to be delivered. Delayed travel is troublesome for some network conversations, as the algorithms employed by TCP assume a predictable, and ideally low, amount of delay between sender and receiver.

Under the category of active queue management, you will find different methods for managing the contents of the queue. Some methods go after the problem of a full queue, dropping enough packets to leave a little room for new arrivals. Other methods go after the challenge of delay, maintaining shallow queue depths, minimizing the amount of time a packet spends in a buffer. This keeps buffered delay reasonable, allowing TCP to adjust traffic speed to a rate appropriate for the congested interface.

Managing a Full Buffer: Weighted Random Early Detection

Random early detection (RED) helps us deal with the problem of a full queue. Buffers are not infinite in size; there is only so much memory allocated to each one. When the buffer is filled with packets, then the new arrivals are tail dropped. This does not bode well for critical traffic like VoIP, which cannot be dropped without impacting the user experience. The way to handle this problem is to ensure the buffer is never

entirely full. If the buffer is never completely full, then there is always room to accept additional traffic.

To prevent a full buffer, RED uses a scheme of proactively dropping selected inbound traffic, keeping spaces open. The more full the buffer gets, the more likely an incoming packet is to be dropped. RED is the predecessor to modern variations such as Weighted Random Early Detection (WRED). WRED takes into consideration the priority of the incoming traffic based on its mark. Higher priority traffic is less likely to be dropped. Lower priority traffic is more likely to be dropped. If the traffic is using some form of windowed transport, such as TCP, these drops will be interpreted as congestion, signaling the transmitter to slow down.

RED and variations also manage the problem of *TCP synchronization*. Without RED, all inbound packets are tail dropped in the presence of a full buffer. For TCP traffic, the packet loss resulting from the tail drop causes transmission speed to throttle back and the lost packets to be retransmitted. Once packets are being delivered again, TCP will attempt to ramp back up to a faster rate. If this cycle happens across many different conversations at the same time, as happens in a RED-free tail-drop scenario, the interface can experience bandwidth utilization oscillations where the link goes from congested (and tail dropping) to uncongested and underutilized as *all* of the throttled-back TCP conversations start to speed back up. When the now-synchronized TCP conversations are talking quickly enough again, the link is again congested, and the cycle repeats.

RED addresses the TCP synchronization issue by leveraging randomness when selecting which packets to drop. Not all TCP conversations will have packets dropped. Only certain conversations will, randomly selected by RED. The TCP conversations flowing through the congested link never end up synchronized, and the oscillation is avoided. Link utilization is more steady.

Managing Buffer Delay, Bufferbloat, and CoDel

An obvious question might arise at this point. If packet loss is a bad thing, why not make the buffers big enough to handle congestion? If the buffers are bigger, more packets can be queued up, and maybe you can avoid this pesky problem of packet loss. In fact, this strategy of sizable buffers has found its way into various network devices and some network engineering schemes. However, when link congestion causes buffers to fill and stay filled, the large buffer is said to be *bloated*. This phenomenon is so well known in the networking industry, it has a name: bufferbloat.

Bufferbloat has a negative connotation because it is an example of too much of a good thing. Buffers are good. Buffers provide a bit of leeway to give a burst of packets somewhere to stay while an egress interface catches up. To handle small bursts of traffic, buffers are necessary, with the critical tradeoff of introducing delay; however,

oversizing buffers does not make up for undersizing a link. A link has a specific amount of carrying capacity. If the link is chronically asked to transmit more data than it is able to carry, then it is ill suited to the task required of it. No amount of buffering can overcome a fundamental network capacity issue.

Increasing the depth of a buffer ever larger does not improve link throughput. In fact, a constantly filled buffer puts a congested interface under an even greater strain. Consider a couple of examples, contrasting Unacknowledged Datagram Protocol (UDP) and Transmission Control Protocol (TCP).

1. In the case of VoIP traffic, buffered packets arrive late. Dead air is enormously disruptive to a real-time voice conversation. VoIP is an example of traffic transported via UDP over IP. UDP traffic is unacknowledged. The sender sends the UDP packets along with no concern about whether they make it to their destination. There is no retransmission of packets if the destination host does not receive a UDP packet. In the case of VoIP, the packet arrives on time, or it does not. If it does not, then there is no point in retransmitting it, because it is far too late to matter. The humans doing the talking have moved on.

   LLQ might come to your mind as the answer to this problem, but part of the issue is the oversized buffer. A large buffer will take time to service causing delay in the VoIP traffic delivery, even if LLQ is servicing the VoIP traffic. It would be better to drop VoIP traffic sitting in the queue too long than send it too late.

2. In the case of most application traffic, the traffic is transported via TCP over IP, rather than UDP. TCP is acknowledged. A TCP traffic sender waits for the receiver to acknowledge receipt before more traffic is sent. In a bufferbloat situation, a packet sits in the full, oversized buffer of a congested interface for an overly long time, delaying the delivery of the packet to the receiver. The receiver gets the packet and sends an acknowledgment. The acknowledgment was slow in arriving at the sender, but it did arrive. TCP does not care how long it takes for the packet to arrive, so long as it gets there. And thus, the sender keeps sending traffic at the same speed through the congested interface, which keeps the oversized buffer full and the delay times long.

   In extreme cases, the sender might even retransmit the packet, while the original packet is still sitting in the buffer. The congested interface finally sends the original buffered packet to the receiver, with a second copy of the same packet now in flight, putting even more strain on an already congested interface!

These scenarios illustrate inappropriately sized buffers are, in fact, not good. A buffer must be appropriately sized both for the speed of the interface it services and the nature of the application traffic likely to pass through it.

One attempt on the part of the networking industry to cope with the oversized buffers found along certain network paths is *controlled delay*, or CoDel. CoDel assumes an oversized buffer but manages packet delay by monitoring how long a packet has been in the queue. This is known as the sojourn time. When the packet sojourn time has exceeded the computed ideal, the packet is dropped. This means packets at the head of the line—those that have waited the longest—are going to be dropped before packets currently at the tail end of the queue.

CoDel's aggressive stance toward dropping packets allows TCP flow control mechanisms to work as intended. Rather than packets suffering from high delay while still being delivered, they are dropped before the delay gets too long. The drop forces a TCP sender to retransmit the packet and slow down the transmission, a strongly desirable result for a congested interface. The aggregate result is a more even distribution of bandwidth to traffic flows contending for the interface.

In early implementations, CoDel has been shipping in consumer-edge devices parameterless. Certain defaults about the Internet are assumed. Assumptions include a 100ms or less roundtrip time between senders and receivers, and a 5ms delay is the maximum allowed for a buffered packet. This parameterless configuration makes it easier for vendors of consumer-grade network gear to include. Consumer networks are an important target for CoDel, as the mismatch of high-speed home networks and lower-speed broadband networks causes a natural congestion point. In addition, consumer-grade network gear often suffers from oversized buffers.

Final Thoughts on Quality of Service

Quality of Service is a deep topic; a lot of research has been done in understanding how flows react to specific network conditions, and how network devices should handle queueing and packet processing to ensure the minimal amount of traffic is dropped, and delay and jitter are minimized, under even the worst of network conditions. There are several broad areas of QoS you need to understand in order to be an effective network engineer, including packet classification, packet marking, translation of packet marking across different networks, and queue processing. Each of these interacts with the transport protocols in ways that are not always obvious.

The next chapter will dive into a topic from a completely different realm of network engineering—virtualization. Working through the problem set considered in the early chapters of this book, virtualization plays a role in multiplexing multiple virtual topologies across a single physical topology.

Further Reading

Baker, Fred, David L. Black, Dr. Kathleen M. Nichols, and Steven L. Blake. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. Request for Comments 2474. RFC Editor, 1998. doi:10.17487/RFC2474.

Baker, Fred, and Gorry Fairhurst. *IETF Recommendations Regarding Active Queue Management*. Request for Comments 7567. RFC Editor, 2015. doi:10.17487/RFC7567.

Bennett, Jon, Shahram Davari, Dimitrios Stiliadis, William Courtney, Kent Benson, Jean-Yves Le Boudec, Victor Firoiu, Dr. Bruce S. Davie, and Anna Charny. *An Expedited Forwarding PHB (Per-Hop Behavior)*. Request for Comments 3246. RFC Editor, 2002. doi:10.17487/RFC3246.

Bollapragada, Vijay, Russ White, and Curtis Murphy. *Inside Cisco IOS Software Architecture*. Indianapolis, IN: Cisco Press, 2000.

Floyd, S., and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance." *IEEE/ACM Transactions on Networking* 1, no. 4 (August 1993): 397–413. doi:10.1109/90.251892.

Gettys, Jim, and Kathleen Nichols. "Bufferbloat: Dark Buffers in the Internet." *ACM Queue*, November 2011. http://queue.acm.org/detail.cfm?id=2071893.

"History Of Networking—Fred Baker—QoS & DS Bit." *Network Collective*, August 2, 2017. http://thenetworkcollective.com/2017/08/hon-fred-baker-qos/.

Nichols, Kathleen. "Controlling Queue Delay." *ACM Queue*, May 2012. http://queue.acm.org/detail.cfm?id=2209336.

Postel, J., ed. *Internet Protocol*. Request for Comments 791. RFC Editor, 1981. doi:10.17487/rfc791.

"RED in a Different Light." *Jg's Ramblings*, December 17, 2010. https://gettys.wordpress.com/2010/12/17/red-in-a-different-light/.

Srikant, Rayadurgam. *The Mathematics of Internet Congestion Control*. 2004 edition. Boston, MA: Birkhäuser, 2003.

Stringfield, Nakia, Russ White, and Stacia McKee. *Cisco Express Forwarding*. 1st edition. Indianapolis, IN: Cisco Press, 2007.

Weiss, Walter, Dr. Juha Heinanen, Fred Baker, and John T. Wroclawski. *Assured Forwarding PHB Group*. Request for Comments 2597. RFC Editor, 1999. doi:10.17487/rfc2597.

Review Questions

1. QoS is sometimes deployed to counter the impact of running a File Transfer Protocol, such as FTP or a backup program, and a real-time streaming application, such as voice over IP, over the same link. Why do these two kinds of application interact poorly in a single queue? A hint: packet sizes matter.
2. The chapter notes that TCP sends traffic until it encounters congestion and then backs off. What mechanism in TCP causes this effect? What happens if a large number of TCP sessions with packets in a single queue all have a single packet dropped at the same time?
3. How does WRED try to mitigate the effect of dropping packets across a set of TCP flows at the same time?
4. Trace the way in which the ToS bits in an IPv6 header are translated into an MPLS header and then from an MPLS header to an Ethernet header. In what places is information lost in these translations?
5. Some vendors have recommended the same DSCP values be used in different parts of the network to express different classes or types of service. Would you agree with this recommendation? What complexities does it add, and where does it make things simpler?
6. What kinds of traffic might you place into a high-priority class, and why? What kinds in a scavenger class, and why?
7. According to the State/Optimization/Surface three-way tradeoff, adding state should increase optimization while also increasing complexity, etc. Consider the case of adding more classes of service in a network. Describe the tradeoffs between additional state, increased optimization, and where the interaction surfaces between the different layers of protocols in the network might be impacted.
8. Traffic engineering is a completely different way to implement Quality of Service in a network. Can you use traffic engineering to resolve all Quality of Service problems in all networks? Describe a network engineering situation or topology in which it seems like traffic engineering would be able to solve most QoS requirements and one where it would not.
9. What percentage of traffic is generally recommended to be placed in the low-latency queue in an LLQ system? Explain why.
10. How does SD-WAN take the complexity of managing QoS Class and Type of Service "out of the hands of humans"? What are the advantages and disadvantages of such an approach?

Chapter 9

Network Virtualization

Learning Objectives

After reading this chapter, you should understand:

- What problems network virtualization is used to solve
- How a tunneled packet is switched through a network
- What two problems every network virtualization solution must solve
- The general concepts of a tunnel, overlay, underlay, and over-the-top service
- The concept of the inner and outer header, and how each is used in a virtual topology
- The basic operation of segment routing
- The basic concept of Software-Defined Wide Area Networks
- A basic understanding of at least some tradeoffs in building and operating virtual topologies
- The concepts of shared fate and shared link risk groups

Network virtualization is, in the simplest terms possible, the creation of logical topologies built on top of a physical topology. These logical topologies are often called virtual topologies—hence the concept of network virtualization. These topologies may consist of a single virtual link across a larger network, called a tunnel, or a collection of virtual links that appear to be a complete network on top of the physical network, called an overlay.

This chapter will begin with a discussion about why virtual topologies are created and used, illustrated by two use cases. The second section of this chapter will consider the problems any virtualization solution must solve, and the third section will consider complexity and network virtualization. Following this, two examples of virtualization technologies will be considered: segment routing (SR) and Software-Defined Wide Area Networks (SD-WAN).

Understanding Virtual Networks

Virtualization adds complexity in protocol design, network design, and troubleshooting, so why virtualize? The reasons tend to reduce to separating multiple traffic flows across a single physical network. This might sound suspiciously like another form of multiplexing because it *is* another form of multiplexing. The primary differences between the forms of multiplexing considered to this point and virtualization are

- Allowing multiple control planes to operate with different sets of reachability information across a single physical topology
- Allowing multiple sets of reachable destinations to operate across a single physical topology without interacting with one another

The multiplexing techniques considered to this point have focused on allowing multiple devices to use a single physical network (or set of wires), allowing every device to talk to every other device (so long as they know about one another from a reachability perspective). Virtualization focuses on breaking up the single physical network into multiple reachability domains, where every device within a reachability domain can communicate with every other device within the same reachability domain, but devices cannot communicate across reachability domains (unless there is some connection point between the reachability domains).

Figure 9-1 illustrates a network with a virtual topology laid on top of the physical topology.

In Figure 9-1, a virtual topology has been created on top of the physical network, with the virtual link [C,H] created to carry traffic across the network. In order to create the virtual topology, C and H must have some sort of local forwarding information separating the physical topology from the virtual topology, which would normally pass through either E or D. This would normally take the form of either a special set of virtual interface entries in the local routing table, or a Virtual Routing and Forwarding (VRF) table containing only information about the virtual topology.

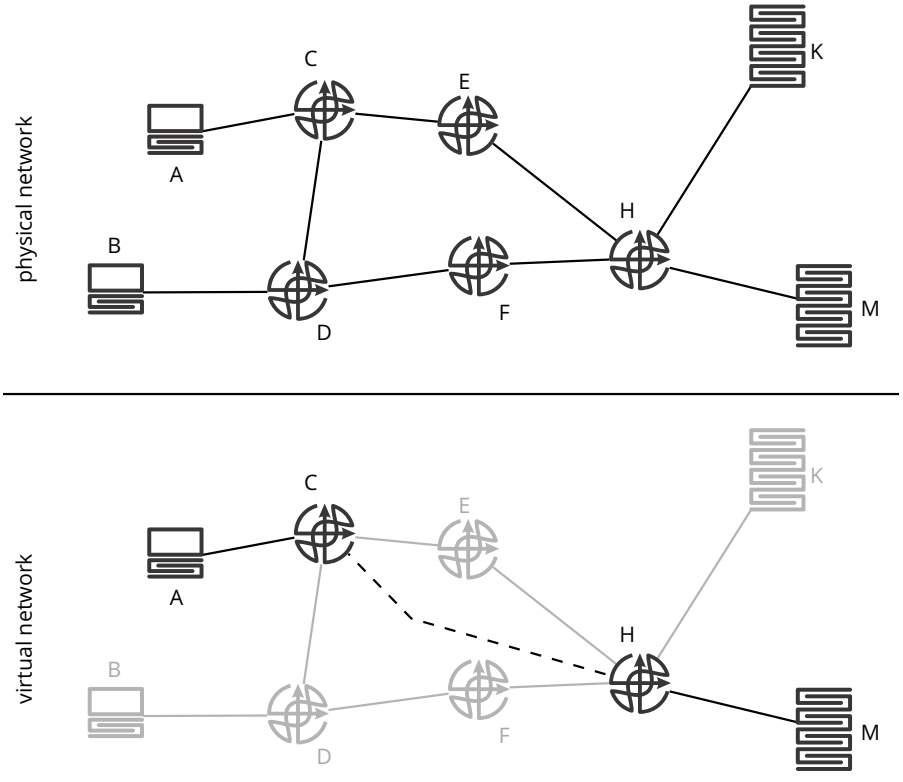


Figure 9-1 *A Physical and a Virtual Topology*

Considering the packet flow through the virtual topology can be helpful in understanding the concepts. What would the packet flow look like if C and H had virtual interfaces? Figure 9-2 illustrates.

In Figure 9-2, the forwarding process follows these steps:

1. A transmits a packet toward M.
2. C receives this packet, and, examining its local routing table, finds the shortest path to the destination is through a virtual interface toward H. This virtual interface is normally called a tunnel interface; it appears, from the routing table's perspective, like any other interface on the router.
3. The virtual interface through which the packet needs to be transmitted has rewrite instructions that include adding a new header, the tunnel header, or outer header, onto the packet, and forwarding the resulting packet. The original packet header is now called the inner header. C adds the outer header and processes the new packet for forwarding.

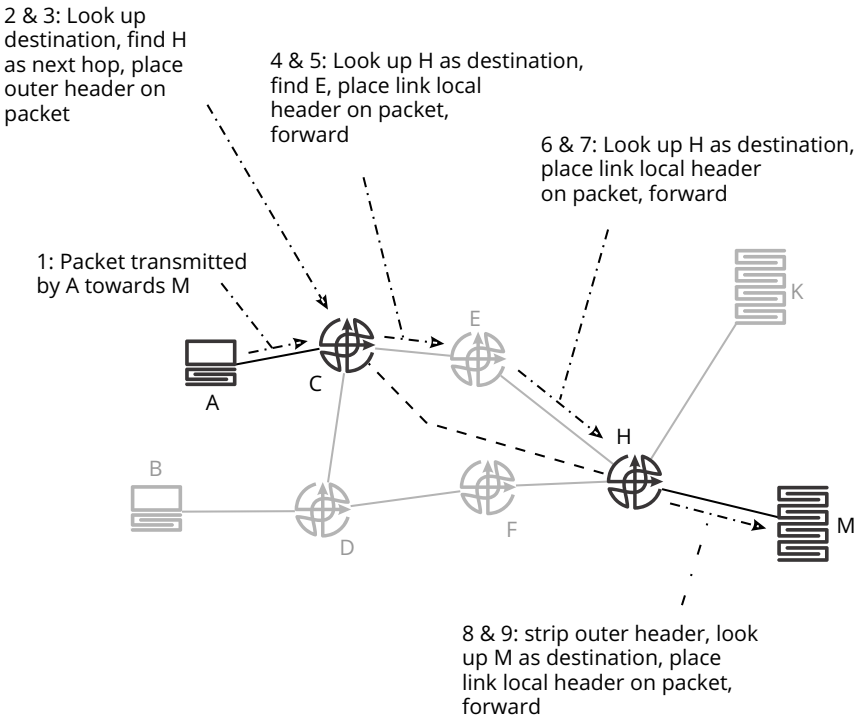


Figure 9-2 *Forwarding a Packet across a Virtual Link*

4. C now examines the new destination, which is H (remember the original destination was M). H is not directly connected, so C needs to look up how to reach H. This is called a recursive lookup, as C is looking for the path to an intermediate destination to take the packet toward, but not to, the final destination.
5. C will now place the correct information onto the packet, in a link local header, to forward the traffic to E.
6. When E receives this packet, it will strip the outer forwarding information, the link local header, and forward the traffic based on the first header C placed on the packet, during the initial lookup. This outer header tells E to forward the packet to H; E does not see or switch on the original inner header placed on the packet by A.
7. E will add a new link local header so the packet will be correctly forwarded to H, and transmit the packet on the correct interface.

8. When H receives the packet, it will strip the link local header and discover the outer header. The outer header says the packet is destined for H itself, so H will strip this header, and discover the original packet header or the inner header.
9. H will now look up M in its local routing table and discover M is locally connected. H will place the correct link local header on the packet and transmit it through the correct interface so the packet reaches M.

If C and H are using VRFs rather than tunnel interfaces, the process in the preceding list changes at steps 2 and 8. At step 2, C will look up M as a destination in the VRF associated with the [A,C] link. When C finds that traffic toward M should be forwarded through a virtual topology via H, it will place an outer header on the packet and process the packet again, based on this outer header, through the base VRF, or rather the routing table representing the physical topology. When H receives the packet, it will strip off the outer header and process the packet again using the VRF to which M is connected to look up the information needed to forward the traffic to its final destination. The tunnel interface, in this case, is replaced with a separate forwarding table; rather than processing the packet through the same table twice using two different destinations, the packet is processed through two different forwarding tables.

The term tunnel has many different definitions; for this book, a tunnel will be used to describe a virtual link where an outer header is used to encapsulate an inner header, and

- The inner header is at the same layer, or a lower layer, than the outer header (for instance, an Ethernet header carried inside an IPv6 header; normally IPv6 is carried inside Ethernet).
- At least some network devices in the path, whether virtual or physical, forward the packet based on the outer header alone.

Moving from virtual interfaces to VRFs is conceptually different enough to engender different descriptive terms. The underlay is the physical (or potentially logical!) topology through which traffic is tunneled. The overlay is the set of tunnels making up the virtual topology. Most of the time, the terms *underlay* and *overlay* are not used with single tunnels, or in the case of a service running over the public Internet. A service that builds a virtual topology across the public Internet is often called an over-the-top service.

Again, these terms are used somewhat interchangeably, and even in a very sloppy way, in the larger network engineering world. With this background, it is time to turn to use cases, in order to inform the problem set virtualization solutions need to solve.

Providing Ethernet Services over an IP Network

Although applications should not be built with Ethernet connectivity as an underlying assumption, many are. For instance:

- Some storage and database vendors build their devices with the assumption that *Ethernet connectivity means short distance and short delay,* or they design systems on top of proprietary transport protocols directly on top of Ethernet frames, rather than on top of Internet Protocol (IP) packets.
- Some virtualization products embed assumptions about connectivity into their operation, such as the reliability of the Ethernet to IP address cache for the default gateway and other reachable destinations.

These kinds of applications require what appears to be an Ethernet link between the devices (whether physical or virtual) running different nodes or copies of the application. Beyond this, some network operators believe running a large flat Ethernet domain is simpler than running a large-scale IP domain, so they would prefer to build the largest Ethernet domains they can (*"switch where you can, route where you must"* was a common saying in the days when switching was performed in hardware, while routing was performed in software, so switching packets was much faster than routing them). Some campuses are also built with the underlying idea of never asking a device to switch their IP address once they are connected. As users may be connected to different Ethernet segments based on their security domain, so each Ethernet segment must be available at every wireless access point and often at each Ethernet port in the campus.

Given a network based on IP, which anticipates Ethernet as one of the many transports on top of which IP will run, how can you provide Ethernet connectivity to devices interconnected over an IP network? Figure 9-3 illustrates the problems to be solved.

In Figure 9-3, a process running on A, with the IP address 2001:db8:3e8:100::1 needs to be able to communicate with a service running on B with the IP address 2001:db8:3e8:100::2 as if they are on the same Ethernet segment (the two services need to see one another in neighbor discovery, etc.). To make the problem more complex, the service at A also needs to be able to move to K without changing its local neighbor discovery cache or default router. The network itself, which is shown as a small section of a spine and leaf fabric, is a routed network running IPv6.

What would be required to allow the requirements to be met?

There must be a way to carry Ethernet frames over the IP network separating the servers. This would normally be some form of tunneling encapsulation, as described at the beginning of this section. Tunneling would allow Ethernet frames to

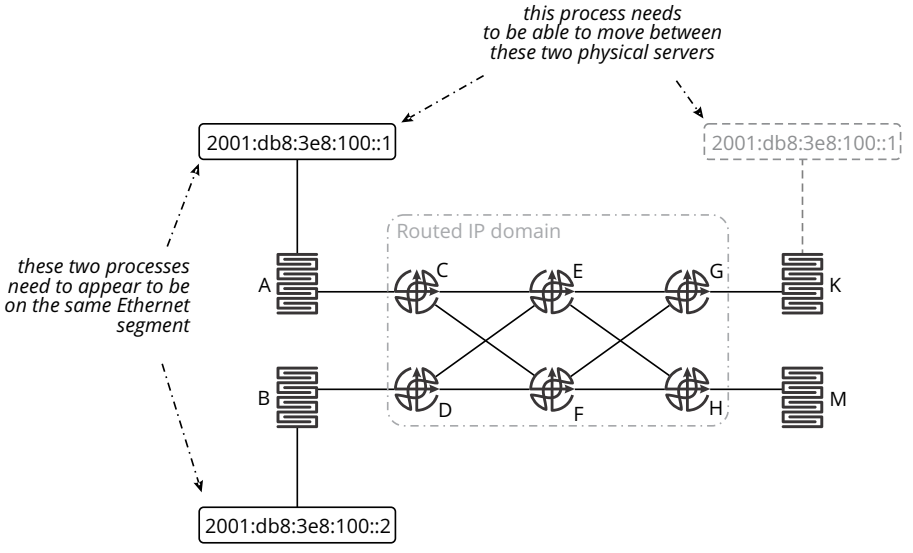


Figure 9-3 *Ethernet over IP Problem Example*

be received at C, for instance, encapsulated in some sort of outer header so they can be transported across the routed network. When the packet containing the Ethernet frame reaches D, this outer header can be stripped off and the Ethernet frame forwarded locally. From the perspective of D, the frame is locally originated.

There must be a way to learn about the destinations reachable via the tunnel and draw traffic into the tunnel. These are actually two separate, but related, problems. Drawing traffic into the tunnel might involve running a second control plane with its own VRFs, or adding additional information into an existing control plane about the Ethernet Media Access Control (MAC) addresses reachable at each edge router.

There may be a requirement to transfer Quality of Service (QoS) markings from the inner header to the outer header, so traffic is handled correctly when it is forwarded. See Chapter 8, "Quality of Service," for more information on carrying QoS markings between the two headers in a tunnel.

Virtual Private Access to a Corporate Network

Almost every organization has remote workers of some sort, either full time, or just people who travel, and most organizations have remote offices of some kind, where a small group of people work away from the main office to interact with a local community in some way, such as retail or sales. All of these people still need access to

network resources, such as email, travel systems, files, etc. These services cannot be exposed to the public Internet, of course, so some other access mechanism must be provided. Figure 9-4 illustrates the problem space.

There are two primary concerns in this use case:

- How can the traffic between the individual host—B—and the three hosts in the small office—C, D, and E—be protected from being intercepted and read by an attacker? How can the destination addresses themselves be protected from exposure into the public network? These problems involve some sort of security, which, in turn, implies some form of packet encapsulation.
- How can the quality of the user's experience in these remote locations be managed to support voice over IP and other real-time applications? Because providers on the Internet do not support quality of service, some other form of quality assurance must be provided.

The problem set to solve here, then, includes two more general issues.

- **There must be a way to encapsulate the traffic being carried across the public network** without exposing the original header information and without exposing the information carried in the packet to inspection. The easiest solution for these problems is to tunnel (often in an encrypted tunnel) the traffic from A and F to the edge router in the organization's network, G, where the encapsulation can be removed and the packets forwarded to A.

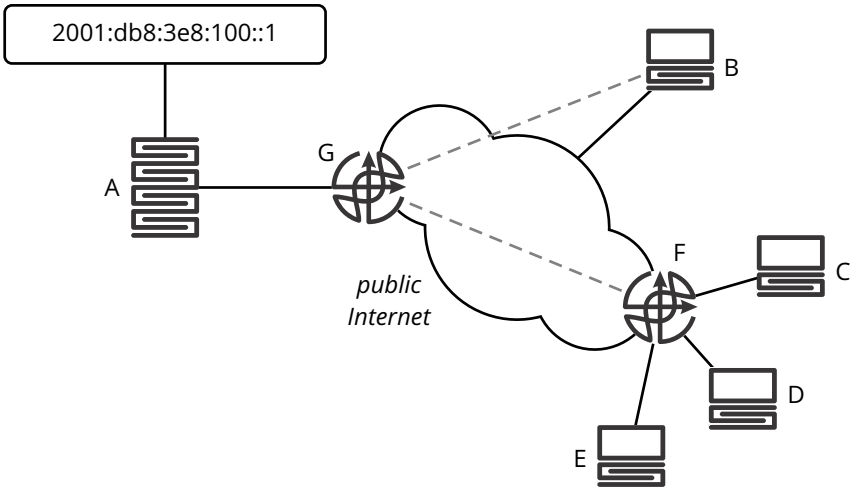


Figure 9-4 *Virtual Private Networks over a Public Network*

- **There must be a way to advertise the reachable destinations** from G toward the remote users, and the existence of (or reachability of) the remote users to G, and the network behind G. This reachability information must be used to draw traffic into the tunnels. The control plane, in this case, may need to redirect traffic among the various entry and exit points to the public network, and try to control the path of the traffic through the network, in order to ensure the remote users receive a good quality of experience.

A Summary of Virtualization Problems and Solutions

The two use cases in the preceding sections expose the two questions every network virtualization solution must solve:

How is traffic encapsulated within the tunnel so the packets and control plane information can be separated from the underlying network?

The solution for this problem is generally some form of encapsulation into which the original packet is placed as it is carried through the network. The primary consideration for the encapsulation is hardware switching support in the underlay network, to allow the efficient forwarding of encapsulated packets. A secondary consideration is the size of the encapsulating packet format; each octet of additional encapsulation header reduces the amount of payload the tunnel can carry (unless there is a differential between the Maximum Transmission Unit, or MTU, in the network designed to account for the additional header information tunneling imposes).

Note

Path MTU Detection (PMTUD) often does a poor job of detecting the MTU of encapsulated packets; because of this, manual tuning of MTU at the point where the tunnel header is imposed is often required.

How are the destinations reachable through the tunnel advertised through the network?

In more general tunneled solutions, the tunnel becomes "just another link" in the overall network topology. The destinations reachable through the tunnel, and the additional virtual link, are simply included as a part of the control plane, like any

other destinations and links. In these solutions, there is one routing or forwarding table in each device, and a recursive lookup is used to process the packet through forwarding at the point where traffic enters the tunnel, or the tunnel headend. Traffic is drawn into the tunnel by modifying the metrics so the tunnel is a more desirable path through the network *for those destinations the network operator would like to be reached through the tunnel*. This generally means largely manual solutions to the problem of drawing traffic into the tunnel, such as setting the tunnel metric lower than the path over which the tunnel runs, and then filtering the destinations advertised through the tunnel to prevent the advertisement of destinations that should be unreachable through the tunnel. In fact, if the destinations reachable through the tunnel include the tunnel termination point (the tunnel tailend), a permanent routing loop can form, or the tunnel will cycle between forwarding traffic correctly and not forwarding traffic at all.

In overlay and over-the-top solutions, a separate control plane is deployed (or a separate database of reachability information is carried for the destinations reachable in the underlay and overlay in a single control plane). Destinations reachable through the underlay and overlay are placed into separate routing tables (VRFs) at the tunnel headend, and the table used to forward traffic is based on some form of classification system. For instance, all the packets received on a particular interface may be placed into an overlay tunnel automatically, or all the packets with a specific class of service set in their packet headers, or all traffic destined to a specific set of destinations. Full overlay and over-the-top virtualization mechanisms do not generally rely on metrics to draw traffic into the tunnel at the headend.

One other optional requirement is to provide for quality of service, either by copying the QoS information from the inner header to the outer header, or by using some form of traffic engineering to carry traffic along the best available path.

Segment Routing

Segment routing (SR) may, or may not, be considered a tunneled solution, based on the specific implementation, and how strongly you want to adhere to the definition of tunnels presented in the "Understanding Virtual Networks" section earlier in this chapter. This section will consider the basic concept of segment routing and two possible implementation schemes—one using IPv6 flow labels and one using Multiprotocol Label Switching (MPLS) labels.

Each device in an SR-enabled network is given a unique label. A label stack describing the path in terms of these unique labels can be attached to any packet, causing it to take the specific path indicated. Figure 9-5 illustrates.

Each router in Figure 9-5 advertises an IP address as an identifier along with a label attached to this IP address. In SR, the label attached to the router identifier is called a node segment identifier (node SID). As each router in the network is assigned a unique label, a path can be described through the network using just these labels. For instance:

- If you wanted to forward traffic from A to K along the path [B,E,F,H], you could describe this path using the labels [101,104,105,107].
- If you wanted to forward traffic from A to K along the path [B,D,G,H], you could describe this path using the labels [101,103,106,107].

The set of labels used to describe a path is called the label stack. There are two links between D and H; how can this be described? There are several options available in SR, including:

- The label stack may include just the node SIDs describing the path through the network in terms of the routers, as previously shown. In this case, if the label stack included the pair [103,107], D would simply forward to H normally, based on local routing information, so it would use whatever local process it would use in forwarding any other packet, such as load sharing across the two links, to forward the SR-labeled traffic, as well.

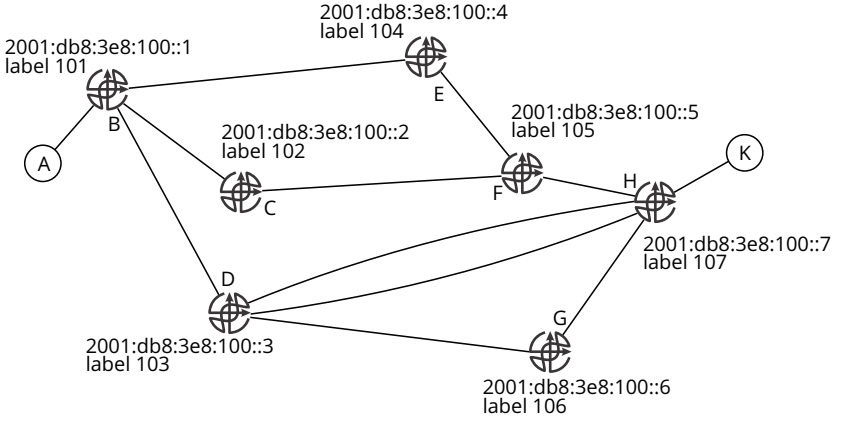


Figure 9-5 *Segment Routing*

- The label stack could include an explicit label to load share over any available set of paths available at this point in the network.
- H could assign a label per inbound interface, as well as a node SID tied to its local router identifier. These labels would be advertised just like the node SID, but as they describe an adjacency, they are called an adjacency SID. The adjacency SID is locally unique; it is unique to the router advertising the adjacency SID itself.

A third kind of SID, the prefix SID, describes a specific reachable destination (a prefix) within the network. A node SID can be implemented as a prefix SID tied to a loopback address on each router in the network.

The entire path does not need to be described by the label stack. For instance, the label stack [101,103] would direct traffic to B, then to D, but would then allow D to use any available path to reach the destination IP address at K. The label stack [105] would ensure traffic passing through the network toward K would pass through F; it does not matter how the traffic reached that point in the network, nor how it was forwarded after it reaches F, so long as it passes through F while being forwarded toward K.

Each label in the stack represents a segment; packets are carried from label to label across each segment in the network to be transported from the headend of the path to the tailend of the path.

Segment Routing with Multiprotocol Label Switching

MPLS was invented as a way to blend the advantages of Asynchronous Transfer Mode (ATM), which is no longer widely deployed, with IP switching. In the earlier days of network engineering, the chipsets used for switching packets were more constrained in their capabilities than they are now; many of the chipsets being used were Field Programmable Gate Arrays (FPGAs) rather than Application-Specific Integrated Circuits (ASICs), so the length of the field on which the packet was switched was directly correlated to the speed at which the packet could be switched. It was often easier to recycle a packet, or to process it twice, than it was to include a lot of complex information in the header so the packet can be processed once.

Note

Packet recycling is still often used in many chipsets to support inner and outer headers, or even to process different parts of a longer, more complex, packet header.

MPLS encapsulates the original packet into an MPLS header, which is then used to switch the packet through the network. Figure 9-6 shows the MPLS header.

The entire header is 32 bits; the label is 20 bits. Three operations can be carried out by an MPLS forwarding device:

- The current label in the MPLS header can be swapped with another label (SWAP).
- A new label can be pushed onto the packet (PUSH).
- The current label can be popped, and the label under the current label processed (POP).

The PUSH and POP operations are carried directly into SR; the SWAP operation is implemented in SR as a CONTINUE, which means the current label is swapped with the same label (i.e., a header with the label 100 will be replaced with a label of 100), and the processing of this current segment will continue. The easiest way to understand the processing is through an example; Figure 9-7 illustrates.

In Figure 9-7, each router has a globally unique label assigned from the Segment Routing Global Block (SRGB); these are advertised through a routing protocol or some other control plane. When A receives a packet destined for N, it will choose a path through the network using some local mechanism. At this point:

- To begin the process, A will PUSH a series of MPLS headers on the packet that describe the path through the network, [101,103,104,202,105,106,109, 110]. When A switches the packet toward B, it will POP the first label in the stack, as there is no need to send B its own label in a header. The label stack on the [A,B] link will be [103,104,202,105,106,109,110].
- When B receives the packet, it examines the next label on the stack. Finding the label to be 103, it will POP this label and forward the packet to D. The SR label

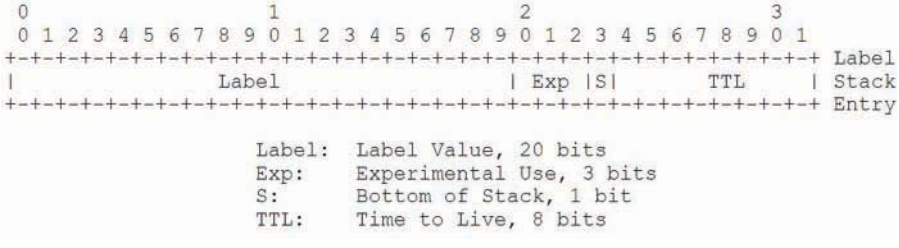


Figure 9-6 *The MPLS Header*

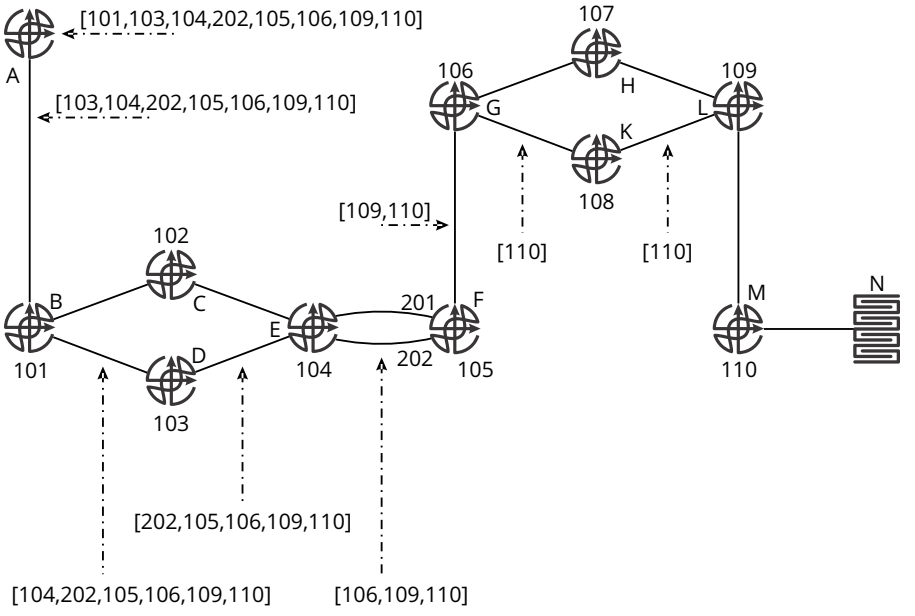


Figure 9-7 *Switching a Packet through a Series of Segments*

stack, in this case, has picked out one of two possible equal cost paths through the network, so this is an example of SR choosing a specific path. The label stack on the [B,D] link will be [104,202,105,106,109,110].

- When D receives the packet, the top label on the stack will be 104; D will POP this label and send the packet to E. The label stack on the [D,E] link will be [202,105,106,109,110].
- When E receives this packet, the top label on the stack is 202. This is an adjacency selector, so it selects for a specific interface rather than a specific neighbor. E will select the correct interface, the lower of the two interfaces in the illustration, and POP this label. The top label is now the node SID for F, which can be removed, since the packet is being transmitted to F; E will recycle the packet and POP this label as well. The label stack on the [E,F] link will be [106,109,110].
- When the packet reaches F, the next label in the stack is 106. This label indicates the packet should be transmitted to G. F will POP the label and transmit it to G. The label stack on the [F,G] link will be [109,110].

- When the packet reaches G, the next label on the stack is 109, which indicates the packet should be forwarded toward L. As G is not directly connected to L, it can use a local, loop-free (generally the shortest) path toward L. In this case, there are two equal cost paths toward L, so G will POP the 109 label and forward over one of these two paths toward L. On the [G,L] segment, the label stack is [110].
- Assume G chooses to send the packet via K. When K receives the packet, it will have a label stack containing [110], which is not the local label, nor it is an adjacent node. In this case, the label needs to remain the same, or the segment needs to CONTINUE. To implement this, K will SWAP the current label, 110, for another copy of the same label, so the K will forward the traffic with the same label. On the [K,L] link, the label stack will be [110].
- When L receives the packet, the only remaining label will be 110, which indicates the packet should be forwarded to M. L will POP the 109 label, effectively removing all the MPLS encapsulation, and forward the packet to M.
- When M receives the packet, it will forward the packet using normal IP to N, the final destination.

The *stack of labels* concept in MPLS is implemented as a series of MPLS headers stacked on top of one another. Popping the label means to remove the topmost label, pushing a label means adding a new MPLS header onto the packet, and continuing means swapping the label with an identical label. When you are working with a stack of labels, the concepts of *inner* and *outer* are often confusing, particularly as many people use the idea of a label and a header interchangeably. Perhaps the best way to reduce confusion is to use the term *header* to refer to the entire label stack and the original header being carried inside MPLS, while referring to the labels as *individual labels in the stack*. The inner header would then be the original packet header, while the outer header would be the stack of labels; the inner label would be the next label on the stack at any point in the packet's travels through the network, while the outer label would be the label on which the packet is actually being switched.

Although the example given here uses IP packets inside MPLS, the MPLS protocol is designed to carry just about any protocol, including Ethernet. SR MPLS is not, therefore, limited to being used to carry a single type of traffic, but can also be used to carry Ethernet frames over an IP/MPLS-based network. This means SR can be used to support the first use case discussed in this chapter, providing Ethernet services over an IP network.

Is MPLS a Tunnel?

Many bits, in the form of the written and spoken words, have been spilled on the question of whether or not MPLS is a tunneling protocol. The way tunneling is defined here is that it is an action, rather than a protocol; this is an intentional attempt to separate the idea of the tunneling protocol from the concept of tunneling as an action taken in carrying traffic through the network. In the case of MPLS, this means it may, or may not, be a tunneling protocol, depending on how it is being used—just like any other protocol. For instance, if you have a stack of labels placed on top of a packet with an IP header, the outer label, the one on which the packet is being switched, is not (technically) a tunnel. This outer header, in an MPLS network, is actually local to the segment, so it is either popped or pushed at every router. This is analogous to an Ethernet header on a per link basis. The inner header, however, is being carried within the MPLS packet, and hence is technically being tunneled. The inner label is not used *at the current device* for switching the packet; it is simply carried *as part of the packet*.

This definition is not perfect—few definitions in the real world are. For instance, in the case of an MPLS SWAP or SR CONTINUE, is the label being used to switch the packet or not? Also, unlike the Ethernet header on a packet, the MPLS header is actually used in making a forwarding decision. The Ethernet header, in contrast, is simply used to reach the next hop and then discarded. The more appropriate comparison would perhaps be: The MPLS header is like the Ethernet header used to reach the hop beyond the device that the router is currently transmitting to.

Regardless of these limitations, this definition will generally suffice to mentally manage the difference between tunneling and not tunneling in MPLS, as well as most other protocols.

Segment Routing with IPv6

The operation of SR on MPLS and SR on IPv6 is similar in all respects except how the label stack is carried and processed. SR headers in IPv6 are carried in the flow label field, shown in Figure 9-8.

In the IPv6 SR implementation, the SR label stack is carried in the routing header of the IPv6 packet header. The information in this header is designed specifically to provide information about the nodes through which "this packet" should pass when being routed through the network, so it serves the same purpose as the SR label

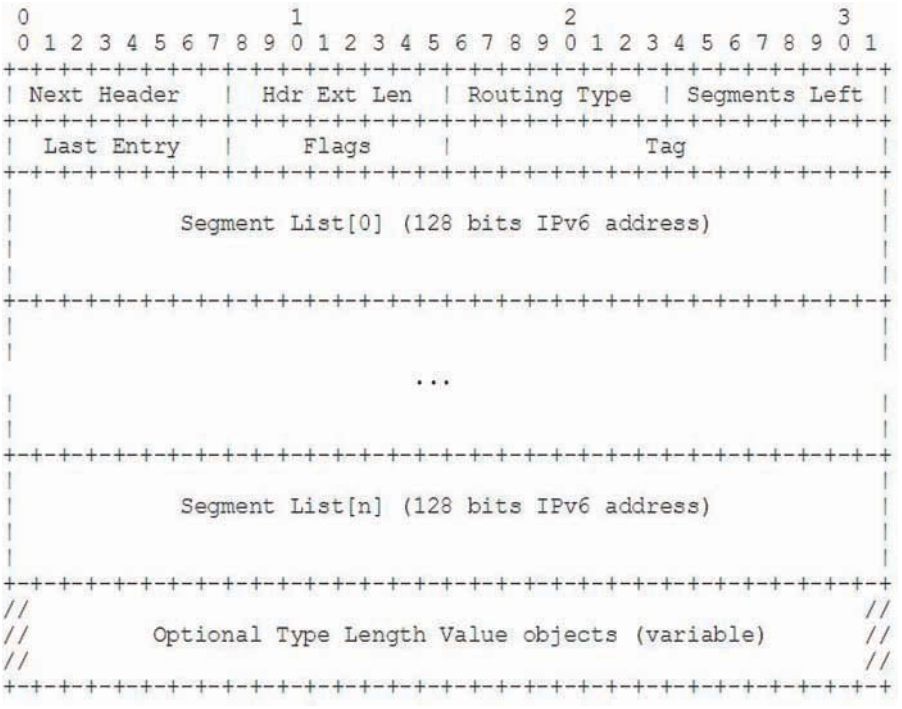


Figure 9-8 *The SR IPv6 Header Extension*

stack. In the case of IPv6 implementations of SR, each label is 128 bits, so some local IPv6 address can be used as an SID.

The one interesting point is the IPv6 specifications indicate the IPv6 header must not be changed by a router when processing the packet (see RFC8200 for further details). Instead of popping, pushing, and swapping labels, then, SR IPv6 relies on each node along the path having a pointer to the current label in the stack being processed.

Signaling Segment Routing Labels

SR is technically a source routing mechanism, because the source chooses the path through the network—although the source routing in SR can be much looser than traditional source routing. For each label on the stack, there are two possible ways a node along the path can process the packet:

- The label provides explicit instructions about how the packet should be handled at this device; POP or CONTINUE the segment (label) and process the packet accordingly.

- The label does not provide explicit instructions about how the packet should be handled at this device; use local routing information to forward the packet and CONTINUE the segment.

In neither case does the processing node need to know about the entire path to switch the packet; it either simply follows the label path as specified, or it processes the packet based on purely local information. Because of this paradigm, signaling SR is simple. Two types of signaling need to occur.

The local node, prefix, and adjacency SIDs assigned to a node in the network need to be advertised by each node in the network. This signaling is primarily carried in routing protocols; for instance, the Intermediate System to Intermediate System (IS-IS) protocol is extended by the draft *IS-IS Extensions for Segment Routing*[1] to carry prefix SIDs using a sub Type Length Value (sub-TLV), as shown in Figure 9-9.

Extensions to other routing and control plane protocols are proposed for standardization, as well; see the "Further Reading" section at the end of the chapter for a list of these extension proposals. Because path calculation in SR is source based, there is no need to carry a path in a distributed routing protocol. The only real need is to provide each node in the network with the information needed to carry SR node, prefix, and adjacency information.

In the case where SR paths are calculated by a centralized device or controller, there needs to be a way to advertise a label path to use in order to reach a particular destination. Extensions have been proposed to the Border Gateway Protocol (BGP) in *Advertising Segment Routing Policies in BGP*,[2] and in the Path Computation Element Protocol (PCEP) in *PCEP Extensions for Segment Routing*.[3] These two kinds of advertisements are separate from one another, as the only node in the network that needs to either calculate or impose the segment list is the tunnel headend or the point where traffic enters the segment path.

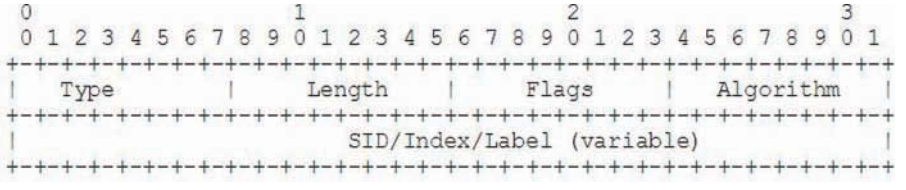


Figure 9-9 *The IS-IS Sub-TLV for Carrying a Prefix SID*

1. Previdi et al., "IS-IS Extensions for Segment Routing."
2. Previdi et al., "Advertising Segment Routing Policies in BGP."
3. Sivabalan et al., "PCEP Extensions for Segment Routing."

Software-Defined Wide Area Networks

Many organizations need to provision and support large numbers of remote offices. For instance:

- Retail chains may have hundreds or even thousands of stores and locations worldwide.
- A regional bank may have hundreds of branch offices and thousands of cash machine locations.

When fixed location private line services were all service providers offered at any scale, these kinds of problems were solved using large-scale hub-and-spoke networks. Figure 9-10 illustrates a hub-and-spoke network.

The network shown in Figure 9-10 is actually rather small; the three dots in the center of the remote sites may represent hundreds or thousands of additional sites. In many implementations (especially older ones), the links between the two hub routers, A and B, and the remotes, such as C and N, are point-to-point links. This means the hub router must have an interface configured for each remote router,

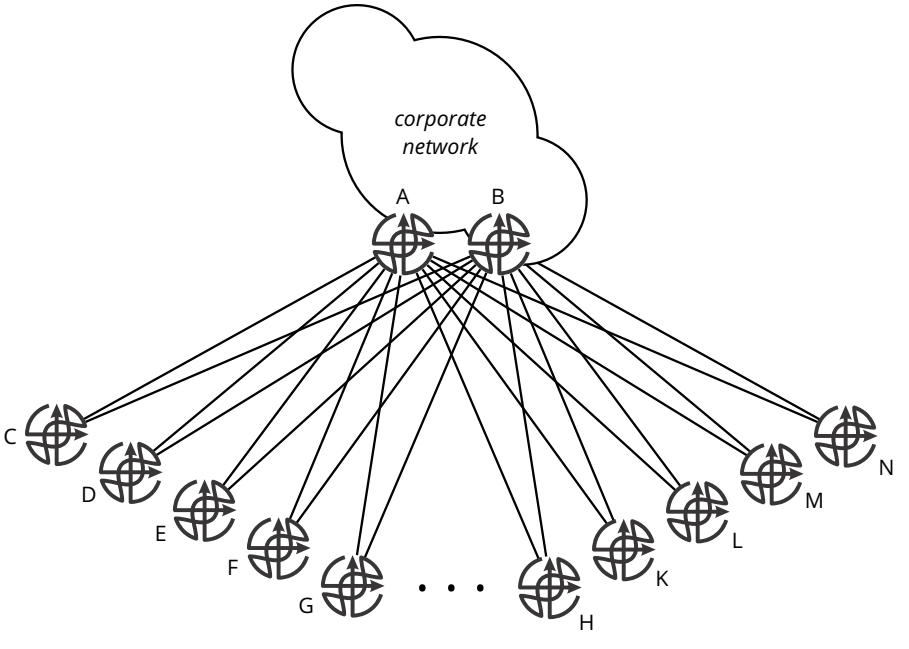


Figure 9-10 *A Hub-and-Spoke Network*

routing filters, packet filters, and any Quality of Service configurations. Not only is this a major problem from a configuration perspective, but it is also difficult to maintain thousands of individual neighbors in terms of processor and memory utilization.

To reduce the amount of processing power required in maintaining such a network, protocols were modified to prevent treating the remote sites as if they were part of the tree. Instead, these modifications allowed these remote sites to be treated as if they were leaves, or stub networks. Another step toward making these kinds of networks easier to create and manage was using a point-to-multipoint interface (with the appropriate underlying technology, such as Frame Relay), at the hub routers. When the connections to the remote sites are configured as point-to-multipoint, the hub routers, A and B, treat all the spokes as if they are on a single broadcast segment (like an Ethernet segment, in effect). Each spoke router, however, still treats its connection to the hub routers as a point-to-point link. Even with these modifications, building and maintaining such large networks is still very difficult. Links must be purchased, and managed to each remote site, remote equipment must be configured and managed, the configuration of the hub routers must be managed, etc.

Software-Defined Wide Area Network (SD-WAN) solutions were originally developed to solve this specific problem set. Originating in Cisco's Dynamic Multipoint Virtual Private Network (DMVPN), the idea behind the DMVPN was to use a tunneled overlay, or over-the-top, network running on top of the public Internet. This allowed the remote sites to use locally available Internet connectivity, rather than purchasing a circuit per site, and reduced configuration and maintenance time through autoconfiguration and other tools.

SD-WAN takes the concept of an over-the-top network one step further. An SD-WAN solution is normally built using several components:

- A specialized appliance or virtualized service to replace the routers normally placed at the hub and spoke locations
- A modified version of a standard routing protocol to provide reachability (and potentially one measure of circuit liveness) and to pass policies through the network
- An implementation of either IP Security (IPsec) or Transport Layer Security (TLS) to provide secure tunneled transport between the hub-and-spoke devices
- A controller to monitor the state of each virtual link, the applications using the link, and the amount of goodput versus the amount of traffic, and to make dynamic adjustments to traffic flow and QoS settings to optimize application operation across the over-the-top virtual network

There are many different ways in which SD-WANs can be implemented; for instance:

- The SD-WAN can replace the "last mile"; rather than installing a circuit to each remote site, you can use SD-WAN solutions to reach an exchange or colocation point, and then carry the traffic through a more traditional service through a provider back to the hub routers (this is a form of backhaul).
- The SD-WAN can replace the entire path from the organization's network to the remote sites.
- The SD-WAN can be used to draw traffic into a cloud service, where some preliminary processing might take place, or some applications might be deployed, with just traffic that must be carried into the organization's network carried the rest of the way into the hub routers.

There are tradeoffs with SD-WAN and other over-the-top solutions, as there are with any other networking technology. For instance, pushing corporate remote site traffic over a "plain" public Internet connection (or pair of services, or some other Ethernet-terminated service) may be "good enough" in some situations, but providers tend to treat traffic in higher-priced services better (naturally enough), particularly in outages.

Complexity and Virtualization

Virtualization is often undertaken to find a simpler way to solve some of the problems noted in the initial sections of this chapter, such as traffic separation. There are, as with all things in the network engineering world, tradeoffs. In fact, if you have not found the tradeoff, you have not looked hard enough. This section will consider some (though certainly not all) of the various complexity tradeoffs in the realm of network virtualization. The basis of this discussion will be the complexity tradeoff triad considered in Chapter 1, "Fundamental Concepts":

- **State:** The amount of state and the speed at which state in the network changes (particularly the control plane)
- **Optimization:** The optimal use of network resources, including such things as traffic following the shortest path through the network
- **Surface:** The number of layers, the depth of their interaction, and the breadth of their interaction

Interaction Surfaces and Shared Risk Link Groups

Every virtualization system ever conceived, implemented, and deployed creates shared risk of some sort. For instance, consider a single link that is carrying several virtual links, each of which is carrying traffic. It should be obvious (in fact trivial) to observe that if the single physical link fails, all of the virtual links will fail. Of course, you can simply reroute the virtual links onto another physical link. Right? Maybe or maybe not. Figure 9-11 illustrates.

From the perspective of A and D, there are two links available through B and C, each one providing independent connectivity between the host and the server. The reality is, however, both provider 1 and provider 2 have purchased virtual links through a single link from provider 3. When the single link in provider 3's network fails, the traffic might be redirected from the path through provider 1 to the path through provider 2, but as both links share the same physical infrastructure, neither link will be able to carry the traffic.

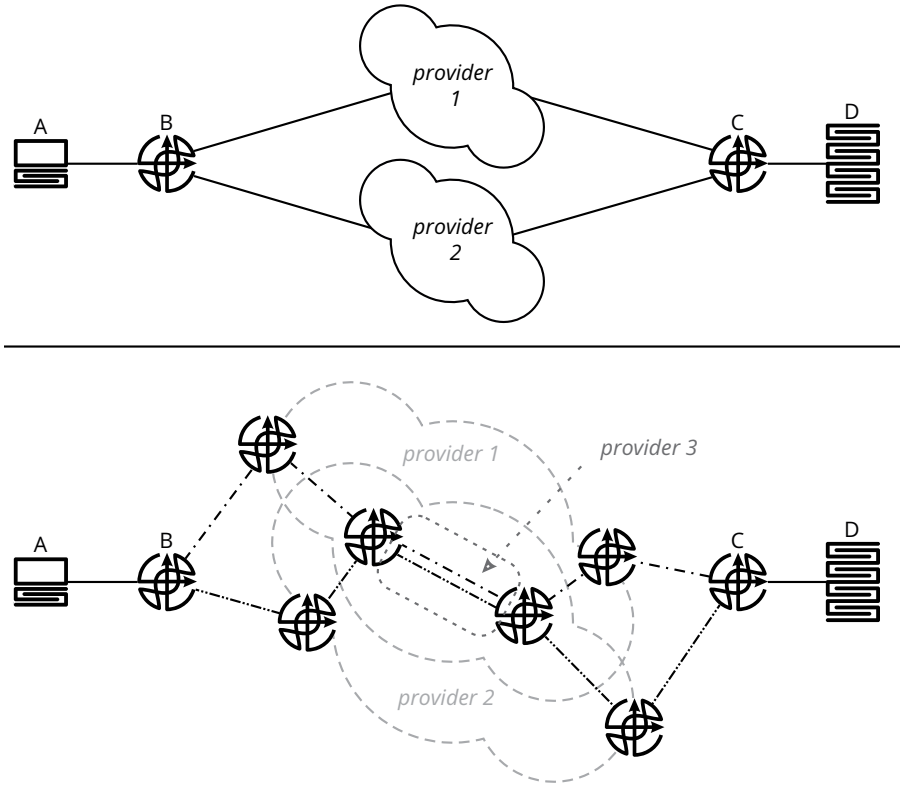


Figure 9-11 *Shared Risk Link Groups*

The two links in this situation are said to share fate, because they are part of a Shared Risk Link Group (SRLG). It is possible to find and work around SRLGs, or shared fate situations, but doing so adds complexity to the control plane and/or network management. For instance, there is no way to discover these shared fate situations without either manually testing different failure situations at the physical level or examining network maps to find places where multiple virtual links pass over the same physical link. In the situation described in Figure 9-11, finding the shared fate situation would be almost impossible, as neither provider is likely to tell you it is using a link from a second provider, shown as provider 3 in the illustration, in order to provide service.

Once these shared fate situations are discovered, some action must be taken to avoid a single failure from causing a major network outage. This normally requires either injecting information into the design process, adding complexity to the design, or injecting information into the control plane (see RFC8001 as an example of the type of signaling required to manage SRLGs in a traffic-engineered control plane).

Essentially, the problem comes down to this set of statements:

- Virtualization is a form of abstraction.
- Abstraction removes information about the network state in order to reduce complexity or provide services through the implementation of policy.
- Any nontrivial reduction of information about the network state will reduce the optimal use of resources in some way.

The only counter to the final state of these three is to leak information through the abstraction, so optimal use of resources can be restored—in this case, the failure of a single link not causing a complete failure of traffic flow through the network. The only solution, then, is to make the abstraction a leaky abstraction, reducing the effectiveness of the abstraction at controlling the scope of state and the implementation of policy.

Interaction Surfaces and Overlaid Control Planes

It is common, in network engineering, to overlay two routing protocols, or two control planes, on top of one another. While this is not often considered a form of virtualization, it is, in fact, just that—splitting state between two different control planes to control the amount of state, and the rate at which state changes, to reduce the complexity of both control planes. This is also common when running virtual overlays in a network, as there will be an underlay control plane providing reachability between the tunnel headend and tailend, and an overlay control plane providing

reachability within the virtual topology. Two overlaid control planes will interact in sometimes unexpected ways. Figure 9-12 is used to illustrate.

In Figure 9-12:

- Every router in the network, including B, C, D, and E, is running two control planes (or, if it is simpler, routing protocols, hence *protocol 1* and *protocol 2* in the illustration).
- Protocol 1, the overlay, depends on protocol 2, the underlay, to provide reachability between the routers running protocol 1.
- Protocol 2 does not have any information about connected devices, such as A and F; this information is all carried in protocol 1.
- Protocol 1 requires much longer to converge than protocol 2.
- The lower-cost path from B to E is through C, rather than through D.

Given this set of protocols, assume C, in Figure 9-12, is removed from the network, the two control planes are allowed to converge, and then C is reconnected to the network. What will be the result? The following will occur:

- After C is removed, the network will reconverge with two paths in the local routing table at B:
  - F is reachable through E.
  - E is reachable through D.

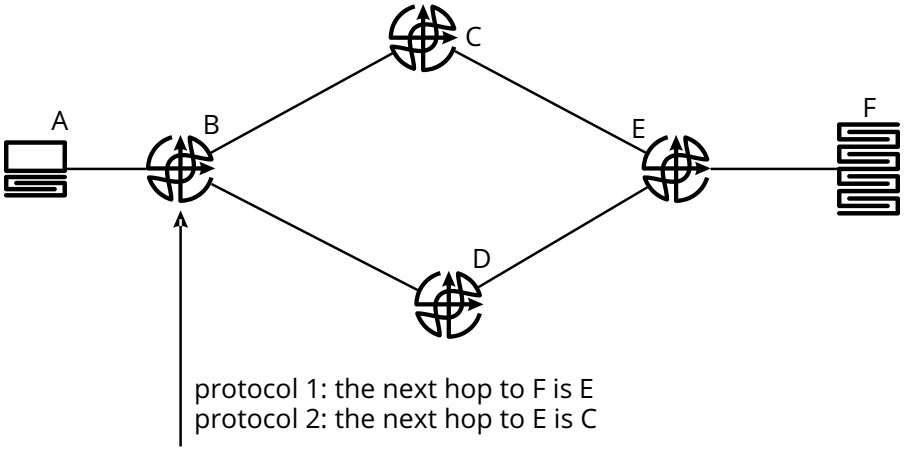


Figure 9-12 *Overlaid Control Planes, Virtualization, and Interaction Surfaces*

- Once C is reconnected to the network, protocol 2 will converge quickly.
- Once protocol 2 is reconverged, the best path toward E, from the perspective of B, will be through C.
- Therefore, B will now have two routes in the local routing table:
  - F is reachable through E.
  - E is reachable through C.
- B will shift to the new routing information, and hence will send traffic toward F through C *before protocol 1 converges,* and hence *before C has learned about the best path to F.*
- From the time when B starts forwarding traffic destined to F to C, and the time when protocol 1 convergences, traffic destined to F will be dropped.

This is a rather simple example of overlaid protocols interacting in an unexpected way. To solve the problem, you need to inject information about the state of the convergence of protocol 1 into protocol 2, or you must somehow force the two protocols to converge at the same time. In either case, you are essentially adding state back into the two protocols to account for their difference in convergence time, as well as creating an interaction surface between the protocols.

Note

This example describes the actual convergence interaction between IS-IS and BGP, or the Open Shortest Path First (OSPF) protocol and BGP. To solve this problem, the faster protocol is configured to wait until BGP has converged before installing any routes in the local routing table.

Final Thoughts on Network Virtualization

Network virtualization is an important tool in the hands of the engineer to simplify designs and solve otherwise unsolvable problems. All virtualization solutions require at least two elements to solve the problems virtualization poses:

- Some way to tunnel traffic through a network so traffic can be separated out into a virtual topology
- Some way to discover and advertise reachability across the virtual topology, and some way to draw traffic into the virtual topology

There are a number of interesting, and often unexpected, interaction points between complexity and virtualization that network engineers need to be aware of. All technologies involve tradeoffs of one kind or another, so engineers should be aware of, and intentionally seek out, these tradeoffs when working with virtualization technologies.

The number of virtualization technologies available in the network world almost seems to be without limit sometimes. As network engineers sometimes say: "Please, take my tunneling protocols; there are always plenty to go around."

Further Reading

Boutros, Sami, Ali Sajassi, Samer Salam, John Drake, and Jorge Rabadan. *Virtual Private Wire Service Support in Ethernet VPN*. Request for Comments 8214. RFC Editor, 2017. doi:10.17487/RFC8214.

Deering, Dr. Steve E., and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. Request for Comments 8200. RFC Editor, 2017. doi:10.17487/RFC8200.

Drake, John, Wim Henderickx, Ali Sajassi, Rahul Aggarwal, Dr. Nabil N. Bitar, Aldrin Isaac, and Jim Uttaro. *BGP MPLS-Based Ethernet VPN*. Request for Comments 7432. RFC Editor, 2015. doi:10.17487/RFC7432.

Farrel, Adrian, Olufemi Komolafe, and Seisho Yasukawa. *An Analysis of Scaling Issues in MPLS-TE Core Networks*. Request for Comments 5439. RFC Editor, 2009. doi:10.17487/RFC5439.

Filsfils, Clarence, Kris Michielsen, and Ketan Talaulikar. *Segment Routing Part I*. 1st edition. CreateSpace Independent Publishing Platform, 2017.

Filsfils, Clarence, Stefano Previdi, Ahmed Bashandy, Bruno Decraene, Stephane Litkowski, and Rob Shakir. "Segment Routing with MPLS Data Plane." Internet-Draft. Internet Engineering Task Force, June 2017. https://datatracker.ietf.org/doc/html/draft-ietf-spring-segment-routing-mpls-10.

Filsfils, Clarence, Stefano Previdi, Bruno Decraene, Stephane Litkowski, and Rob Shakir. "Segment Routing Architecture." Internet-Draft. Internet Engineering Task Force, June 2017. https://datatracker.ietf.org/doc/html/draft-ietf-spring-segment-routing-12.

Filsfils, Clarence, Stefano Previdi, Bruno Decraene, and Rob Shakir. "Resiliency Use Cases in SPRING Networks." Internet-Draft. Internet Engineering Task Force, May 2017. https://datatracker.ietf.org/doc/html/draft-ietf-spring-resiliency-use-cases-11.

Ghein, Luc De. *MPLS Fundamentals*. 1st edition. Indianapolis, IN: Cisco Press, 2006.

Monge, Antonio Sanchez, and Krzysztof Grzegorz Szarkowicz. *MPLS in the SDN Era: Interoperable Scenarios to Make Networks Scale to New Services*. 1st edition. Beijing: O'Reilly Media, 2016.

O'Connor, Darren. *Day One: MPLS for Enterprise Engineers*. Juniper Networks Books, 2014.

O'Dell, Michael D., Joseph Malcolm, Jim McManus, Daniel O. Awduche, and Johnson Agogbua. *Requirements for Traffic Engineering Over MPLS*. Request for Comments 2702. RFC Editor, 1999. doi:10.17487/RFC2702.

Previdi, Stefano, Clarence Filsfils, Ahmed Bashandy, Hannes Gredler, Stephane Litkowski, Bruno Decraene, and Jeff Tantsura. "IS-IS Extensions for Segment Routing." Internet-Draft. Internet Engineering Task Force, June 2017. https://datatracker.ietf.org/doc/html/draft-ietf-isis-segment-routing-extensions-13.

Previdi, Stefano, Clarence Filsfils, Paul Mattes, Eric C. Rosen, and Steven Lin. "Advertising Segment Routing Policies in BGP." Internet-Draft. Internet Engineering Task Force, July 2017. https://datatracker.ietf.org/doc/html/draft-ietf-idr-segment-routing-te-policy-00.

Previdi, Stefano, Clarence Filsfils, Kamran Raza, John Leddy, Brian Field, Daniel Voyer, Daniel Bernier, et al. "IPv6 Segment Routing Header (SRH)." Internet-Draft. Internet Engineering Task Force, July 2017. https://datatracker.ietf.org/doc/html/draft-ietf-6man-segment-routing-header-07.

Psenak, Peter, Shraddha Hegde, Clarence Filsfils, and Arkadiy Gulko. "ISIS Segment Routing Flexible Algorithm." Internet-Draft. Internet Engineering Task Force, July 2017. https://datatracker.ietf.org/doc/html/draft-hegdeppsenak-isis-sr-flex-algo-00.

Psenak, Peter, Stefano Previdi, Clarence Filsfils, Hannes Gredler, Rob Shakir, Wim Henderickx, and Jeff Tantsura. "OSPF Extensions for Segment Routing." Internet-Draft. Internet Engineering Task Force, August 2017. https://datatracker.ietf.org/doc/html/draft-ietf-ospf-segment-routing-extensions-19.

———. "OSPFv3 Extensions for Segment Routing." Internet-Draft. Internet Engineering Task Force, March 2017. https://datatracker.ietf.org/doc/html/draft-ietf-ospf-ospfv3-segment-routing-extensions-09.

Sajassi, Ali, John Drake, Nabil Bitar, Ravi Shekhar, Jim Uttaro, and Wim Henderickx. "A Network Virtualization Overlay Solution Using EVPN." Internet-Draft. Internet Engineering Task Force, March 2017. https://datatracker.ietf.org/doc/html/draft-ietf-bess-evpn-overlay-08.

Sivabalan, Siva, Clarence Filsfils, Jeff Tantsura, Wim Henderickx, and Jonathan Hardwick. "PCEP Extensions for Segment Routing." Internet-Draft. Internet Engineering Task Force, April 2017. https://datatracker.ietf.org/doc/html/draft-ietf-pce-segment-routing-09.

Tappan, Dan, Yakov Rekhter, Alex Conta, Guy Fedorkow, Eric C. Rosen, Dino Farinacci, and Dr. Tony Li. *MPLS Label Stack Encoding*. Request for Comments 3032. RFC Editor, 2001. doi:10.17487/RFC3032.

Viswanathan, Arun, Eric C. Rosen, and Ross Callon. *Multiprotocol Label Switching Architecture*. Request for Comments 3031. RFC Editor, 2001. doi:10.17487/RFC3031.

Zhang, Fatai, Oscar Gonzalez de Dios, Matt Hartley, Zafar Ali, and Cyril Margaria. *RSVP-TE Extensions for Collecting Shared Risk Link Group (SRLG) Information*. Request for Comments 8001. RFC Editor, 2017. doi:10.17487/RFC8001.

Review Questions

1. How is virtualization different from mutliplexing?
2. What is the difference between virtual interface and VRF forwarding in a network device (such as a router)?
3. Some overlay control planes include the reachable destinations from both the underlay and the overlay in a single protocol. An example of this would be Ethernet VPNs (eVPNs), in which both the IP reachability of the underlay and the Ethernet reachability of the overlay are carried in a single protocol, the Border Gateway Protocol. How are the overlay and underlay reachability separated?
4. Draw a network where the interaction of the underlay and overlay control planes create a loop.
5. Under what situation would you need to have adjacency SIDs, rather than just node SIDs?
6. Describe another situation where an SRLG in an overlay over an Ethernet network would be impossible to detect but would cause multiple virtual links to fail when a single link fails.
7. Research virtual circuits in Frame Relay. Would you consider this a tunneling mechanism or not? Explain.

Chapter 10

Transport Security

Learning Objectives

After reading this chapter, you should be able to:

- Understand the concept of data exhaust and how it can be used by an attacker
- Understand the basic operation of encryption systems, including asymmetric and symmetric encryption
- Understand the concept of a web of trust
- Understand the basic principles of how keys can be exchanged
- Understand the basic techniques used to hide user information
- Understand the man-in-the-middle attack
- Understand the basic operation of Transport Layer Security

When you log in to a financial or medical website and sign in, you should expect that the information you retrieve cannot be intercepted and read by anyone along the path between your computer and the server. A less obvious, but just as important, problem is the information you send to the site should not be open to change while it is being transported by the network.

But how can these things be ensured? These are two of the areas transport security can be used to address. This chapter will consider the transport security problem space, followed by an investigation of several kinds of solutions, including encryption. Finally, this chapter will look at the Transport Layer Security (TLS) specification as an example of transport layer encryption.

The Problem Space

Security generally resolves to one of four problems: proving the data has not been changed in transmission, preventing anyone other than the intended recipient from accessing the information, protecting the privacy of the humans using the network, and proving information has been delivered (or work has been done). The second and third problems, preventing unauthorized access to data as it crosses the network and protecting user privacy, are related problems but will be treated separately in the following sections. The final problem noted, the *proof of traversal* problem (which is similar to the *proof of work* problem faced in other information technology contexts), is not considered here, as it is an area of active research with few deployed systems.

Validating Data

If you log in to your bank's website and transfer $100 from one account to another, you would likely be upset if the amount actually transferred was $1,000 instead, or if the account numbers were changed so the $100 ended up in someone else's account. There are a number of other situations where making certain the data transmitted is the same as the data received, such as

- If you purchase a pair of blue shoes, you do not want a set of red ones delivered instead.
- If your doctor gives you a prescription for medicine to help your heartburn (probably resulting from the stress of working as a network engineer), you do not want medicine for arthritis (probably from typing so many documents and books) to be delivered.

There are a lot of situations where the data received must match the data transmitted, and the originator and/or receiver must be verifiable.

Protecting Data from Being Examined

The data protection examples given previously can be taken one step further: you do not want someone to see your account number, prescription, or other information as it is being transported across the network. Account numbers, passwords, and any kind of personally identifiable information (PII) are all very crucial, as these kinds of information can be used to break into accounts to steal money, or even used to steal someone's identity entirely.

How can this kind of information be protected? The primary means of protection used to prevent unauthorized users (or attackers; see Chapter 21, "Security: A Broader Sweep," later for a full definition of the elements of an attack) is encryption.

Protecting User Privacy

Privacy is not just nice to have on the global Internet; it is a requirement for users to trust the system. This is true of local networks, as well; if users believe they are being spied on in some way, they are not likely to use the network. Rather, they are likely to use *sneakernet,* printing information out and hand-carrying it, rather than transferring it over the network. While many people believe privacy is not a valid concern, there are many valid concerns in this area.

For instance, a common saying in the information management field is *knowledge is power.* Knowing about a computer or network gives you some measure of power over the computer, network, or system. For instance, assume a bank configures an automated backup for a particular database table; when the balances in the account held in the table change by a particular amount, the backup is kicked off automatically. This might seem like a perfectly reasonable sort of backup job, but it does involve some amount of data exhaust.

Note

Data exhaust is information about the physical movements of people or information that can be used to infer what those people or that information is doing. For instance, if you always take the same route to work every morning, someone can infer, once you have made some small part of the trip, combined with a time of day, you are going to work. The same sorts of data exhaust exist in the network world; if, every time, at a particular time of day, a particular piece of data of a certain size is transmitted through the network, and it happens to coincide with a particular event, such as transferring money between two accounts, then when this particular data appears, the transfer must be taking place. Browsing, email history, and other online actions all leave data exhaust, which can sometimes be used to infer the contents of a data stream even if the stream is encrypted.

The vulnerability here is: if a threat actor puts the backup together with the change in account value, that person will know specifically what the pattern of account activity is. Enough clues of this sort can be developed into an entire set of attack plans.

The same is true of people; having knowledge about people can give you some ability to influence people in specific directions. While the influence over people is not as great as the influence over machines, handing one person power over another always carries moral implications that need to be handled carefully.

Nothing to Hide

"If you have done nothing wrong, you have nothing to hide." This is a widespread fallacy worth considering for a moment. The fallacy first implies *hiding something means you have done something wrong* is the primary point of hiding information about yourself. In reality, as noted earlier, information can often be used to shape a person's (or a culture's) perceptions of reality, beliefs, and actions in unhealthy ways. Humans are (it is generally but not universally agreed) flawed. The presumption should always be information is given when it is needed for a specific reason, and is not kept when it is no longer needed, to protect people from using data about another person in unethical or unintentional ways. Most often, in modern information technology systems, the presumption runs the other way—"information wants to be free," and should only be controlled when there is a specific reason to control it.

The second point to remember is, as mentioned, *humans are flawed*. There is probably some embarrassing action you have taken in the past, or something someone would consider "wrong" or "harmful." While it is important not to cover up serious and real crimes, it is also important to allow some grace in the interaction between humans, just in order to make a society work.

The third point to remember is, returning to *knowledge is power,* that power can be used asymmetrically. Companies often hide information about themselves from users and yet expect users (and employees) to be completely transparent. Asymmetrical power can be harmful to people in the real world. Remember that every company is ultimately made up of employees, each of whom is a customer in some other context, and each of whom deserves privacy.

One phrase used to describe the problem of information leaked by users as they interact on a network is data exhaust (see the previous note). There are many forms of data exhaust, some of which are nearly impossible to defend against.

The Solution Space

While every solution to the security and privacy issues described in the preceding sections generally involves hard math, this section will (attempt to) describe the

solutions without the math. Readers who would like to learn more about the mechanisms considered here are encouraged to look at the "Further Reading" section at the end of the chapter for resources describing specific kinds of encryption algorithms and the math involved.

Encryption

Encryption takes a block of information (the *plaintext)* and encodes it using some form of mathematical operation to obscure the text, resulting in the *ciphertext*. To recover the original plain text, the mathematical operations must be reversed. While encryption is often approached as a mathematical construct, it is sometimes easier to start by thinking of it as a substitution cipher with a substitution table that varies based on the key used. Figure 10-1 illustrates.

Figure 10-1 shows a four-bit block of information—a trivial example but still useful to illustrate the point. The encryption process is conceptually a series of straight substitutions:

- If 0001 is found in the original block of data (the plaintext) and key 1 is in use, then 1010 is substituted into the actual transmitted stream (the ciphertext).
- If 0010 is found in the plaintext and key 1 is in use, then 0100 is substituted into the transmitted data.

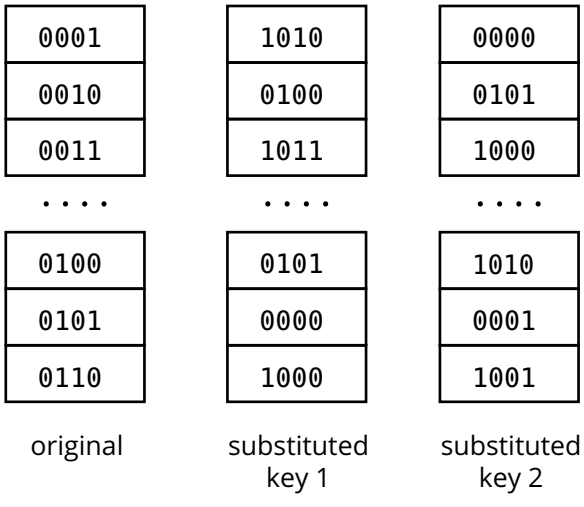


Figure 10-1 *A Cipher Block as a Substitution Table*

- If 0001 is found in the plaintext and key 2 is in use, then 0000 is substituted into the transmitted data.
- If 0110 is found in the plaintext and key 2 is in use, then 1001 is substituted into the transmitted data.

The process of substituting one block of bits for another is called a *transform*. These transforms must be symmetrical: they must not only allow the plaintext to be encrypted to the ciphertext, but they must also allow the plaintext to be recovered (unencrypted) from the ciphertext. In a substitution table, this process involves looking up the key on the ciphertext side of the table and substituting the plaintext equivalent.

The size of the substitution table is determined by the size of the block, or the number of bits encoded at one time. If a 128-bit block is used, for instance, the lookup table would need to have 2^{128} entries—a very large number indeed. This kind of space can be still be searched by an efficient algorithm quickly, so the block must have some other features than simply being large.

The first is that the ciphertext side of the substitution block must be *as random as possible*. For a transform to be ideal, any pattern found in the plaintext must not be available for analysis in the resulting ciphertext. The ciphertext output must appear to be as close to a random set of numbers as possible, no matter what the input is.

The second is the substitution block should be as large as is practically possible. The more random and larger the substitution block is, the harder it is to work back from the plaintext and ciphertext to discovering the substitution pattern being used. To perform a brute-force attack against a substitution using a 128-bit block size, the attacker must correlate as many of the 2^{128} entries in the plaintext block with the 2^{128} entries in the ciphertext substitution block—if the information only uses a small (or sparse) set of possible entries from the original 128-bit space, there is little practical way to make the correlation fast enough to make this sort of attack practical—given the encrypting sender changes its key often enough.

Note

There is a law of diminishing returns when it comes to the size of the block; at some point, increasing the block size does not increase the effectiveness of the cipher at hiding information.

Density is best explained with an example. Assume you are using a straight substitution cipher in the English language, where each letter is replaced by the letter offset by four steps in the alphabet. In this sort of (trivial) cipher:

- Each A would be replaced by an E.
- Each B would be replaced by an F.
- Each C would be replaced by a G.
- Etc.

Now try encrypting two different sentences using this transform:

- THE SKY IS BLUE == XLI WOC MW FPYI
- THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG == XLI UYMGO FVSAR JSB NYQTIH SZIV XLI PEDC HSK

For the attacker trying to figure out how the ciphertext version of the sentence relates to the plaintext version, the first sentence presents 9 matching pairs of letters out of the space of 26 possible letters. There is a good chance you can guess what the correct transform is—move four steps to the right—from this small sample, but it is possible there is some "trick" involved that causes future messages encrypted using this transform to fail to be unencrypted correctly. The second sentence is, however, a well-known example of a sentence containing every possible letter in the English alphabet. The transform can be validated against every possible value in the entire input and output range, making the discovery of the transform trivial.

In this example, the first sentence would be less dense than the second. In real cryptographic systems, the general idea would be to use just several thousand possible symbols out of a space of 2^{128} or 2^{512} possible symbols, which creates a much less dense information set to work with. At some point, the density becomes low enough, the transform complex enough, and the ciphertext random enough, that there is no practical way to compute the relationship between the input (the plaintext) and the output (the ciphertext).

In real life, the substitution blocks are not precomputed in this way. Rather, a cryptographic function is used to calculate the substitution value in real time. These cryptographic functions take a block-sized input, the plaintext, perform the transform, and output the correct ciphertext. The key is a second input that modifies the output of the transform so each key causes the transform to produce a different output. If the key size is 128 bits, and the block size is 256 bits, there are $2^{128} \times 2^{256}$ possible output combinations from the transform. Figure 10-2 illustrates.

In Figure 10-2, each substitution table is the block size; if the block size is 256 bits, then there are 2^{256} possible substitutions in each table. Each key generates a new table, so if the key is 128 bits, then there are 2^{128} possible tables. There are two general ways to attack such an encryption system.

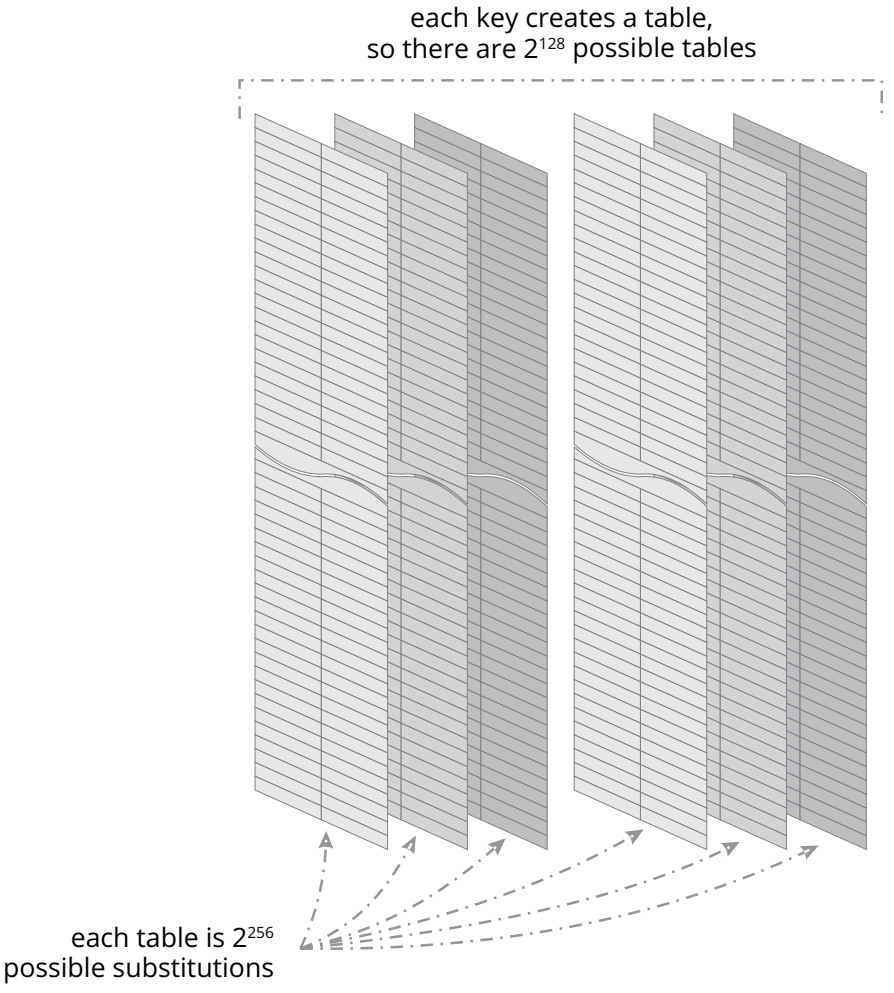


Figure 10-2 *Substitution Tables Generated by Large Key Transforms*

The first way to attack this type of encryption system is to try to map every possible input value to every possible output value, revealing the entire substitution table. If the input only ever represents a small set of the possible inputs (the table is sparsely used, or is a sparse array, more precisely), this task is nearly impossible. If the user changes her key, and hence the particular table among the possible set of tables, often enough, there is no way to perform this mapping faster than the key is changed.

Note

There are still potential weaknesses even in large blocks combined with transforms to produce nearly random output—in other words, even if the transform is close to ideal. If you collect 23 people in a single room, there is a high probability two of them will have the same birthday—but this seems irrational because there are 365 potential days (not counting leap years) on which a person could be born each year. The reason for the disparity between what appears should happen and what does happen is this: in the real world, people's birthdays are clustered on a very small number of days throughout the year. The input data, then, is a very dense "spot" in a moderately large set of possible values. When this happens, the sparseness of the data can work *against* the encryption system. If a small set of data is repeated in the larger set on a regular basis, the attacker can focus on just the substitutions used most often and potentially discover the contents of *enough* of the message to make recovery of most of the meaning reasonably possible.

The second way to attack an encryption system of this kind is to attack the transform itself—the cryptographic function. Remember these large substitution tables are often impossible to generate, store, and transport, so some form of cryptographic function is used to take a block of plaintext as an input and generate a block of ciphertext as the output. If you could discover this transform function, then you can calculate the output in the same way the transmitter and receiver are, and unencrypt the plaintext in real time.

In the real world, this problem is made more complex by

- Kerckhoffs' principle, which states *the transform itself must not be a secret*. Rather *only the key used to select which table among the possible tables should be kept secret*.
- At least some plaintext and ciphertext can sometimes be recovered from an ongoing encrypted data transmission for various reasons—perhaps a mistake, or perhaps the point of the encryption is to verify the text, rather than keeping the text from being read.

Given these restrictions, there are several key points to consider:

- The difficulty of computing the key from the plaintext, ciphertext, and cryptographic function (transform) must be very high.

- The randomness of the output of the cryptographic function must be very high, to reduce the possibility of brute-force attacks—just trying every possible key in the space—being successful.
- The key space must be large, again to prevent brute-force attacks from being successful.

The quality of a cryptographic function is determined by the ability of the function to produce as close to a random output from virtually any input in a way so an attacker is prevented from discovering which key is being used, even though they have both the plaintext and the ciphertext. Cryptographic functions, then, normally use some form of one of the most difficult problems to calculate. One in particular that is often used is computing the factors of very large prime numbers.

Security and Obscurity

No security through obscurity. If you get close enough to a security engineer for long enough, or involved in any sort of debate over proper security, you will likely hear these words somewhere along the way. A more formal name for this phrase is Kerckhoffs' principle, which states: The security of the information carried in an encrypted block or stream should rely on the secrecy of the private key, not the secrecy of the algorithm.

There is one problem with this phrase, however: it is often used out of context. To understand the real meaning of the phrase, you need to go back in time to the origin of encryption algorithms. In the physical lock world, revealing the plans of a lock will often reveal various passageways to bypassing or defeating the lock. This habit was carried over to early software security vendors; the reasoning is if an attacker knows how the encryption algorithm works, he will be able to find ways to defeat the encryption algorithm.

But encryption algorithms are not door locks; what is an important safeguard in one realm can be a dangerous crutch in another. Hiding code developed to encrypt plaintext does not really make the code more secure; in fact, just the opposite happens. Instead of improving security, obscuring encryption code and processes simply prevents experts in the field from finding flaws and possible ways to defeat the code before these are exposed in real deployments. Ultimately, security by obscurity is dangerous in this particular context.

So this is a good principle, but it can be misapplied. For instance, if a network operator attempts to hide internal network architecture, addressing, or even blocking external hosts from reaching internal ones, at least some security experts will counter with "this is simply security by obscurity; you should not do that." Taken in this sense, however, *encrypting data is also security by obscurity.* Hiding information and hiding information about your infrastructure are both essentially hiding information, and hiding information is essentially a form of obscurity.

How can you tell when you should apply "no security by obscurity," and when you should not? Perhaps the best rule of thumb is this: hiding processes, algorithms, and implementations is not a useful addition to security in the cyber world. Hiding information, however, often is. It can be hard to apply this rule of thumb in many situations, but it should be a good start in thinking through the issues and making the right decision in each particular case.

What happens if you are using a 128-bit block, and you have 56 bits of data to transport? The most natural thing to do in this situation would be to pad the plaintext with some number; most likely all 0s or all 1s. The quality of the output is dependent, to some degree, on the sparseness of the input; the fewer the range of numbers used as an input, the more predictable the output of the cryptographic function will be. In this case, it is important to use padding that is *as close to random as possible;* there is an entire field of study around how to pad blocks of plaintext to "help" the cryptographic function produce ciphertext that is as close to random as possible.

Multiple Rounds of Encryption

It is possible to process the same information through a cryptographic function multiple times. For instance, if you have a 128-bit block and a 128-bit key, you can

- Take the plaintext and, using the key, calculate a ciphertext; call this *ct1*.
- Take *ct1* and, using the key, calculate a second-round ciphertext; call this *ct2*.
- Take *ct2* and, using the key, calculate a third-round ciphertext; call this *ct3*.

The actual transmitted ciphertext would be the final *ct3*. What does this process accomplish? Remember the quality of the encryption process is related to the randomness of the output against the input. Each round will, in many situations, increase the randomness just a bit more. There is a point of diminishing returns in

this process; normally after the third round, the data is not going to become "any more random," and hence more rounds are essentially just wasting processing power and time for very little gain.

Public versus Private Key Cryptography

There is a class of cryptographic functions that can transform the plaintext into ciphertext, and back, using two different keys. This capability is useful when you want to be able to encrypt a block of data with one key and allow someone else to unencrypt the data using a different key. The key you keep secret is called the *private key*, and the key you give to others, or publish, is called the *public key*.

To prove you are the actual sender of a particular file, for instance, you can encrypt the file with your private key. Now anyone with your public key can unencrypt the file, which could only have been sent by you. You would not normally encrypt the entire block of data with your private key (in fact most systems using key pairs are designed so you *cannot* do this); rather a signature is created using your private key that can be verified using your public key. To ensure only the person you want to read something can, you can encrypt some data with her public key, publish it, and only the person with the correct private key can unencrypt it.

Such systems are called public key cryptography (sometimes the names engineers choose are, perhaps, a little *too* obvious), or asymmetric cryptography. In public key cryptography, the public key is often "released into the wild;" it is something anyone with access to a key server or some other source can look up.

The alternative to public key cryptography is symmetric key cryptography. In symmetric key cryptography, the sender and receiver share a single key that is used to both encrypt and unencrypt the data (the shared secret). Given shared secrets are (obviously) difficult to create and use, why is symmetric key cryptography ever used? There are two basic tradeoffs to consider when choosing between symmetric and public/private key cryptography:

- **Processing complexity:** Public key cryptography systems generally require a good deal more processing power to encrypt and unencrypt the transmitted data. Symmetric key systems are generally much easier to develop and deploy in a way that does not require large amounts of processing power and time. Because of this, public key cryptography is often used to encrypt very small amounts of data, such as a private key (see the example in the following section).
- **Security:** Public key cryptography generally requires a somewhat unique set of mathematical transform mechanisms. Symmetrically keyed systems tend to have a wider range of available transforms that are also more complex and hence more secure (they provide more randomness in the output and hence are harder to break).

There is a place for both kinds of systems, given these tradeoffs and real-world requirements.

Key Exchange

Some of the earliest cryptographic systems involved wrapping paper around a cylinder of a specific size; the cylinder had to be somehow carried between the two parties to the encrypted communication without being captured by an enemy. In more recent years, pads of keys were physically carried between the two end points of an encrypted system. Some of these were arranged so a particular page would be used for a certain time period and then ripped out, securely destroyed, and replaced by a new page for the next day. Others were designed so each page in the pad would be used to encrypt one message, at which point the page would be ripped out and replaced—a one-time pad.

> **Note**
>
> The concept of a one-time pad has been carried into the modern world with authentication systems that allow the user to create a code that is used once, and then discarded, to be replaced by a new code the next time the user tries to authenticate. Any system that relies on a code that is used once is still called a one-time pad.

In the modern world, there are other ways you can exchange cryptographic material, whether it is using a shared secret key or retrieving a private key.

Many times in cryptography, it is easier to explain how something works using trivial examples. In the following explanations, Fish and Jeff will be two users who are trying to exchange secure information, with Fish being the initiator and sender, and Jeff being the receiver.

Exchanging Public Keys

Fish would like to send a message to Jeff in a way that only Jeff can read it; to do this, she needs Jeff's public key (remember she should not have access to Jeff's private key). Where can she get this information? She could

- Ask Jeff for it directly. This might seem simple to do, but it could be very difficult in real life. How, for instance, can she be certain she is actually communicating with Jeff?
- Look up Jeff's public key in a public database of keys (a key server). Again, this seems to be straightforward, but how does she know she has found the right person, or someone has not placed a false key for Jeff on this particular server?

These two problems can be solved through some sort of reputation system. For instance, in the case of a public key, Jeff could ask several of his friends, who know him well, to sign his public key using their private keys. Their signature on his public key essentially says, "I know Jeff, and I know this is his public key." Fish can examine this list of friends to determine if there are any of them she can trust. Based on this examination, Fish can determine she either trusts that this specific key is Jeff's key, or she does not.

In this situation, it is up to Fish to determine how much, and what sort of, proof she will accept. Should she, for instance, accept that the key she has is actually Jeff's because

- She directly knows one of Jeff's friends and trusts this third person to tell her the truth.
- She knows someone who knows one of Jeff's friends, and trusts this friend of hers to tell her the truth about Jeff's friend, and hence trusts Jeff's friend to tell the truth about Jeff and his key.
- She knows several people who know several of Jeff's friends and makes a decision to trust this is Jeff's key based on the testimony of several people.

This kind of system is called a web of trust. The general idea is that trust has different levels of transitivity. The concept of transitive trust is somewhat controversial, but the idea behind a web of trust is *if you receive enough evidence, you can build up a trust in a person/key pairing.* An example of this kind of web of trust is the Pretty Good Privacy ecosystem, where people meet at conferences to cross sign one another's keys, building up a web of transitive trust relationships that can be relied on when their communication moves into the electronic-only realm.

Another option is the key server owner could somehow do an investigation of Jeff and determine if he really is who he says he is, and whether or not this is really his key. The clearest "real-world" example of this sort of solution is a *public notary.* If you sign a document in front of a notary, he checks for some form of identification (verifying who you are) and then watches you physically sign the document (verifying your key).

This kind of validation is called a central source of trust (or similar—though it almost always has the word *centralized* in it) or a Public Key Infrastructure (PKI). The solution depends on Fish trusting the process and honesty of the centralized key repository.

Exchanging Private Keys

Given symmetric key cryptography is so much faster to process than public key cryptography, you would ideally like to encrypt any long-standing or high-volume flows using a symmetric shared secret key. But, short of somehow physically exchanging keys, how is it possible to exchange a single private key between two devices that are connected over a network? Figure 10-3 is used to illustrate.

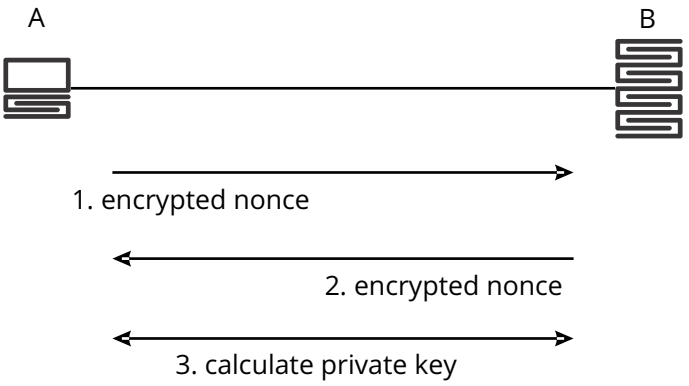


Figure 10-3 *Using Public Keys to Either Exchange or Calculate a Private Session Key*

In Figure 10-3:

1. Assume A begins the process. A will encrypt a nonce, a random number used once in the process and then thrown away (a nonce is a form of a one-time pad, in effect), using B's public key. Because the nonce has been encrypted with B's public key, in theory only B can unencrypt the nonce, as only B should know B's private key.
2. B, on unencrypting the nonce, will now send some new nonce to A. This may include A's original nonce, or A's original nonce plus some other information. The point is that A must know, for certain, that the original message including A's nonce was received by B—and not some other system acting as B. This is ensured by B including some piece of information that was encrypted using its public key, as B is the only system that could have unencrypted it.
3. A and B, using the nonces and other information exchanged to this point, will calculate a private key, which is then used to encrypt/unencrypt information transferred between the two systems.

The steps outlined here are somewhat naïve; there are better, more secure systems, such as the Internet Key Exchange (IKE) protocol; see the "Further Reading" section at the end of the chapter for resources in this area.

Cryptographic Hashes

Assume you wanted to send a large text file, or even an image, and allow receivers to validate it originated from you. What if the data in question is *very* large? Or what if

the data needs to be compressed to be transmitted effectively? There is a natural conflict between cryptographic algorithms and compression; cryptographic algorithms attempt to produce maximally random output, and compression algorithms attempt to take advantage of nonrandomness in the data to compress data into a smaller number of bits. Or perhaps you want the information to be read by anyone who would like to read it, which means not encrypting it, but you would like receivers to be able to verify you transmitted it if they would like to.

Cryptographic hashes are designed to provide a solution to resolve these problems. There is a brief explanation of hashes in Chapter 7, "Packet Switching." You might have already noticed at least one similarity between the idea of a hash and a cryptographic algorithm. Specifically, a hash is designed to take a very large piece of data, and create a fixed length representation so there are very few collisions in the output for a wide range of inputs. This is very similar to the concept of *as close to random output for any input* required of a cryptographic algorithm. Another similarity worth mentioning is that hash and cryptographic algorithms both work better with a very sparsely populated input space.

A cryptographic hash simply replaces the normal hash function with a cryptographic function. In this case, the hash can be calculated and either posted alongside the data or transmitted with the data.

Cryptographic hashes can either be used with symmetric or public key systems, but they are normally used with public key systems.

Obscuring User Information

Returning to the chapter introduction, another security problem space is data exhaust. In the case of individual users, data exhaust can be used to trace what users are doing while they are on the network (rather than just processes). For instance:

- If you carry a cell phone with you at all times, it is possible to trace the movement of the Media Access Control (MAC) address as it moves between wireless connection points to trace your physical movements.
- Since most data streams are not symmetrical—data passes through large packets, while acknowledgments are passed through small packets—an observer can discover when you are uploading and downloading data, and perhaps even when you are completing small transactions. Combined with the destination server, this information could reveal a good bit about your behavior as a user in a particular situation, or over time. This, and many other kinds of traffic analysis, can be performed even on encrypted traffic.
- As you move from website to website, an observer can trace how long you spend on each one, what you click on, how you reached the next site, what you

have searched for, what sites you keep open at any time, etc. This information can reveal a good bit about you as a person, what you are trying to accomplish, and other personal factors.

Two solutions of interest in this space are covered in the following sections as examples of the sorts of solutions available: MAC address randomization and onion routing.

MAC Address Randomization

The Institute of Electrical and Electronic Engineers (IEEE) originally designed the MAC-48 address space, described in Chapter 4, "Lower Layer Transports," to be assigned by manufacturers of the network interfaces. These addresses would then be used "as is" by manufacturers of networking equipment, so each piece of hardware would have a fixed, immutable hardware address. This process was designed long before cell phones were even a dream on the horizon and before privacy became an issue.

In the modern world, this means a single device can be followed regardless of where it connects to the network. Many users find this unacceptable, particularly as it is not just the provider who can track this information, but anyone who can listen in on the wireless signal, which means anyone with an antenna. One way to solve this is to allow the device to change its MAC address on a regular basis, even perhaps using a different MAC address in each packet. Since a third party listener, outside the provider network, cannot "guess" the next MAC address any device will use, it cannot track a particular device. A device that uses MAC address randomization will also use a different MAC address on each network it joins, so it will not be trackable across multiple networks.

There are attacks against MAC address randomization, primarily centering around the user's authentication to use the network. Most authentication systems rely on the MAC address, because it is programmed into the device, to identify the device, and in turn, the user. Once the MAC address is no longer an unchanging identifier, there must be some other solution. Places where MAC address randomization can be attacked are

- **Timing:** If a device is going to change its MAC address, it must somehow tell the other end of the wireless link about these changes, so the channel between the connected device and the base station can remain viable. There must be some agreed-on system of timing so the changing MAC address can continue communicating across the change. If an attacker can determine when this change will take place, then she can watch the right window of time and discover the new MAC address the device takes on.

- **Sequence numbers:** As with all transport systems, there must be some way to determine if all the packets have been received or dropped. An attacker can track the sequence numbers being used to track packet delivery and acknowledgment. Combined with the timing attack just noted, this can provide fairly certain identification of a specific device across MAC address changes.
- **Information element fingerprints:** Each mobile device has a set of capabilities it can support, such as installed browsers, extensions, apps, and additional hardware. Because each user is unique, the set of applications he uses will also likely be fairly unique, creating a fingerprint of capabilities that will be reported through the information element in response to probes from the base station.
- **Service Set Identifier (SSID) fingerprints:** Each device keeps a list of networks it can currently reach and (potentially) networks it could reach at some point in the past. This list is likely to be fairly unique, and hence can act as a device identifier.

While each of these items may provide some level of uniqueness at a device level, the combination of these items can come very close to identifying a specific device often enough to be practically useful in tracking any specific user connecting to a wireless network.

This does not mean MAC address randomization is useless, but rather this is one step in preserving user privacy when connected to a wireless network.

Onion Routing

Onion routing is a mechanism used to disguise the path of, as well as encrypt, user traffic passing through a network. Figure 10-4 is used to illustrate.

In Figure 10-4, host A wants to send some traffic to K securely, without any other node in the network being able to see the connection between the host and the server, and without any observer being able to see the plaintext. To accomplish this with onion routing, A does the following:

1. It uses a service to find a set of nodes that can interconnect and provide a path to the server, K. Assume this set of nodes includes [B,D,G]; while the illustration shows these as routers, they are more likely software routers running on hosts, rather than dedicated network devices. Host A will first find B's public key and use this information to build a symmetric key encrypted session with B.
2. Once this session is established, A will then find D's public key, and use this information to exchange a set of symmetric keys with D, finally building a session to D using this symmetric secret key to encrypt the secured channel. It is important to note that from D's perspective, this session is with B, rather than A;

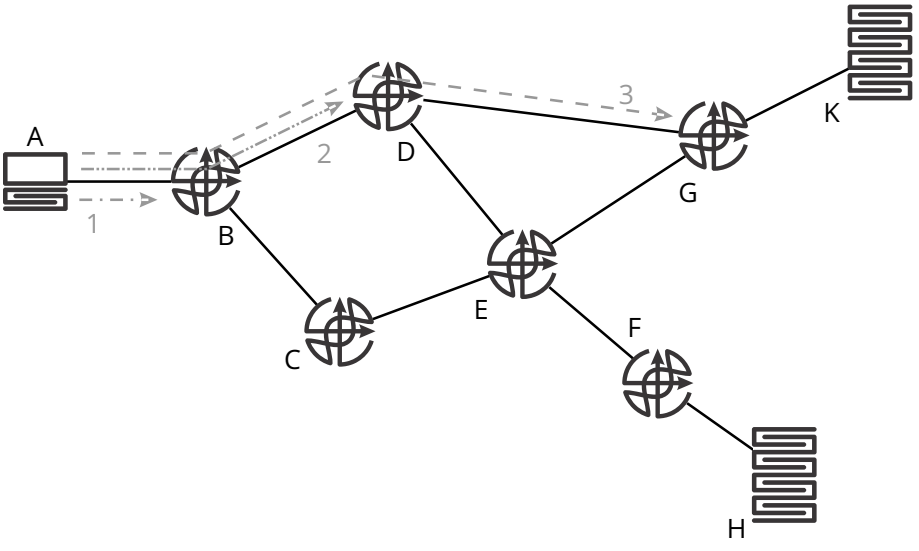


Figure 10-4 *Onion Routing*

host A simply instructs B to take these actions on its behalf, rather than doing them directly. This means that D does not know A is the originator of the traffic; it only knows the traffic is sourced from B and carried across an encrypted link from there.

3. Once this session is established, A will then instruct D to set up a session with G in the same way it instructed B to set up a session with D. D now knows the destination is G but does not know where the traffic will be routed by G.

Host A now has a secure path to K with the following properties:

- The traffic between each pair of nodes along the path is encrypted with a different symmetric private key. An attacker that breaks the connection between one pair of nodes along the path still cannot observe the traffic being transmitted between nodes elsewhere in the path.
- The exit node, which is G, knows the destination but not the source of the traffic.
- The entrance node, which is B, knows the source of the traffic but not the destination.

In this kind of network, only A knows the full path between itself and the destination. The intermediate nodes do not even know how many nodes are in the

path—they know about the previous and next nodes. The primary form of attack against such a system is to take over as many exit nodes as you can, so you can observe the traffic exiting from the entire network, and correlate it back into a full stream of information.

Man in the Middle

Any kind of security should not only examine how you can protect information, but also consider the different ways in which you can cause the protection of data to fail. Given no system is perfect, there will always be some way you can attack the system successfully. If you know the kinds of attacks that can be successfully launched against a transport security system, you can try to design the network and environment in a way that prevents these attacks from being used. Man-in-the-middle (MitM) attacks are common enough that they are worth considering in some detail. Figure 10-5 illustrates.

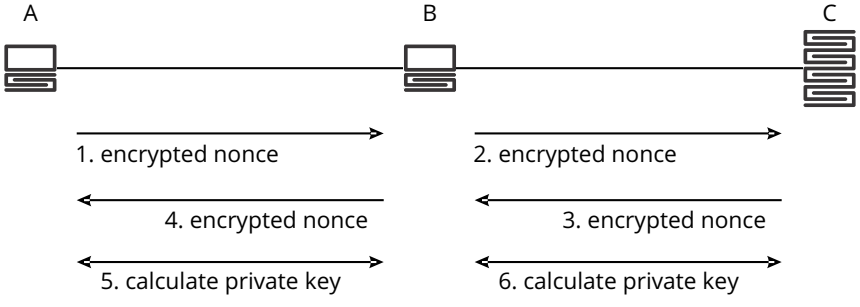


Figure 10-5 *A Man-in-the-Middle Attack*

Figure 10-5 is similar to Figure 10-3 with one addition: there is a host, B, situated between the host A and the server C that would like to start an encrypted session. By some means, either spoofing C's IP address, or modifying the Domain Name Service (DNS) records so C's name resolves to B's address, or perhaps even modifying the routing system so traffic that should be delivered to C is delivered to B instead, the attacker has caused B to receive traffic originating at A and destined to C. In Figure 10-5:

1. Host A sends a semirandom number, called a nonce, to C. This information is received by B.

2. Host B, which the attacker is using as the MitM, transmits this nonce on to C in a way that makes it appear the packet actually originated at A. At this point, the attacker knows the nonce encrypted by A; the attacker does not know A's private key but does have access to anything A sends encrypted with A's private key.
3. The server, C, sends a response with an encrypted nonce, as well. B receives this and records it.
4. Host B passes the nonce it received from C on to A. Host A will still believe this packet came directly from C.
5. Host B calculates a private key with A as if it were C.
6. Host B calculates a private key with C as if it were A.

Any traffic A sends to C will be received by B, which will

- Unencrypt the data A has transmitted using the private key calculated at step 5 in Figure 10-5.
- Encrypt the data A has transmitted using the private key calculated at step 6 in Figure 10-5 and transmit it to C.

During this process, the attacker, at B, has access to the entire flow, in plaintext, between A and C. Neither A nor C realizes they have both built an encrypted session to B, rather than to one another. These kinds of MitM attacks are very difficult to prevent and detect.

Transport Layer Security

Transport Layer Security (TLS), also known as the *Secure Socket Layer* (SSL), is a secure transport layer protocol deployed by default in most web browsers. When users see the small green lock indicating that a website is "safe," this means the SSL certificate is valid, and the traffic between the host (on which the browser runs) and the server (on which the web server runs) is being encrypted. TLS is a complex protocol with a lot of different options; this section will provide a rough overview of its operation. Figure 10-6 illustrates the components of the TLS suite.

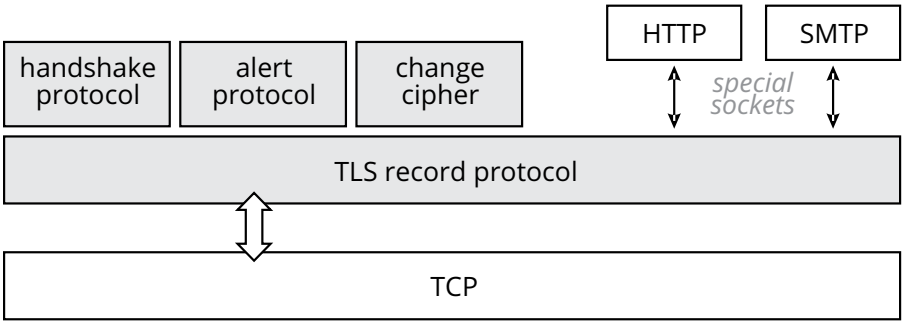


Figure 10-6 *TLS Components*

In Figure 10-6:

- The **handshake protocol** is responsible for initializing sessions and setting up session parameters, including the initial private key exchange.
- The **alert protocol** is responsible for error handling.
- The **change cipher specification** is responsible for starting the encryption.
- The **record protocol** breaks data blocks presented for transport into fragments, (optionally) compresses the data, adds a Message Authentication Code (MAC), encrypts the data using the symmetrical key, adds the original information to the block, and then sends the block to the Transmission Control Protocol (TCP) for transport across the network.

Applications running on top of TLS use a special port number to access the service through TLS. For instance, web services using the Hypertext Transfer Protocol (HTTP) are normally accessible over TCP port 80; TLS-encrypted HTTP is normally accessible through port 443. While the service is the same, the change in the port number allows the TCP process to direct traffic that needs to be unencrypted for the final application to read it.

The MAC, which within this context will mean a Message Authentication Code, is used to ensure the sender is authenticated. While some cryptography systems assume that successfully encrypting data with a key the receiver knows proves the sender is truly who he claims to be, TLS does not. Instead, TLS includes a MAC that validates the sender separately from the keys used to encrypt messages on the wire. This helps prevent MitM attacks against TLS-encrypted data streams.

Figure 10-7 shows the TLS startup handshake, which is managed by the handshake protocol.

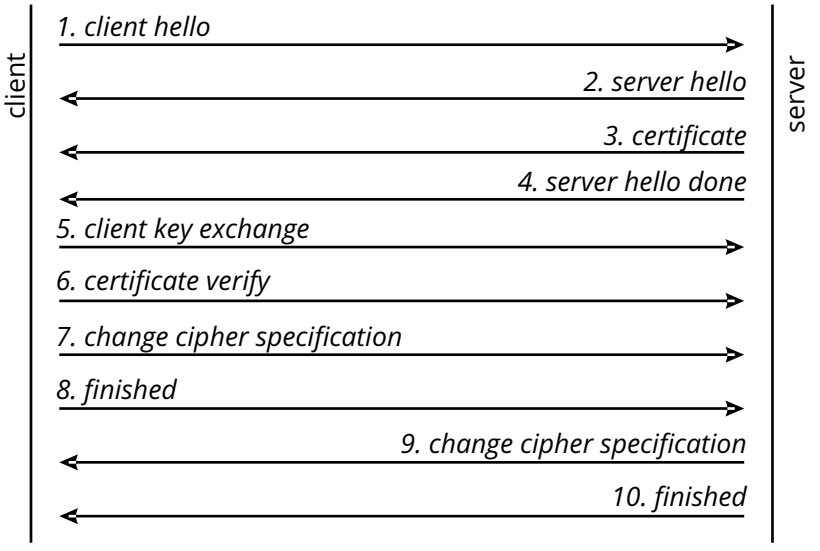


Figure 10-7 *TLS Secure Session Startup Process (Handshake)*

In Figure 10-7:

1. The client hello is sent in plaintext, and contains information about the version of TLS the client is running, 32 random octets (the nonce), a session identifier (which allows a previous session to be recovered or restored), a list of the encryption algorithms (cipher suites) the client supports, and a list of the data compression algorithms the client supports.
2. The server hello is sent in plaintext, as well, and contains the same information as above, from the server's perspective. In the server hello, the encryption algorithm field indicates the kind of encryption that will be used for this session. This is normally the "best" encryption algorithm available at both the client and the server (although it is not always the "best").
3. The server sends its public key (a certificate), along with the nonce that the client sent to the server, where the nonce is now encrypted using the server's private key.
4. The *server hello done* message indicates the client now has the information it needs to complete the session setup.
5. The client generates a private key and uses the server's public key to encrypt it. This is transmitted in the *client key exchange* message toward the server.

6. Once this has been transmitted, the client must sign *something* known to both the server and the client in order to verify the sender is the correct device. Usually, the signature is across all the messages in the exchange up to this point; generally, a cryptographic hash is used to generate a verification.
7. The *change cipher specification* message essentially acknowledges the session is up and running.
8. The *finished* message once again authenticates all the previous handshake messages to this point.
9. The server then acknowledges the encryption session is set up by sending a *change cipher specification* message.
10. The server then sends a *finished* message, which authenticates the prior messages sent in the handshake in the same way as above.

Note

Optional steps in the TLS handshake have been left out of this explanation for clarity.

Once the session is up and running, applications can send information toward the receiving host on the correct port number. This data will be encrypted using the previously negotiated private key and then handed off to TCP for delivery.

Final Thoughts on Transport Security

This chapter has considered three specific problems in the space of transport security: validating data, protecting data from being examined, and protecting user privacy. For network engineers, understanding the theory of how transport security works and where the weak spots in a transport security system interact with the network design is often more important than understanding the intimate details of the actual security mechanisms themselves. Because of this, this chapter has focused on providing a stronger theoretical foundation in the form of "how to think about transport security," rather than on practical implementations of transport security. Readers who are interested in a deeper exploration of transport security are encouraged to look at the "Further Reading" section at the end of this chapter.

Overall, transport security is just one small piece of the overall security required in network engineering; Chapter 21, "Security: A Broader Sweep," considers a broader sweep of security topics at both a network and a system level.

Further Reading

Bauer, Kevin, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. "Low-Resource Routing Attacks Against Tor." In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, 11–20. WPES '07. New York, NY, USA: ACM, 2007. doi:10.1145/1314333.1314336.

Brockners, Frank, Shwetha Bhandari, Sashank Dara, Carlos Pignataro, John Leddy, Stephen Youell, David Mozes, and Tal Mizrahi. "Proof of Transit." Internet-Draft. Internet Engineering Task Force, March 2017. https://datatracker.ietf.org/doc/html/draft-brockners-proof-of-transit-03.

Davies, Joshua. *Implementing SSL / TLS Using Cryptography and PKI*. 1st edition. Hoboken, NJ: Wiley, 2011.

Ducklin, Paul. "What Your Encrypted Data Says about You." *Naked Security*, March 18, 2016. https://nakedsecurity.sophos.com/2016/03/18/what-your-encrypted-data-says-about-you/.

Ferguson, Niels, and Bruce Schneier. *Practical Cryptography*. 1st edition. New York: Wiley, 2003.

Ferguson, Niels, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. 1st edition. Indianapolis, IN: Wiley, 2010.

Katz, Jonathan, and Yehuda Lindell. *Introduction to Modern Cryptography*. 2nd edition. Boca Raton, FL: Chapman and Hall/CRC, 2014.

Kaufman, Charlie, Paul E. Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. *Internet Key Exchange Protocol Version 2 (IKEv2)*. Request for Comments 7296. RFC Editor, 2014. doi:10.17487/RFC7296.

Matte, Célestin, Mathieu Cunche, Franck Rousseau, and Mathy Vanhoef. "Defeating MAC Address Randomization Through Timing Attacks." In *Proceedings of the 9th ACM Conference on Security #38; Privacy in Wireless and Mobile Networks*, 15–20. WiSec '16. New York, NY: ACM, 2016. doi:10.1145/2939918.2939930.

Narayanan, Arvind, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton, NJ: Princeton University Press, 2016.

Paar, Christof, Jan Pelzl, and Bart Preneel. *Understanding Cryptography: A Textbook for Students and Practitioners*. 1st edition. Heidelberg; New York: Springer, 2010.

Piper, Fred, and Sean Murphy. *Cryptography: A Very Short Introduction*. 1st edition. Oxford; New York: Oxford University Press, 2002.

Rescorla, Eric, and Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. Request for Comments 5246. RFC Editor, 2008. doi:10.17487/RFC5246.

Schneier, Bruce. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. 1st edition. Indianapolis, IN: Wiley, 2015.

Shimeall, Tim. "Traffic Analysis for Network Security: Two Approaches for Going Beyond Network Flow Data." *SEI Blog*, September 16, 2016. https://insights.sei.cmu.edu/sei_blog/2016/09/traffic-analysis-for-network-security-two-approaches-for-going-beyond-network-flow-data.html.

Silva, John Edward. "An Overview of Cryptographic Hash Functions and Their Uses." SANS Institute, January 15, 2013. https://www.sans.org/reading-room/whitepapers/vpns/overview-cryptographic-hash-functions-879.

Sobers, Rob. "The Definitive Guide to Cryptographic Hash Functions (Part 1)." *Varonis Blog*, August 2, 2012. https://blog.varonis.com/the-definitive-guide-to-cryptographic-hash-functions-part-1/.

———. "The Definitive Guide to Cryptographic Hash Functions (Part II)." *Varonis Blog*, August 14, 2012. https://blog.varonis.com/the-definitive-guide-to-cryptographic-hash-functions-part-ii/.

Stallings, William. *Cryptography and Network Security: Principles and Practice*. 7th edition. Boston, MA: Pearson, 2016.

Vanhoef, Mathy, Célestin Matte, Mathieu Cunche, Leonardo S. Cardoso, and Frank Piessens. "Why MAC Address Randomization Is Not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms." In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 413–24. ASIA CCS '16. New York, NY: ACM, 2016. doi:10.1145/2897845.2897883.

Review Questions

1. Man-in-the-middle attacks are seen as a major security weakness, but there are many situations in which a system is intentionally placed in the flow of an encrypted stream of data. The system in the middle acts as a proxy, unencrypting and reencrypting the data as it passes through the system. Find at least one

use case for this kind of system, and explain some of the positive and negative aspects of such a system.

2. As an example of data exhaust, research the idea of a web browser fingerprint. Describe the concept, how accurate it is, and what mitigations are available.
3. A hash and a cryptographic algorithm have many similarities and some differences. Describe these similarities and differences.
4. Find at least one cryptographic system that uses multiple rounds of encryption. Why was this number of rounds chosen? Does the encryption system suggest more rounds for higher security, or not?
5. In recent years, the concept of a public notary has become more difficult to design and fulfill. Describe some of the challenges such a system might face and some of the ways in which these challenges might be overcome.
6. Is MAC address randomization implemented differently with IPv4 and IPv6? What are the differences, and why do they exist?
7. Investigate IPsec. How is it different from TLS? At what layer of the protocol stack does it encrypt, and what modes of operation are available?

This page intentionally left blank

Part II

The Control Plane

Building a single packet processing device[md]the router (or layer 3 switch, now commonly just called a switch, to the confusion of just about everyone) being the most common example[md]has been the focus up to this point. Now it is time to begin connecting routers together. Consider the network in Figure P2-1.

An application running on host A needs to obtain some information from a process running on F. Devices B, C, D, and E are, of course, packet processors (routers). To forward packets between hosts A and F, router B is going to be called on to forward packets to F, even though it is not connected to F; likewise, routers C and D are going to need to forward packets to both A and F, even though they are connected to neither of these hosts.

The question posed in this Part II, then, is this:

How do network devices build the tables needed to forward packets along loop-free paths through the network?

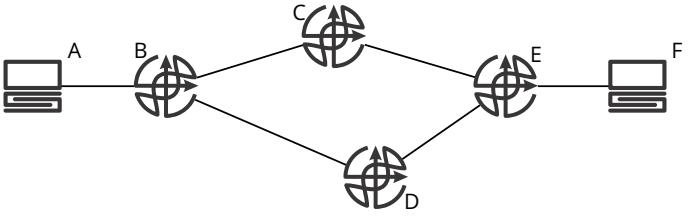


Figure P2-1 *Topology Discovery*

The answer is much more complex than it might immediately appear, for there are actually several problems contained within this one:

- How do devices learn about the topology of the network[md]which links are connected to what devices and destinations?
- How do control planes take this information and build loop-free paths through the network?
- How do control planes detect and react to changes in the network?
- How are control planes scaled to meet the needs of large scale networks?
- What policies are implemented in the control plane, and how?

Each chapter in this part addresses one or more of the sub-problems of the larger question asked in the preceding list. Two chapters are also dedicated to examples of control planes, to show how the problems and solutions have been implemented by widely deployed protocols. The chapters in Part II include:

- **Chapter 11: Topology Discovery**, which considers how a control plane discovers the network topology and reachability information
- **Chapters 12 and 13: Unicast Loop-Free Paths**, which consider the problem of calculating a set of loop-free paths through the network, and the widely deployed solutions to this set of problems
- **Chapter 14: Reacting to Topology Changes**, which considers the options a control plane has to react to a change in the network topology
- **Chapter 15: Distance Vector Control Planes**, which considers control planes based on Bellman-Ford and the Diffusing Update Algorithm
- **Chapter 16: Link State and Path Vector Control Planes**, which considers routing protocols based on Dijkstra's shortest path first algorithm, and routing protocols that keep a list of path elements through which a routing update has passed
- **Chapter 17: Policy in the Control Plane**, which considers what problems policy needs to solve in the control plane, and a range of solutions for those problems

- **Chapter 18: Centralized Control Planes**, which considers Software Defined Networks, Programmable Networks, and other control planes that centralize all or some of the policy or the calculation of loop-free paths
- **Chapter 19: Failure Domains and Information Hiding**, which considers route filtering, aggregation, summarization, and other forms of routing protocol policy
- **Chapter 20: Examples of Information Hiding**, which considers flooding domain implementation in link state protocols and route aggregation in the Border Gateway Protocol

This page intentionally left blank

Chapter 11

Topology Discovery

Learning Objectives

After reading this chapter, you should understand:

- The basic terms *node* or *vertex* and *edge* as used in graph theory, and how they relate to network devices
- What a reachable destination is
- The two basic things a control plane must discover about a network
- How network devices discover one another and detect the lack of two-way connectivity
- How the MTU can be discovered through the network
- The difference between proactive control planes with reactive reachability learning, proactive control planes, and reactive control planes
- The problems that need to be solved in redistributing information between control planes.

Network diagrams typically show just a few types of devices, including routers, switches, systems connected to the network (generally speaking, various sorts of hosts), and various sorts of appliances (such as firewalls). These are often interconnected with links, represented as lines. An example is provided in Figure 11-1.

Network diagrams, like many forms of abstraction, hide a lot of information to make the information included more accessible. First, network diagrams tend to be

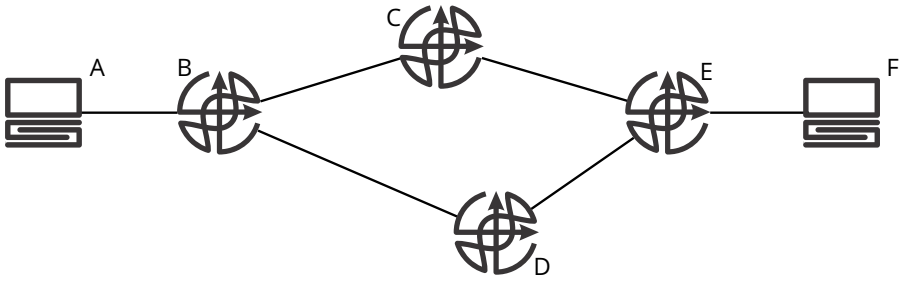


Figure 11-1 *Topology Discovery Example*

somewhere between logical and physical representations of the network. Such diagrams normally do not show every physical connection in the network; for instance, a network diagram may show a bundle of links as a single link, or a single physical wire that has been multiplexed as several logical links (such as Ethernet, or some other broadcast link, which is a single physical channel used by multiple devices to communicate).

Note

There is often some confusion about the term *multiplexing* in network engineering. Many engineers tend to think of sharing two virtual links (see Chapter 9, "Network Virtualization") as the only form of network multiplexing. However, *any time* there are multiple devices sharing a single link, a situation ultimately requiring some form of addressing, time-based division of traffic, or frequency-based division of traffic, multiplexing is being used. Virtualization can be seen as a second layer of multiplexing, or multiplexing on top of multiplexing.

Definition by Platypus Considered Harmful

It is often tempting to try to make definitions precise in all situations. As tempting as this is, however, you should resist it. Imagine, for a moment, the first human on the face of the Earth wakes up on the first day of her existence and discovers a platypus. As the platypus lays eggs, has fur, and is warm blooded like a mammal and has a bill like a bird, it would be tempting for this first human to define the rest of the animals based on the platypus.

Starting with the platypus is going to cause a problem: it is going to stretch the original set of classifiers out of shape enough to make further classification almost impossible.

Hence, it is important not to let the platypus determine your definitions. Rather, start from the most common cases and allow the edges to be somewhat soft. Sometimes devices must be able to shift roles when the context shifts, and sometimes ideas, technologies, and devices just aren't going to fit into a neat category. It is better to allow the categories to stand for the general case, and note the exceptions, than it is to end up with convoluted definitions and classifiers that ultimately are ineffective at their primary job, which is to describe things.

There is a balance here, of course—it may be that a definition becomes "not useful" over time, as a greater number of exceptions are discovered to the rule. On the other hand, it is almost never useful to discard a definition for a single exception.

There are many cases of this problem in network engineering; you will encounter two early in this chapter. The first of these is *multiplexing*, as noted in the previous note; the second is *node,* which is defined in the text following this sidebar.

Second, network diagrams often leave out the logical complexity of services. The control plane, however, cannot mask these sorts of complexities out.

Instead, the control plane must gather information about the network locally and from other control planes, advertise it to other devices running the control plane, and build a set of tables the data plane can use to forward traffic across each device in the network, from source to destination. This chapter is going to consider the problem:

How does the control plane learn about the network?

This question can be broken down into multiple parts:

- What is the control plane trying to learn about? Or perhaps, what are the components of a network topology?
- How does the control plane learn about devices connected to the network?
- What are the basic classifications used in describing the advertisement of information about the network?

> **Note**
>
> The mechanisms used to carry information about the network are not considered in this chapter, as they are typically intimately tied to the way in which the set of loop-free paths is calculated.

Nodes, Edges, and Reachable Destinations

The first problem to solve is really a meta-question: what kinds of information does a control plane need to learn and distribute in order to build loop-free paths through a network? A word of warning about the following section, however: Networking terms are difficult to nail down, as individual terms are often used to describe a variety of "things" in the network, depending on the context in which they are used.

Node

A node either processes packets (including forwarding packets), sends packets, or receives packets in a network. The term is taken from graph theory, where they can also be called vertices, although this term is more loosely applied in network engineering. There are several kinds of nodes in a network, including

- **Transit node:** Any device that is designed to accept packets on one interface, process them in some way, and send them on another interface. Examples of transit nodes are routers and switches; they are often just called *nodes,* as they will be here, rather than *transit nodes.*
- **Leaf node:** Also called an end system or host; any device designed to run applications that generate and/or accept packets from one or more interfaces. These are network *sources* and *sinks;* most often these nodes are actually called *hosts,* rather than *leaf nodes,* to differentiate them from the shorthand *nodes,* which typically means a *transit node.*

There are many readily apparent holes in these two definitions. What should a device be called that accepts a packet on one interface, terminates the connection in a local process or application, generates a new packet, and then transmits that new packet out of a different interface? The problem becomes more difficult if the

information contained in the two packets is roughly the same, as in the case of a *proxy server,* or some other similar device. In these cases, it is useful to classify the device as either a leaf or a node within a specific context, *depending on the role it is taking in relation to other devices within the context*. To give an example, from the perspective of a host, a proxy server acts as a network forwarding device, as the operation of the proxy server is (somewhat) transparent to the host. From the perspective of an adjacent node, however, proxy servers are hosts, as they terminate traffic streams, and (generally) participate in the control plane the same way a host would.

Edge

An edge is any connection between two network devices across which packets are forwarded. The nominal case is a point-to-point link connecting two routers—but this is not the only case. In graph theory, an edge *only* connects precisely two nodes. In network engineering, there are the notions of multiplexed, multipoint, and other kinds of multiplexed links. These are most often modeled as a set of point-to-point links, particularly when building a set of loop-free paths through the network. In network diagrams, however, multiplexed links are often drawn as a single link with multiple nodes attached.

Reachable Destination

A reachable destination can describe a single host or service, or a set of hosts or services, reachable through the network. The nominal example of a reachable destination is either a host or a set of hosts on a *subnet,* but it is important to remember the term can also describe a *service* in some contexts, such as a particular process running on a single device, or many copies of a service available on a number of devices. Figure 11-2 illustrates.

In the network illustrated in Figure 11-2, reachable destinations may include

- Any of the individual hosts, such as A, D, F, G, and H
- Any of the individual nodes, such as B, C, or E
- A service or process running on a single host, such as S2
- A service or process running on multiple hosts, such as S1
- A set of devices attached to a single physical link, or edge, such as F, G, and H

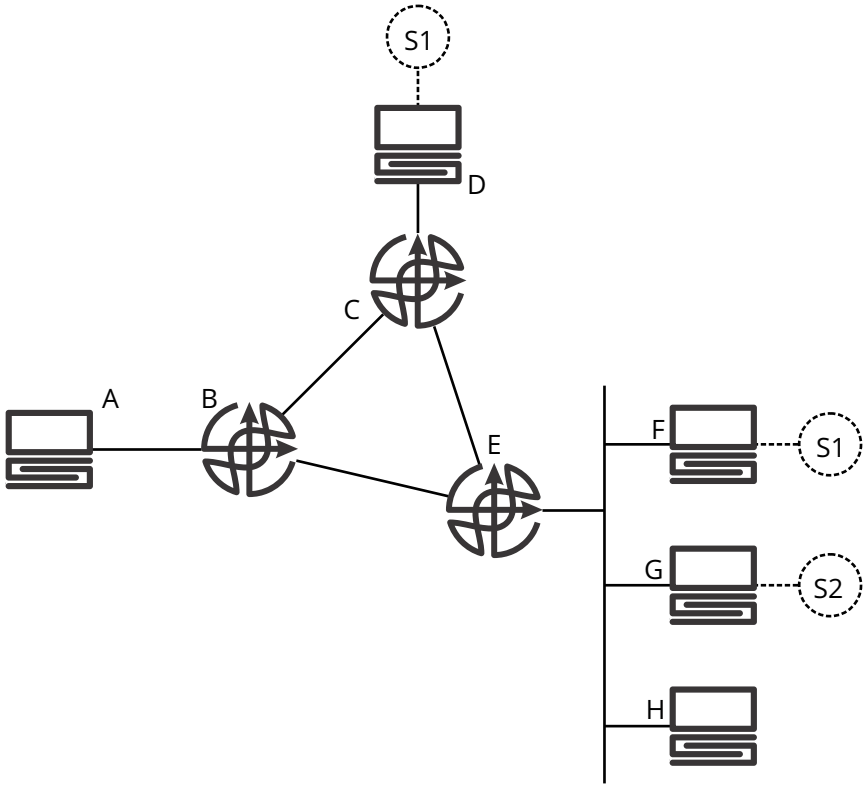


Figure 11-2 *An Illustration of Reachable Destinations*

This last reachable destination is also represented as an *interface* onto a particular link or edge in the network. Hence, router E could have a number of reachable destinations, including

- The interface onto the link connecting router E to C
- The interface onto the link connecting router E to B
- The interface onto the link connecting router E to the hosts F, G, and H
- The network representing reachability to the hosts F, G, and H
- Any number of internal services that might be advertised as individual addresses, ports, or protocol numbers
- Any number of internal addresses attached to virtual links that do not exist in the physical network, but might be used to represent internal state within the device (not shown in Figure 11-2)

The concept of a *reachable destination*, then, can mean a lot of different things depending on the context. In most networks, a reachable destination is either a single host, a single link (and the hosts attached to the link), or a set of links (and the hosts attached to those links) *aggregated* into a single reachable destination.

> **Note**
>
> An example of reachable destinations being aggregated is provided in Chapter 5, "Higher Layer Data Transports." Using a shorter prefix length IP address to represent a set of longer prefix subnets is a form of aggregation.

Topology

The topology is *the set of links (or edges) and nodes that describe the entire network*. Normally, the topology is described and drawn as a graph, but it can also be represented in a data structure designed to be consumed by machines, or a tree, which is normally designed to be consumed by humans.

Topological information can be summarized by simply making destinations that are physically (or virtually) connected several hops away appear to be directly attached to a local node, and then removing the information about the links and nodes in any routing information carried in the control plane from the point of summarization. Figure 11-3 illustrates this concept

Learning about the Topology

It would seem simple enough to learn about the network topology: examine the attached links. What appears simple in networks, however, often turns out to be complex. Examining the local interface can tell you about the *link*, but not about other network devices attached to the link. Further, even if you can detect another network device running the same control plane on a particular link, this does not mean the other device can detect *you*. There are, then, several issues to explore.

Detecting Other Network Devices

Given routers A, B, and C are attached to a single link, as illustrated in Figure 11-4, what mechanisms can they use to detect one another, as well as exchange information about their capabilities?

The first point to note about the network shown on the left side of Figure 11-4 is *the interfaces do not correspond to neighbors*. The actual neighbor relationships

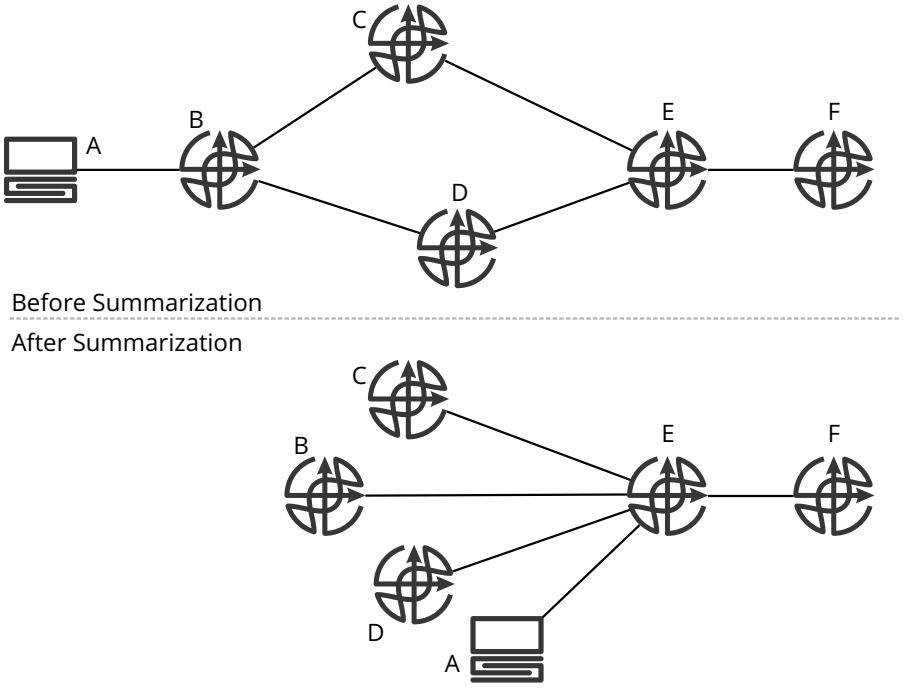


Figure 11-3 *Summarization of Topology Information in the Control Plane*

are shown on the right side of Figure 11-4. Each router in this network has two neighbors, but only one interface. This illustrates the point that the control plane cannot use interface information to discover neighbors; there must be some other mechanism the control plane can use to find neighbors.

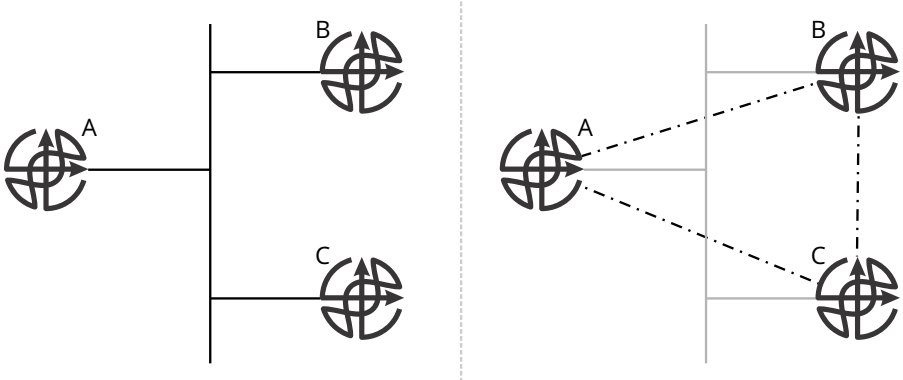


Figure 11-4 *Network Device Neighbor Discovery in the Control Plane*

Manual configuration is one widely deployed solution to this problem. Particularly in control planes designed to overlay another control plane, or control planes designed to build neighbor relationships across multiple routed hops through the network, manual configuration is often the easiest mechanism available. From a complexity perspective, manual configuration adds very little to the protocol itself; there is no need for any form of multicast neighbor advertisements, for instance. On the other hand, manual configuration of neighbors does require configuring the neighbor information, which increases complexity from a configuration point of view. In the network in Figure 11-4, router A would need to have neighbor relationships configured with B and C, router B would need to have neighbor relationships configured with A and C, and router C would need to have neighbor relationships configured with A and B. Even if the configuration of neighbors is automated, manual configuration deepens and broadens the interaction surfaces between the management and control planes.

Inferring neighbors from routing advertisements is a solution that was once widespread, but has become less common. In this scheme, each device advertises reachability and/or topology information on a periodic basis. The first time a router receives routing information from some other device, it adds the remote device to a local neighbor table. So long as a neighboring device continues sending routing information on a regular basis, the neighbor relationship will be considered active, or *up*.

When inferring neighbors from routing advertisements, it is important to be able to determine when a neighbor has failed (so reachability and topology information learned from the neighbor can be removed from any local tables). The most common way to solve this problem is with a pair of timers: the *hold* or *dead* timer, and the *update* or *advertisement* timer. So long as the neighbor sends an update or advertisement within the dead or hold timer, it is considered up or active. If an entire dead period passes without receiving any updates, the neighbor is considered dead, and some action is taken to either validate the topology and reachability information learned from the neighbor, or it is simply removed from the table.

The normal relationship between the dead and update timer is 3×—the dead timer is set to three times the update timer. Hence, if a neighbor does not send three updates or advertisements in a row, the dead timer wakes up, and begins processing the down neighbor.

Explicit hellos are the most common neighbor discovery mechanism. Hello packets are transmitted based on a *hello timer,* and the neighbor is considered dead if a hello is not received during the interval of a dead or hold timer. This is similar to the dead and update timers used in inferring neighbors from routing advertisements. Hellos typically contain information about the neighboring system, such as capabilities supported, device level identifiers, etc.

Centralized registration is another mechanism sometimes used to discover, and propagate information about, neighboring devices. Each device connecting to the network will send information about itself to some service, and, in turn, learn about

other devices connected to the network from this centralized service. This centralized service must somehow be discovered, of course, which is generally accomplished using one of the other mechanisms mentioned.

Detecting Two-Way Connectivity

In control planes with more complex adjacency formation processes—particularly protocols that rely on hellos to form neighbor relationships—it is important to detect if two routers can see one another (communicate bidirectionally) before forming a relationship. Ensuring two-way connectivity not only prevents unidirectional links from creeping into the forwarding table, but it also prevents a constant cycle of neighbor formation—discover a new neighbor, build the correct local tables, advertise reachability to the new neighbor, time out waiting for a hello or some other information, remove the neighbor, or discover the new neighbor. There are three broad options in managing two-way connectivity between network devices.

Do not bother checking for two-way connectivity. Some protocols do not try to determine if two-way connectivity exists between network devices in the control plane, but rather assume a neighbor from which packets are being received must also be reachable.

Carry a list of neighbors heard from on the link. For protocols that use hellos to discover neighbors and maintain liveness, carrying a list of reachable neighbors on the same link is a common method to ensure two-way connectivity exists. Figure 11-5 illustrates.

In Figure 11-5, assume router A is powered on before B. In this case:

1. A will send hellos with an empty neighbor list, as it has not heard hellos from any other network device on the link.
2. When B is powered on, it will receive A's hello, and hence include A in a list of neighbors it has heard in its hello packets.

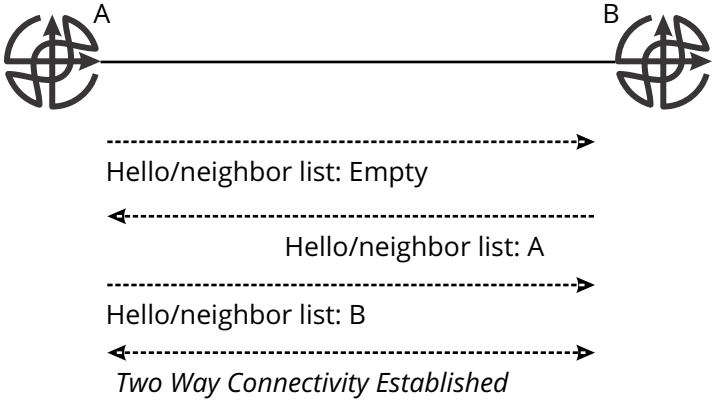


Figure 11-5 *Two-Way Handshake for Control Plane Two-Way Connectivity Check*

3. When A receives B's hello, it will, in turn, include B in its "heard from" neighbor list in its hello packets.
4. When both A and B are reporting one another in their "heard from" neighbor lists, both routers can be certain two-way connectivity has been established.

This process is often called a *three-way handshake*, based on the three steps:

1. A must send a hello to B, so B can include A in its neighbor list.
2. B must receive A's hello, and include A in its neighbor list.
3. A must receive B's hello with itself (A) in B's neighbor list.

Rely on an underlying transport protocol. Finally, control planes can rely on an underlying transport mechanism to ensure two-way connectivity exists. This is an uncommon solution, but there are some widely deployed solutions. For instance, the Border Gateway Protocol (BGP), explained in Chapter 16, "Link State and Path Vector Control Planes," relies on the Transmission Control Protocol (TCP), considered in Chapter 5, "Higher Layer Data Transports," to ensure two-way connectivity between BGP speakers.

Detecting the Maximum Transmission Unit

It is often useful for a control plane to move beyond just checking for two-way connectivity. Many control planes also check to make certain the Maximum Transmission Unit (MTU) on both interfaces onto the link are configured with the same MTU. Figure 11-6 illustrates the problem being solved with a link-level MTU check in the control plane.

In a situation where the MTU is mismatched between two interfaces on the same link, it is possible for a neighbor relationship to form but routing and other information to fail to be carried between the network devices. While many protocols have some mechanism to prevent information about the resulting unidirectional links from being used in calculating loop-free paths through the network, it is still useful to detect this situation so it can be explicitly reported and repaired. Several techniques are commonly used by control plane protocols to either explicitly detect this condition, or to at least prevent the initial stages of neighbor formation from taking place.

The control plane protocol can include the locally configured MTU in a field in the hello packets. Rather than just checking for the existence of a neighbor during the three-way handshake, each router can also check to make certain the MTU on both ends of the link match before adding a newly detected network device as a neighbor.

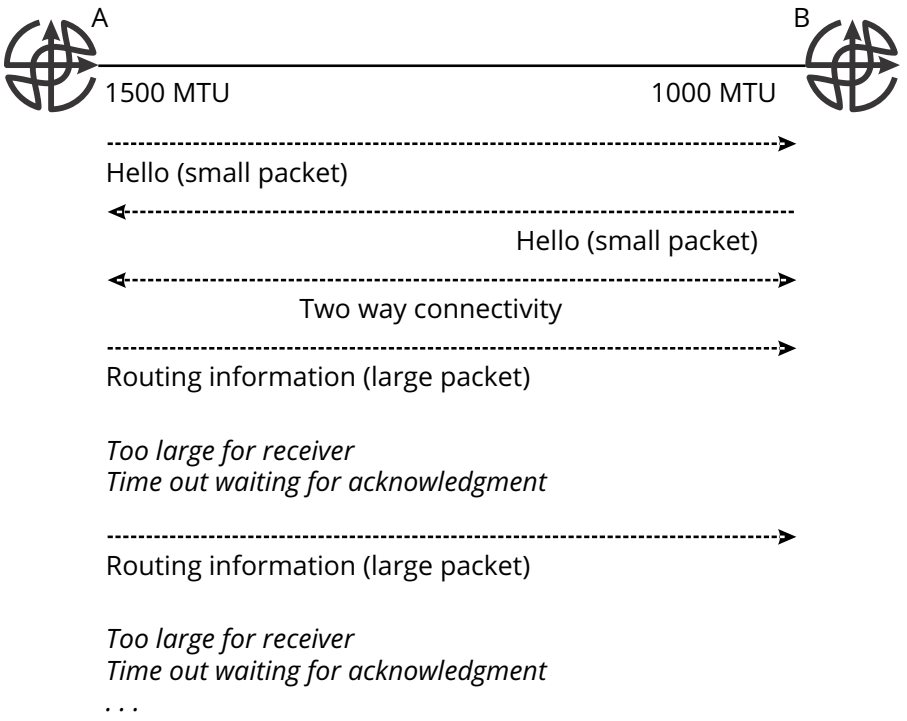


Figure 11-6 *The Impact of a Mismatched MTU on the Control Plane*

Another option is to pad the hello packets to the MTU of the local interface. If the padded, maximum-sized, hello packet is not received by some other device on the link, the initial stages of the neighbor relationship will not complete. The three-way handshake cannot be completed if both devices are not receiving one another's hello packets.

Finally, the control plane protocol can rely on an underlying transport to regulate packet sizes so the communicating devices can receive them. This mechanism is primarily used in control planes designed to overlay some other control plane, particularly in the case of interdomain routing and network virtualization. Overlay control planes often rely on Path MTU (PMTU) discovery to provide an accurate MTU between two devices connected through multiple hops.

The MTU size itself can have a large impact on the performance of a control plane in terms of its speed of convergence. For instance, assume a protocol must send information describing 500,000 destinations over a multihop link with 500ms of delay, and each destination requires 512 bits to describe:

- If the MTU is less than 1,000 bits, the control plane will require 500,000 round trips to exchange the entire database of reachable destinations, or around 500,000 × 500ms, which is 250,000 seconds, or close to 70 hours.

- If the MTU is 1,500 octets, or 12,000 bits, the control plane will require around 21,000 round trips to describe the entire database of reachable destinations, or around 21,000 × 500ms, which is around 175 minutes.

The importance of compressing such a database, using some sort of windowing mechanism to reduce the number of full round trips required to exchange the reachability information and increasing the MTU, is readily apparent.

Learning about Reachable Destinations

Neighbor discovery allows the control plane to learn about the topology of the network, but how is information about reachable destinations learned? In Figure 11-7, how does router D learn about hosts A, B, and C?

There are two broad classes of solutions to this problem—*reactive* and *proactive*—discussed in the following sections.

Learning Reactively

In Figure 11-7, assume host A has just been powered on, and the network is only using dynamic learning based on transmitted data traffic. How can router D learn about this newly attached host? One possibility is for A to simply start sending packets. For instance, if A is manually configured to send all packets toward destinations it does not know how to reach (essentially, anything that is *off segment*, a concept considered in Chapter 6, "Interlayer Discovery") to D, A has to send at least one packet for D to discover its existence. On learning of A, D can cache any relevant information for some time—generally for as long as A appears to be sending traffic. If A does not send traffic for some time, D can time the entry for A in its local cache out.

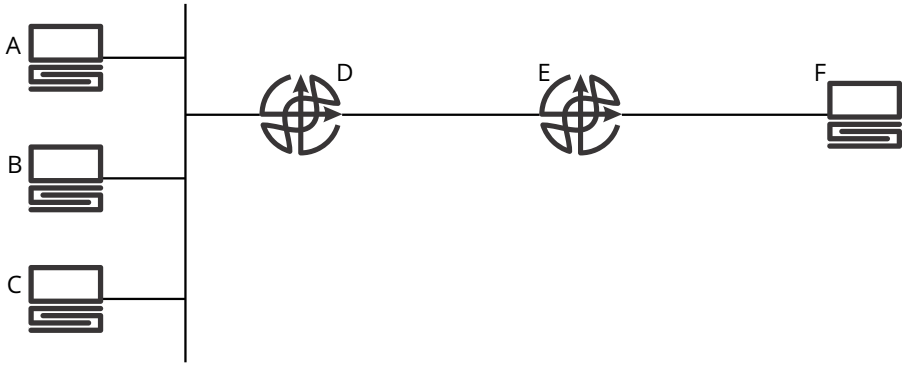


Figure 11-7 *Discovering Reachability*

This process of discovering reachability based on actual traffic flow is *reactive discovery*. From a complexity perspective, reactive discovery trades optimal traffic flow against the information known about, and potentially carried, in the control plane.

It will take some amount of time for reactive discovery mechanisms to operate—that is, for D to learn about the existence of A once the host starts sending packets. For instance, if host F begins sending traffic toward A the moment A is powered on, traffic may be forwarded through the network to D, but D will not have the information required to forward the traffic onto the link, and hence to A. During the time between host A being powered on and D discovering its existence, packets will be dropped—a situation that will appear, to F, to be a network failure at the worst, and some additional jitter (or perhaps an unpredictable response across the network) at best.

Cached entries will need to be timed out over time. This will normally require balancing a number of factors, including how large the cache is, how much device information is cached, and how often the cache entry has been used in some past time period.

How long it will take to time out this cached information and any security risk of some other device using stale information is the foundation for an attack. For instance, if A moves its connection from D to E, the information D has learned about A will remain in D's cache for some time. During this time, if another device connects to the network to D, it can impersonate A. The longer cached information is valid, the more possible it is to execute this type of attack.

Learning Proactively

Some reachability information can be learned proactively, which means the router does not need to wait for an attached host to start sending traffic to learn about it. This capability tends to be important in environments where hosts can be highly mobile; for instance, in a data center fabric where virtual machines may move between physical devices while keeping their address or other identifying information, or in networks that support wireless devices, such as mobile phones. There are four widely used ways to learn reachability information proactively, covered here:

- **A neighbor discovery protocol** can be run between the edge networking nodes (or devices) and connected hosts. The information learned from such a neighbor discovery protocol can then be used to inject reachability information in the control plane. While neighbor discovery protocols are widely deployed, the information learned through these protocols is not widely used to inject reachability information into the control plane.
- **Reachability information can be learned through device configuration.** Almost all network devices (such as routers) will have a reachable address

configured or discovered on all host-facing interfaces. Network devices can then advertise these attached interfaces as reachable destinations. In this situation, the link (or wire), the network, or the subnet is the reachable destination, rather than individual hosts. This is the most common way for routers to learn network layer reachability information.

- **Hosts can register with an identity service.** In some systems, a service (whether centralized or distributed) keeps track of where hosts are attached, including such information as the first hop router through which traffic should be sent to reach them, name to address mapping, services each host is capable of providing, services each host is searching for and/or using, and other information. Identity services are common, although they are not often highly visible to network engineers. Such systems are very common in high mobility environments, such as consumer-facing wireless networks.
- **The control plane can pull information from an address management system,** if one is deployed throughout the network. This is a very uncommon solution, however. Most of the interaction between the control plane and address management systems would be through local device configuration; the address management system assigns an address to an interface, and the control plane picks up this interface configuration to be advertised as a reachable destination.

Advertising Reachability and Topology

Once topology and reachability information are learned, the control plane must distribute this information through the network. While the method used to advertise this information is somewhat dependent on the mechanism used to calculate loop-free paths (as which information is required where to calculate loop-free paths will vary depending on how these paths are calculated), there are some common problems and solutions that will apply to every possible system. The primary problems are *deciding when to advertise reachability* and *reliably transporting information through the network*.

Deciding When to Advertise Reachability and Topology

When should the control plane advertise topology and reachability information? The obvious answer might be "when it is learned"—but the obvious answer is often the wrong answer. Determining when to advertise information actually involves a careful balance between optimal network performance and managing the amount of control plane state. Figure 11-8 will be used to illustrate.

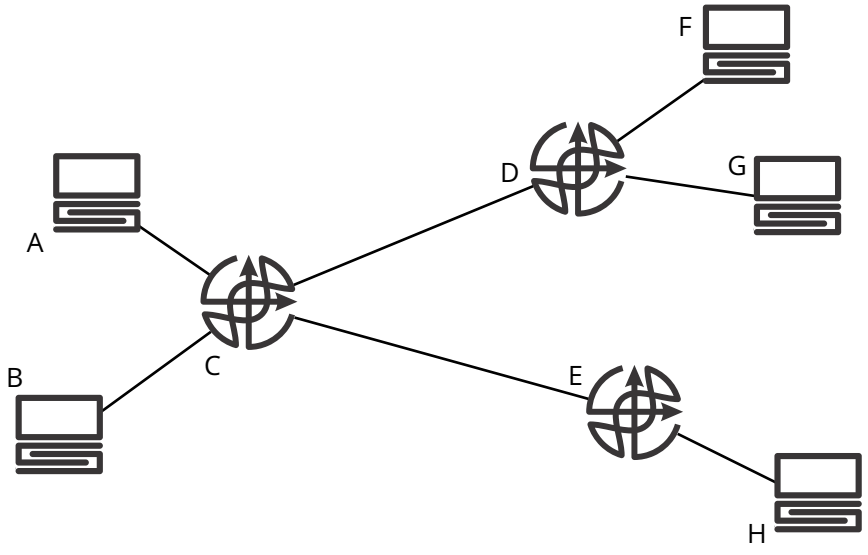


Figure 11-8 *When to Advertise Reachability and Topology Information*

Assume hosts A and F are sending data to one another almost constantly, but B, G, and H do not send traffic at all for some extended period. Two obvious questions arise in this situation:

- While it might make sense for router C to maintain reachability information about B, why should D and E maintain this information?
- Why should router E maintain reachability information about host A?

From a complexity perspective, there is a direct tradeoff between the amount of information carried and held in the control plane and the ability of the network to accept and forward traffic quickly. Considering the first question, for instance, the tradeoff appears as C's ability to send traffic from B to G on receiving it versus C maintaining less information in its forwarding tables, but being required to obtain the information required to forward traffic through some mechanism on receiving packets that need to be forwarded. There are three broad solutions to this problem.

- **A Proactive Control Plane:** The control plane can proactively discover the topology, calculate a set of loop-free paths through the network, and advertise reachability information.
- **Proactive Topology Discovery with Reactive Reachability:** The control plane can proactively discover the topology and calculate a set of loop-free

paths. However, the control plane can wait until reachability information is needed to forward packets before discovering and/or advertising reachability.

- **A Reactive Control Plane:** The control plane can reactively discover the topology, calculate a set of loop-free paths through the network (generally on a per destination basis), and advertise reachability information.

If C learns, keeps, and distributes reachability information proactively, or this network is running a proactive control plane, then new flows of traffic can be forwarded through the network without any delays. If the devices illustrated are running a reactive control plane, C would

- Wait until the first packet in the flow toward G (for instance)
- Discover the path to G using some mechanism
- Install the path locally
- Begin forwarding traffic toward G

The same process would need to be performed at D for traffic being forwarded toward A from G and F (remember flows are almost always bidirectional). During the time the control plane is learning a path to the destination, traffic is (almost always) being dropped, because the network devices do not have any forwarding information for this reachable destination (from the network device's perspective, the reachable destination does not exist). The time required to discover and build the correct forwarding information may fall between a few hundred milliseconds to a few seconds; during this time, the host and applications will not know whether or not connectivity will eventually be established, or if the destination is just unreachable.

Control planes can be broadly classified into

- Proactive systems advertise reachability information throughout the network before it is needed. Another way to phrase this is to say proactive control planes keep reachability information for every destination installed at every network device, regardless of whether the information is being used or not. Proactive systems increase the amount of state carried and stored in the control plane to make the network more transparent to hosts, or rather more optimal for short-lived and time-sensitive flows.
- Reactive systems wait until forwarding information is needed to obtain it, or rather they react to the events in the data plane to build control plane information. Reactive systems decrease the amount of state carried in the control plane by making the network less responsive to applications, and less optimal for short-lived or time-sensitive flows.

As with all tradeoffs in network engineering, the two options described here are not exclusive. It is possible to implement a control plane that contains some proactive, and some reactive, elements. For instance, it is possible to build a control plane that has minimal amounts of reachability information describing rather suboptimal paths through the network, but that can discover more optimal paths if a longer lived, or quality of service sensitive flow, is detected.

Reactive Distribution of Reachability

Returning to Figure 11-8 as a reference, assume a reactive control plane has been deployed, and B would like to start exchanging data flows with G. How can C develop the forwarding information required to correctly switch this traffic?

The router can send a query through the network or send a query to a controller to discover a path to the destination. For instance:

- When B first connects to the network, and C learns about this newly attached host, C could send information about B as a reachable destination to a controller attached to the network.
- In the same way, when G connects to the network, and D learns about this newly attached host, D could send information about G as a reachable destination to a controller attached to the network.

Because the controller learns about every host (or reachable destination) attached to the network (and, in some systems, the entire topology of the network, as well), when C needs to learn how to reach host G, the router can query the controller, which can provide this information.

Note

The concept of a centralized controller implies a single controller providing information for the entire network, but this is not how the term *centralized control plane* is commonly used throughout the network engineering world. The idea of *centralization,* however, is rather loose in network engineering. Rather than indicating a single device, *centralized* is generally used to mean *not carried hop by hop through the network, and not computed by each network device independently.* See Chapter 18, "Centralized Control Planes," for more information.

The router (or host) can send an *explorer* packet that records the route from the source to the destination and report this information to the source of the explorer, which is then used as a source route. Figure 11-9 illustrates.

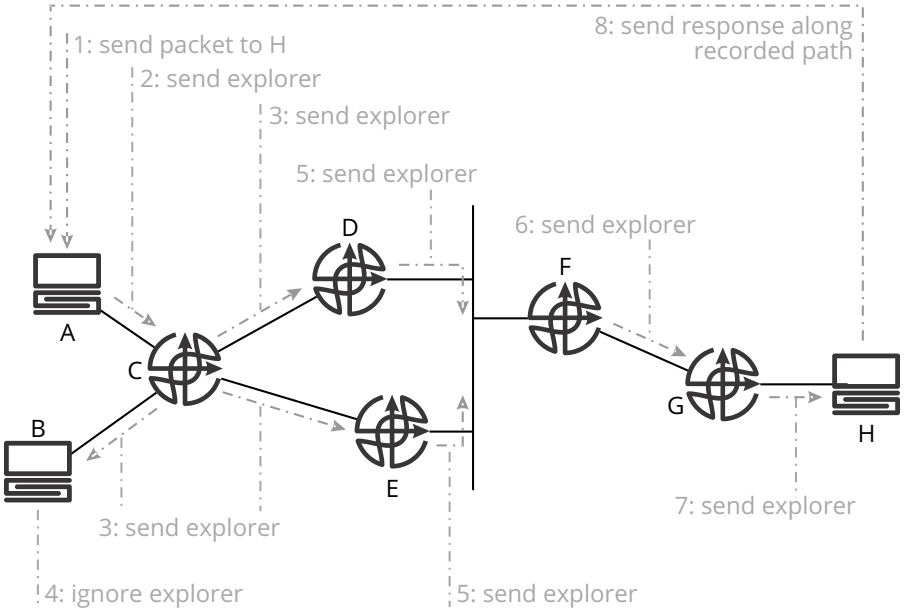


Figure 11-9 *Source Route Discovery*

Using Figure 11-9, and assuming host-based source routing:

1. Host A needs to send a packet to H but does not have a path.
2. A sends an *explorer* to its default gateway, router C.
3. C does not have a route to the destination, so it forwards the explorer packet onto all links other than the one it received the packet on; hence to B, D, and E.
4. B is a host, has no further interfaces, and is not the target of the explorer, so it ignores the explorer packet.
5. Neither D nor E has a path to H, so they both forward the explorer onto all interfaces except the one they received the packet on; hence onto the multi-access link shared between themselves and F.
6. F receives two copies of the same explorer packet; it chooses one based on some local criteria (such as the first received, or some control plane policy) and forwards it onto all the interfaces on which it did not receive the packet, toward G.
7. G receives the packet and, given it does not have a path to reach H, forwards it onto the only other link it has, which leads to H.
8. H receives the explorer and responds.

In this scheme, each device along the path adds itself to a list of *traversed nodes* before forwarding the explorer packet to *all interfaces except the one on which it was received*. In this way, when H receives the explorer packet (which is ultimately directed at finding a path to H), the packet now describes a complete path from A to H. When H replies to the explorer, it places this path into the body of the packet; when A receives the response, it will now have a complete path from A to H.

Note

In some implementations, A would not either generate or receive the response to the explorer packet. Rather C, the first hop router, could perform these functions. In the same way, H itself may not respond to these explorer packets, but rather G, or any other network device along the path that has information about how to reach G. The general concept and processing remain the same in these cases, however.

To send packets to H, then, A inserts this path into the packet header in the form of a *source route* containing the path [A,C,D,F,G,H]. When each router receives this packet, it will examine the source route in the header to determine which router to forward the traffic to next. For instance, C will examine the source route information in the packet header and determine the packet needs to be sent to D next, while D will examine this information and determine it needs to send the packet to F.

Note

In some implementations, every explorer is actually sent to the destination, which then determines which path traffic should take. There are, in fact, a number of different ways to implement source routing; the process given here is just one example to explain the general idea of source routing.

Proactive Distribution of Reachability

Proactive control planes, in contrast to reactive control planes, distribute reachability and topology information throughout the network when the information becomes available, rather than when it is needed to forward packets. The primary challenge proactive control planes face is in ensuring that reachability and topology information is carried reliably between the nodes in the network, resulting in every device having the same reachability information.

Note

This is really a distributed database problem; Chapter 14, "Reacting to Topology Changes," considers the distribution of reachability and topology within the context of a database in more detail.

Dropping control plane information can result in permanent routing loops or create *routing black holes* (so called because they consume traffic transmitted to destinations with no trace), both of which seriously reduce the usefulness of the network for applications (probably an understatement). There are several widely used mechanisms to ensure the reliable transportation of control plane information through a network.

A control plane can transmit information periodically, timing out older information. This is similar to neighbor formation, in that each router in the network will transmit the reachability information it has to all neighbors (or on all interfaces, depending on the control plane), based on a timer, usually called an *update* or *advertisement* timer. Reachability information, once received, is held in a local table and timed out over some time period, often called the *hold timer* (again, just like a neighbor discovery hello).

The remaining mechanisms described here rely on an existing neighbor discovery system to ensure the reliable delivery—and continued reliability—of reachability information. In all of these systems:

- The list of neighbors is used to drive not only the transmission of new reachability information, but also verifying the correct receipt of reachability information.
- So long as a neighbor is active, or alive, reachability information received from that neighbor is assumed to remain valid.

Within the context of neighbor-based reachability distribution, there are several commonly used mechanisms to make certain reachability information is carried device to device; often any given control plane will deploy more than one of the techniques described here.

The control plane can use sequence numbers (or some other mechanism) to ensure correct replication. Sequence numbers can actually be used to describe individual packets and large blocks of reachability information; Figure 11-10 illustrates.

On receiving a packet, the receiver can send an acknowledgment of the receipt of the packet by noting the sequence numbers it has received. A separate sequence number can be used to describe individual Network Layer Reachability

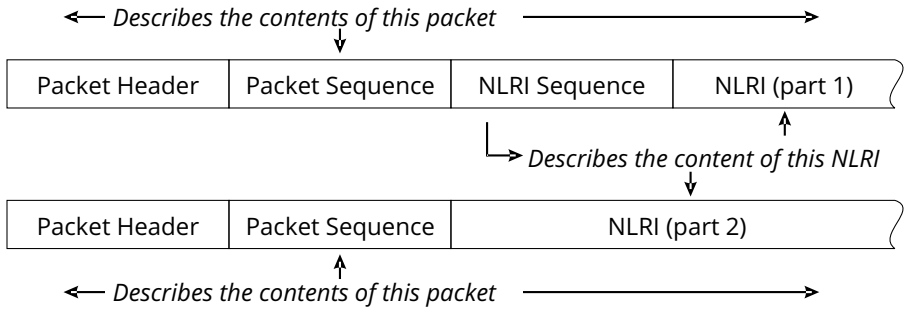


Figure 11-10 *Sequence Numbers Used to Ensure Reliable Replication*

Information (NLRI) as it is carried through the network. NLRI information spread out over several packets can then be described using a single sequence number.

The control plane can describe the database to ensure correct replication. For instance, a control plane could describe the information in the database as

- A list of sequence numbers matching individual entries containing reachability information contained in the database
- Groups of contiguous sequence numbers contained in the database (a somewhat more compact way to represent all the sequence numbers)
- A set of sequence numbers paired with hashes of the information within each reachability information entry; this has the advantage of not only describing the entries in the database, but also of providing a way for the receiver to verify the contents of each entry, yet without carrying the entire database to perform the check
- A hash across blocks of reachability entries contained in the database, which can be calculated across the same entries by the receiver and directly compared to determine if entries are missing

These kinds of database descriptors can be transmitted periodically, or only when there are changes, or even in other specific situations to not only ensure the network devices have synchronized databases, but also to determine what is missing or in error, so the additional information can be requested.

Each of these schemes has advantages and disadvantages; generally, protocols will implement a scheme that allows an implementation to not only check for missing information, but also information that has been inadvertently corrupted either in memory or during transmission.

Redistribution between Control Planes

There are many instances where it is more effective, or in line with specific policy restrictions, for a control plane to learn reachability and topology information from another control plane, rather than through the mechanisms outlined up to this point in this chapter. Some examples might be as follows:

- Two organizations need to interconnect their networks, but neither wants to allow the other to control the policies and operation of their control planes
- A large organization is made up of many business units, each of which is allowed to run its own internal network based on local conditions and application requirements.
- An organization needs some way to allow two control planes to interoperate while transitioning from one to the other.

The reasons for allowing one control plane to learn reachability information from another are almost boundless. Given the requirement, many network devices allow operators to redistribute information between control planes. Redistributing reachability raises two control plane–related problems: how to handle metrics and how to prevent routing loops.

> **Note**
>
> Redistribution can be seen as exporting routes out of one protocol and into another. In fact, import/export and redistribution are often used to mean the same thing, either by different vendors, or even in different situations by the same vendor.

Redistribution and Metrics

The relationship between link properties, policies, and metrics are defined by each control plane protocol independently of other protocols; in fact, a more descriptive, or otherwise more useful, metric system is what sometimes attracts operators to a specific control plane protocol. Figure 11-11 illustrates two sections of a network running two different control planes, each of which uses a different method to calculate link metrics.

Protocols X and Y, in this network, have been configured using two different systems for assigning metrics. In deploying protocol X, the administrator divided 1,000 by the link speed in gigabits. In deploying protocol Y, the administrator set up a

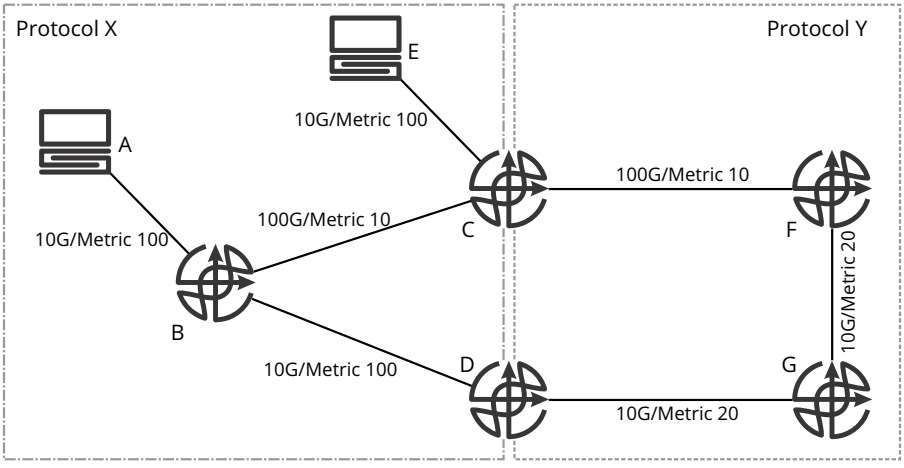


Figure 11-11 *Redistribution and Metrics*

"table of metrics," based on a best guess at the highest and lowest speed links they might have for the next 10 to 15 years, and assigned metrics to different link speeds within this table. The result, as the illustration shows, is incompatible metrics:

- 10G links in protocol X have a metric of 100, while in protocol Y they have a metric of 20.
- 100G links in both protocol X and Y have a metric of 10.

Assuming the lower metric is preferred, if the metrics are added, the [B,C,F] link would be considered a more desirable path than the [B,D,G] link. If the bandwidth is considered, however, both links would be considered equally desirable.

If redistribution is configured between these two protocols, how should these metrics be handled? There are three common solutions to this problem.

The administrator can assign a metric at each redistribution point, which is carried as part of the internal protocol metric. For instance, the administrator might assign a metric of 5 to the destination E at router C when redistributing from protocol X into Y. This destination, E, is injected into protocol Y with a metric of 5 by router C. At router F, the metric to E would be 25 through C. At G, the cost to reach E would be 35, along the path [F,C]. The desirability of using any particular exit point for any specific destination is chosen by the operator when these manual metrics are assigned.

The metric of the "other" protocol can be accepted as part of the internal protocol metric. This does not work in the case where one protocol has a wider

range of available metrics than the other. For instance, if protocol Y has a maximum metric of 63, the 10G metrics from protocol X will be "above maximum"; a situation that is not likely to be optimal. Assuming no such restriction, router C would inject a route to E with a cost of 100 into protocol Y. The cost to reach E at router F would be 110; the cost at G would be 130 through [F,C].

Note

You might recognize a tradeoff between control plane state and optimal use of the network here, another instance of the complexity tradeoffs in real-world protocol design. Carrying the external metric in a separate field adds control plane state, but allows more optimal steering of traffic through the network. Assigning or consuming the external metric reduces control plane state, but at the cost of being able to optimize traffic flow.

The external metric can be carried as a separate field, so each network device can make a separate determination about the best path to each external destination. This third solution is the most widely used, as it provides the best ability to steer traffic between the two networks. In this solution, C injects reachability to E with an external cost of 100. At F, there are two metrics in the advertisement describing reachability to E; the internal metric to reach the redistribution (or exit) point is 20, and the metric to reach E within the external network is 100. At G, the internal metric to reach the exit point is 30, and the external metric is 100.

How would an implementation use both of these metrics? Should the protocol choose the closest exit point, or rather the lowest internal metric? This would optimize the local network usage, and potentially deoptimize the usage of network resources in the external network. Should the protocol choose the exit point closest to the external destination, or rather the lowest external metric? This would optimize network resources in the external network, potentially at the cost of deoptimizing the use of network resources in the local network. Or should the protocol try to combine these two metrics in some way, to optimize the use of resources in both networks as much as possible?

Some protocols choose to always optimize local or external resources, while others will provide operators with a configuration option. For instance, a protocol may allow external metrics to be carried as different *types* of metrics, where one type is considered larger than any internal metric (hence preferring the lowest internal metric first, and using the external metric as a tie breaker), and the other type is where the internal and external metrics are considered equivalent (hence adding the internal and external metrics to make a path decision).

Redistribution and Routing Loops

In the discussion above, you might have noticed that destinations redistributed from one protocol to another always appear as if they are connected to the redistributing router. In essence, redistribution acts as a form of summarization (which means topology information is removed, rather than reachability information), as described earlier in this chapter. While this point isn't crucial to redistribution metrics, it is important to consider in the ability of the control plane to choose the optimal path. In some specific cases, deoptimization can lead to a complete failure of the control plane to choose loop-free paths; Figure 11-12 illustrates.

To build the routing loop in this network:

1. The route to host A is redistributed from protocol X to Y with a manually configured metric of 1.
2. Router E prefers the route through C with a total metric (internal and external) of 2.
3. Router D prefers the route through E with a total metric of 3.
4. Router D redistributes the route to host A into protocol X with the existing metric of 3.
5. Router B has two routes to A: one with a cost of 10 (directly) and one with a metric of 4 through D.
6. Router B chooses the path through D, creating a routing loop.
7. And so on (the loop will continue until each protocol reaches its maximum metric).

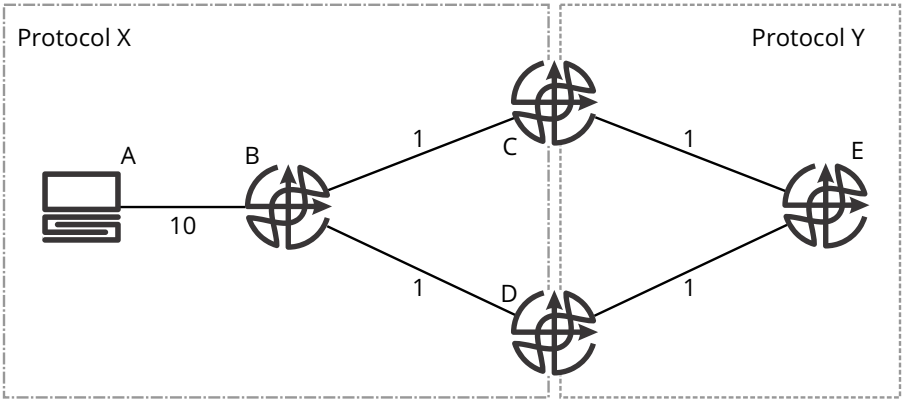


Figure 11-12 *Redistribution Routing Loop*

This example is a little stretched to create a routing loop in a trivial network, but all routing loops caused by redistribution are similar in their structure. It is important, in this example, that not only has topology information been lost (the route to A has been summarized, appearing, from E's perspective, to be directly attached to C), but metric information has been lost as well (the original route, with a cost of 11, is redistributed into protocol Y with a cost of 1 at C). There are a number of common mechanisms used to prevent this routing loop from forming.

The routing protocol can always prefer internal over external routes. In this case, if B always prefers the internal route to A over the external path through D, the routing loop cannot form. Many routing protocols will use an ordering preference when installing routes into the local routing table (or Routing Information Base, RIB), to always prefer internal routes over external ones. The reason for this preference is to prevent routing loops of this type from forming.

Filters could be configured to prevent individual destinations from being redistributed twice. In this network, router D could be configured to prevent any external route received in protocol Y from being redistributed into protocol X. In a situation where there are only two protocols (or networks) with control plane information redistributed between them, this can be a simple solution. In cases where the filters need to be configured for each destination, the filters can quickly become difficult to manage. Mistakes in configuring these filters can either cause some destinations to become unreachable (routing black holes), or permit a loop to form, potentially causing a failure in the control plane.

Routes can be tagged when they are redistributed, and then filtered based on these tags at other redistribution points. For instance, when the route to A is redistributed into protocol Y at C, the route could be administratively tagged with some number, such as *100,* so the route can be easily identified. At router D, a filter could be configured to block any route marked with the tag *100,* preventing the routing loop from forming. Many protocols allow a route to carry an administrative tag (sometimes called a *community,* or some other similar name), and then to filter routes based on this tag.

Final Thoughts on Topology Discovery

This chapter covered a lot of ground, mostly in the process of considering a wide array of problems that control planes face in some fundamental areas. For each of these problems, a range of solutions was offered, many of which are implemented by real control plane protocols used in running networks throughout the world.

Discovering the topology on a per link basis was the first problem considered, including detecting other network devices, determining if two-way connectivity exists between devices, and determining the MTU (and whether or not it matches).

Learning about reachable destinations was the second problem considered. Two broad classes of solutions were considered here: reactive and proactive. Advertising reachability information was divided into the same two broad classes, reactive and proactive, but reliable transmission of information through the network was also considered in some detail. Finally, redistribution between routing protocols was considered, as this is a common way for a control plane to learn about reachable destinations in an indirect way.

You will meet these problems, and their solutions, again in considering actual protocol implementations in Chapter 15, "Distance Vector Control Planes," and Chapter 16, "Link State and Path Vector Control Planes," which consider distributed and centralized control plane implementations in more detail. Each of these problems and their solutions are fundamental to the operation of successful control plane protocols in the real world.

Further Reading

Alekseev, V. B., V. P. Kozyrev, and A. A. Sapozhenko. "Graph Theory," February 2011. https://www.encyclopediaofmath.org/index.php/Graph_theory.

Caldwell, Chris K. "Graph Theory Tutorials," 1995. http://primes.utm.edu/graph/.

Doyle, Jeff, and Jennifer DeHaven Carroll. *Routing TCP/IP, Volume 1*. 2nd edition. New Delhi, India: Cisco Press, 2005.

"Enhanced Interior Gateway Routing Protocol." *Cisco*. Accessed September 4, 2017. https://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/16406-eigrp-toc.html.

Huang, Peng, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. "Gray Failure: The Achilles' Heel of Cloud-Scale Systems." In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, 150–55. HotOS '17. New York, NY, USA: ACM, 2017. doi:10.1145/3102980.3103005.

Krebs, Valdis. "The Social Life of Routers." *Internet Protocol Journal*, December 2000. http://www.orgnet.com/SocialLifeOfRouters.pdf.

Lahey, Kevin. *TCP Problems with Path MTU Discovery*. Request for Comments 2923. RFC Editor, 2000. doi:10.17487/RFC2923.

Mathis, Matt, and John Heffner. *Packetization Layer Path MTU Discovery*. Request for Comments 4821. RFC Editor, 2007. doi:10.17487/RFC4821.

McCann, Jack, Stephen E. Deering, Jeffrey Mogul, and Robert M. Hinden. *Path MTU Discovery for IP Version 6*. Request for Comments 8201. RFC Editor, 2017. doi:10.17487/RFC8201.

Medved, Jan, Nitin Bahadur, Hariharan Ananthakrishnan, Xufeng Liu, Robert Varga, and Alexander Clemm. "A Data Model for Network Topologies." Internet-Draft. Internet Engineering Task Force, March 2017. https://tools.ietf.org/html/draft-ietf-i2rs-yang-network-topo-12.

Moy, John. *OSPF Version 2*. Request for Comments. RFC Editor, April 1998. doi:10.17487/RFC2328.

Rekhter, Yakov, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. Request for Comments 4271. RFC Editor, 2006. doi:10.17487/rfc4271.

Retana, Alvaro, Russ White, and Don Slice. *EIGRP for IP: Basic Operation and Configuration*. 1st edition. Boston, MA: Addison-Wesley Professional, 2000.

Savage, Donnie, Steven Moore, James Ng, Russ White, Donald Slice, and Peter Paluch. *Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)*. Request for Comments 7868. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7868.txt.

White, Russ, Alvaro Retana, and Don Slice. *Optimal Routing Design*. 1st edition. Cisco Press, 2005.

Review Questions

1. Classify each device as either a transit or a leaf node:
   a. A mobile phone being used as a hot spot
   b. A router
   c. A database server
   d. A switch
   e. A proxy server
2. Explain the difference between aggregation and summarization as it is used in the chapter (and throughout this book).
3. Note the kind of neighbor discovery, two-way connectivity check, and link MTU discovery used in each of the following routing protocols:
   a. Open Shortest Path First (OSPF)
   b. Intermediate System to Intermediate System (IS-IS)
   c. Routing Information Protocol (RIP)
   d. Border Gateway Protocol (BGP)

4. Classify each of the following protocols as reactively or proactively discovering the topology and calculating the set of loop-free paths through the network:
   a. Spanning Tree Protocol (STP)
   b. Open Shortest Path First (OSPF)
   c. BABEL
   d. OpenFlow
5. Classify each of the following protocols as reactively or proactively discovering and advertising reachable destinations:
   a. Spanning Tree Protocol (STP)
   b. Open Shortest Path First (OSPF)
   c. BABEL
   d. OpenFlow
6. Describe a situation where an overflow in a cache used to hold forwarding information can cause the control plane to forward packets until the cache is either timed out or otherwise cleared.
7. Read the explanation of a gray failure (from the paper noted in the "Further Reading" section). How do you think gray failures might relate to the discovery of neighbor status and checking for two-way connectivity?
8. Assume you could tag routes as they are being redistributed, and then filter based on those tags at all other redistribution points. Can you explain how this kind of tagging could be used to prevent redistribution routing loops?
9. It seems it would be possible to build a table that converts metrics from one protocol to another automatically during the redistribution process, and yet very few (almost no) routing protocols are designed with this kind of capability. What would be the problem with such a system?
10. One protocol, the Enhanced Interior Gateway Routing Protocol (EIGRP), does allow a routing process to set the external metrics directly from the external routing process. Can you figure out the circumstances when this is possible, and explain why?

Chapter 12

Unicast Loop-Free Paths (1)

Learning Objectives

After reading this chapter, you should be able to understand:

- The relationship between calculating a set of shortest paths and calculating a set of loop-free paths
- The concept of a Loop-Free Alternate and remote Loop-Free Alternate paths
- The difference between a minimum spanning tree and a Shortest Path Tree, and how they are calculated
- The waterfall or continental divide and P/Q models of preventing routing loops
- The concept of a greedy algorithm in finding loop-free paths
- The Bellman-Ford algorithm for finding loop-free paths
- The horizon point and split horizon
- How to find loop-free paths in the Diffusing Update Algorithm (DUAL)

Network engineers typically think of the control plane as doing a wide variety of things, from calculating the shortest path through the network to distributing policy used to forward packets. The idea of the *shortest path,* however, sneaks in the concept of the optimal path. Likewise, the idea of policy also sneaks in the concept of optimization of network resources. While both policy and the shortest path are important, neither one of these is at the root of what the control plane does. The job

of the control plane is to find a set of loop-free paths through a network first; optimization is a nice add-on, but optimization can only be "done" in the context of finding a set of loop-free paths.

The question this chapter will answer, then, is

How does a control plane calculate loop-free paths through a network?

This chapter will begin by examining the relationship between the shortest, or lowest metric, path and loop-free paths. The next topic considered is Loop-Free Alternate (LFA) paths, which are not the best paths but still loop free. Such paths are useful in designing control planes that quickly switch from the best path to an alternate loop-free path in the case of failures or changes in the network topology. Two specific mechanisms used for finding a set of loop-free paths are then discussed; two more are discussed in Chapter 13, "Unicast Loop-Free Paths (2)."

Which Path Is Loop Free?

The relationship between the *shortest path,* generally in terms of metrics, and *loop-free paths* is fairly simple: *the shortest path is always loop free.* The reason for this relationship can be expressed most simply in terms of geometry (or more specifically *graph theory,* which is a specialized field of study within discrete mathematics). Figure 12-1 is used to explain why.

What are the paths available from A, B, C, and D toward the destination?

- From A: [B,H]; [C,E,H]; [D,F,G,H]
- From B: [H]; [A,C,E,H]; [A,D,F,G,H]
- From D: [F,G,H]; [A,C,E,H]; [A,B,H]

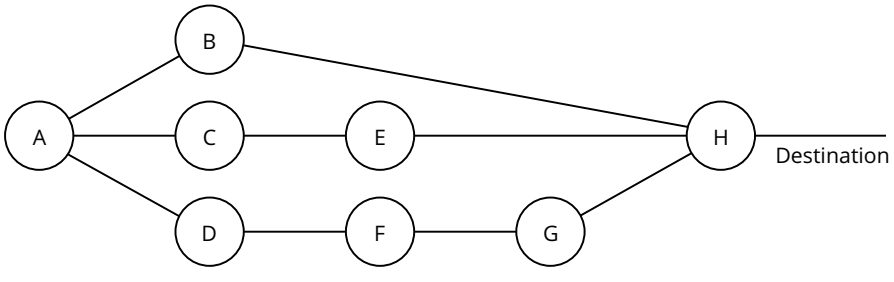


Figure 12-1 *Available Paths Through a Network*

If every device in the network must choose the path it will use toward the destination independently (without reference to the path chosen by any other device), it is possible to form persistent loops. For instance, A could choose the path [D,F,G,H], and D could choose the path [A,C,E,H]. Device A will then forward traffic toward the destination to D, and D will then forward traffic toward the destination to A. There must be some rule other than *choose a path* implemented by the algorithm used to calculate a path on each device, such as *choose the shortest (or lowest cost) path*. But why does choosing the shortest (or lowest cost) path prevent the loop? Figure 12-2 illustrates.

Figure 12-2 assumes A chooses the path [D,F,G,H] to the destination, and D chooses the path through A to the destination. What D cannot know, because it is calculating a path to the destination without any knowledge of what A has calculated, is that A is using the path through D itself to reach the destination. How can the control plane avoid such a loop? By observing that the cost of a path along a loop must always contain the cost of the loop as well as the loop-free element of the path. In this case, the path through A, from the perspective of D, must include the cost from D to the destination. Hence the cost through A, from the perspective of D, will always be greater than the lowest available cost from D. This leads to the following observation:

> **The lowest cost (or shortest) path cannot contain a path that passes through the calculating node**; or rather, **the shortest path is always loop free**.

There are two important points about this observation.

First, this observation does *not* say paths with higher costs are *definitely* loops, only that the lowest cost path *must not* be a loop. It is possible to expand the rule to discover a wider set of loop-free paths beyond the lowest cost path; these are called Loop-Free Alternates.

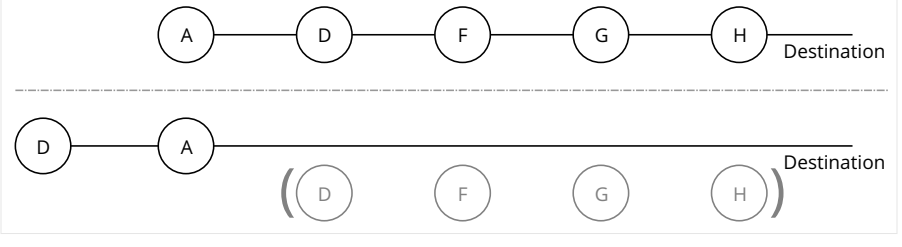


Figure 12-2 *Longer Paths Contain Shorter Ones*

Second, this observation holds only if every node in the network has the same view of the network topology. Nodes can have different views of the network topology for a number of reasons; for instance:

- The network topology has changed, and all the nodes have not yet been notified of the change; hence microloops.
- Some information about the network topology has been removed from the topology database through summarization or aggregation.
- The metrics have been configured so the lowest cost path is inconsistent from different perspectives.

Control planes used in real networks are carefully crafted to either work around or minimize the impact of different devices having different views of the network topology, potentially causing a looped path. For instance:

- Control planes are carefully tuned to minimize the time differential between learning of a topology change and modifying forwarding (or to drop traffic during topology changes, rather than forwarding it).
- When summarizing topology or aggregating reachability, care is taken to preserve cost information.
- Network design "best common practices" encourage the use of symmetric metrics, and many implementations make it difficult or impossible to configure links with truly dangerous metrics, such as a zero link cost.

It often takes a great deal of design work to find, and work around or prevent, the unintended subversion of the shortest path rule in real-world control plane protocols.

Why Not Use a Node List?

An obvious question, at this point, should be: *why not simply use a node list to find loop-free routes?* For instance, in Figure 12-1, if A calculates a path through D, can D just somehow obtain the path A has calculated, discover that D itself is in the path, and hence not use the path through A?

The first problem with this mechanism is in the discovery process. How should D learn about the path A has chosen, and A learn about the path D

has chosen, without causing a race condition? The two devices could choose one another as their next hop toward the destination at the same moment and then inform one another at the same moment, resulting in both choosing some other path at the same time. The result could either be a stable set of loop-free paths, the two devices cycling between choosing one another and having no path to the destination, or satable condition where there is no path to the destination.

The second problem with this mechanism is summarization—the intentional removal of information about the network topology to reduce the amount of state carried in the control plane. The control plane will only have metrics to work with wherever the topology is summarized; hence it is better to use a rule based on metrics, or costs, rather than the set of nodes through which a path passes.

Note both of these problems can be solved; there are, in fact, path-vector algorithms that rely on a list of nodes to calculate loop-free paths through a network. While these systems are widely deployed, they are often considered too complex to be deployed in many network engineering situations. Hence metric-, or cost-, based systems are widely used.

Trees

The simple shortest path rule is used to build a description of a set of paths, rather than a single path, in real-world networks. While a number of different kinds of trees can be used to represent a set of paths through a topology or network, there are two commonly used to describe computer networks: the Minimum Spanning Tree (MST) and the Shortest Path Tree (SPT). The difference between these two kinds of trees is often subtle. The network shown in Figure 12-3 will be used to illustrate the MST and SPT.

In Figure 12-3, a number of different paths will touch every node; for instance, from A's perspective:

1. [A,B,E,D,C] and [A,C,D,E,B], each with a total cost of 10
2. [A,B,E] with a cost of 5 and [A,C,D] with a cost of 3, for a total cost of 8
3. [A,C,D,E] with a cost of 6 and [A,B] with a cost of 1, for a total cost of 7

An *MST* is a tree that visits each node in the network with the minimum overall cost (normally measured as the sum of all the links chosen in the network).

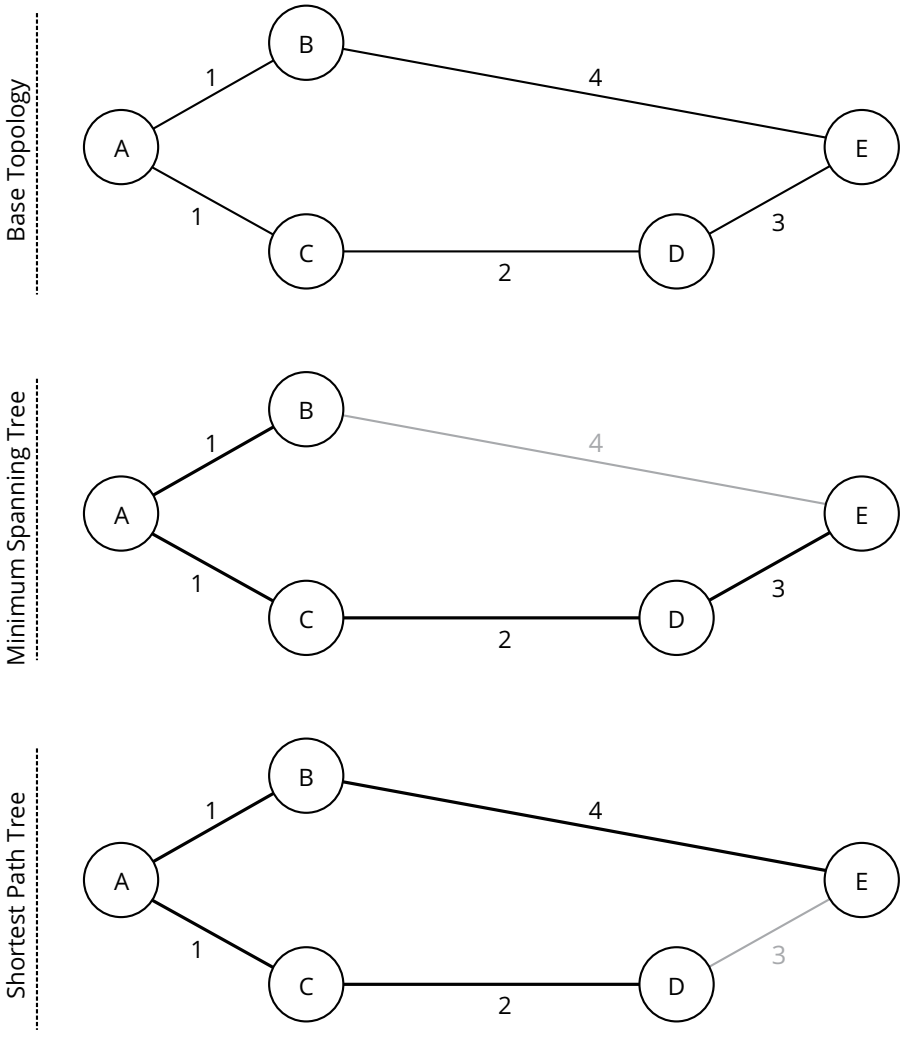


Figure 12-3 *The Minimum Spanning Tree and the Shortest Path Tree*

An algorithm that computes the MST will choose option 3, as it has the lowest total cost along the set of edges required to reach every node in the network.

An *SPT* describes the shortest path to each destination in the network, independent of the total cost of the graph. An algorithm that calculates an SPT would choose, from A's perspective:

- [A,B] to B with a cost of 1, as this path is shorter than [A,C,D,E,B] with a cost of 10
- [A,B,E] to E with a cost of 5, as this is shorter than [A,C,D,E] with a cost of 6

- [A,C] to C with a cost of 1, as this is shorter than [A,B,E,D,C] with a cost of 10
- [A,C,D] to D with a cost of 3, as this is shorter than [A,B,E,D] with a cost of 8

Comparing the set of shortest paths to the set of paths that will touch every node, above, an algorithm that calculates an SPT would choose option 2, rather than 3 in the preceding list. In other words, the SPT will ignore the total cost of the edges in the MST to find the shortest path to each reachable destination (in this case, nodes), while the MST will ignore the shortest path to each reachable destination in order to minimize the cost of the entire graph.

Network control planes most often compute SPTs, rather than MSTs, using some form of *greedy algorithm*. While SPTs are not optimal for solving all network traffic flow problems, they are generally better than MSTs in the types of traffic flow problems that network control planes must solve.

Greedy Algorithms

Greedy algorithms choose locally optimal solutions to solve larger problems. For instance, in calculating the shortest path through a network, a greedy algorithm may choose to visit closer neighboring nodes (can be reached across a link with lower cost) before nodes that are farther away (can be reached across a link with a higher cost). In this way, greedy algorithms can be said to relax computation, normally by either ignoring or approximating global optimization.

Sometimes greedy algorithms can fail; when they do fail, they can fail spectacularly, providing the worst possible solution. For instance, with the right set of metrics, it is possible for a greedy algorithm, such as Dijkstra's (described in Chapter 13, "Unicast Loop-Free Paths (2)"), to calculate the set of longest paths through a network, rather than the set of shortest. Hence greedy algorithms are sometimes considered a *heuristic,* as they *approximate* the solution to a hard problem, or can solve it in constrained environments, rather than actually solving the general problem.

In the real world, computer networks are designed in a way to make these algorithms compute the best possible solution to the problem at hand in every case—namely, finding the shortest set of paths through a network.

Alternate Loop-Free Paths

The shortest path rule, as described in the preceding section, is a negative test, rather than a positive one; it can always be used to find a loop-free path among a set of

available paths, but not to determine which other paths in the set might also happen to be loop free. Figure 12-4 illustrates.

In Figure 12-4, it is easy to observe that the shortest path from A to the destination is along the path [A,B,F]. It is also easy to observe that the paths [A,C,F] and [A,D,E,F] are alternate paths to the same destination. But are these paths loop free? The answer depends on the meaning of *loop free:* normally a loop-free path is one in which the traffic will not loop through any node (will not visit any node in the topology more than once). While this definition is generally good, it is possible to narrow the definition in the case of a single node with multiple next hops over which it can send traffic toward a reachable destination. Specifically, the definition can be narrowed to:

A path is loop free if the next hop device will not forward traffic toward a specific destination back to me (the sending node).

In this case, the path through C, from A's perspective, can be said to be loop free if C does not forward traffic toward the destination through A. In other words, if A transmits a packet to C for *Destination,* C will not forward the packet back to A, but rather will forward the packet closer to *Destination.* This definition simplifies the problem of finding alternate loop-free paths somewhat. Rather than considering the entire path toward the destination, A needs to only consider whether or not any particular neighbor will forward traffic back to A itself when forwarding traffic towards the destination.

Consider, for instance, the path [A,C,F]. If A sends a packet to C for the destination beyond F, will C forward this packet back to A? The paths available to C are

- [C,A,B,F], with a total cost of 5
- [C,A,D,E], with a total cost of 6
- [C,F], with a total cost of 2

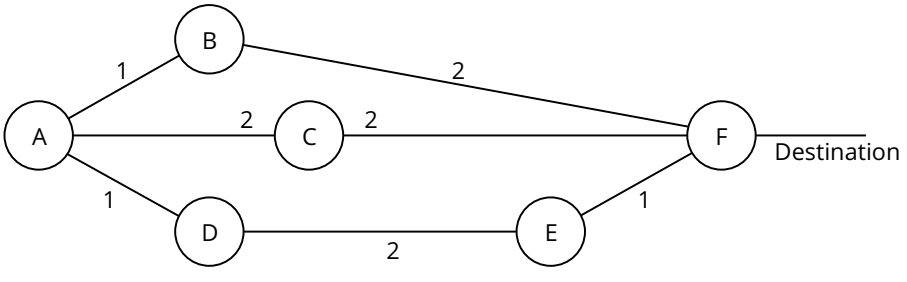


Figure 12-4 *Alternate Loop-Free Paths*

Given C is going to choose the shortest path to the destination, it will choose [C,F], and hence will not forward the traffic back to A. Turning this into a question: why will C not forward traffic back to A? Because it has a path that is *lower cost than any path through A* to reach the destination. This can be generalized and called a *downstream neighbor:*

Any neighbor with a path that is shorter than the local path to the destination will not loop traffic back to me (the sending node).

Or rather, given that the local cost is represented as LC, and the neighbor's cost is represented as NC, then

If NC < LC, then the neighbor is downstream.

Now consider the second alternate path shown in Figure 12-4: [A,D,E,F]. Once again, if A sends traffic toward the destination to D, will D loop the traffic back to A? The paths D has available are

- [D,A,C,F], with a total cost of 5
- [D,A,B,F], with a total cost of 4
- [D,E,F], with a total cost of 3

Assuming D will use the shortest available path, D would forward any such traffic through E, rather than back through A. This can be generalized and called a Loop-Free Alternate (LFA):

Any neighbor with a path that is shorter than the local path to the destination plus the cost of the neighbor to reach me (the local node) will not loop traffic back to me (the local node).

Or rather, given the local cost is represented as LC, the neighbor's cost is represented as NC, and the cost back to the local node (from the neighbor's perspective) is BC:

If NC + BC < LC, then the neighbor is an LFA.

There are two other models often used to explain Loop-Free Alternates: the waterfall model and P/Q Space. It is useful to look at these models in a little more detail.

Waterfall (or Continental Divide) Model

One way to prevent loops in the routes calculated by a control plane is to simply not advertise routes to neighbors that would forward traffic back to me (the sending node). This is called split horizon; it leads to the concept of traffic flowing through a network acting like water along a waterfall, or stream bed, taking the path of least resistance toward the destination, as shown in Figure 12-5.

In Figure 12-5, if traffic enters the network at C (at Source 2) and is destined beyond E, it will flow down the right side of the ring. If, however, traffic enters the

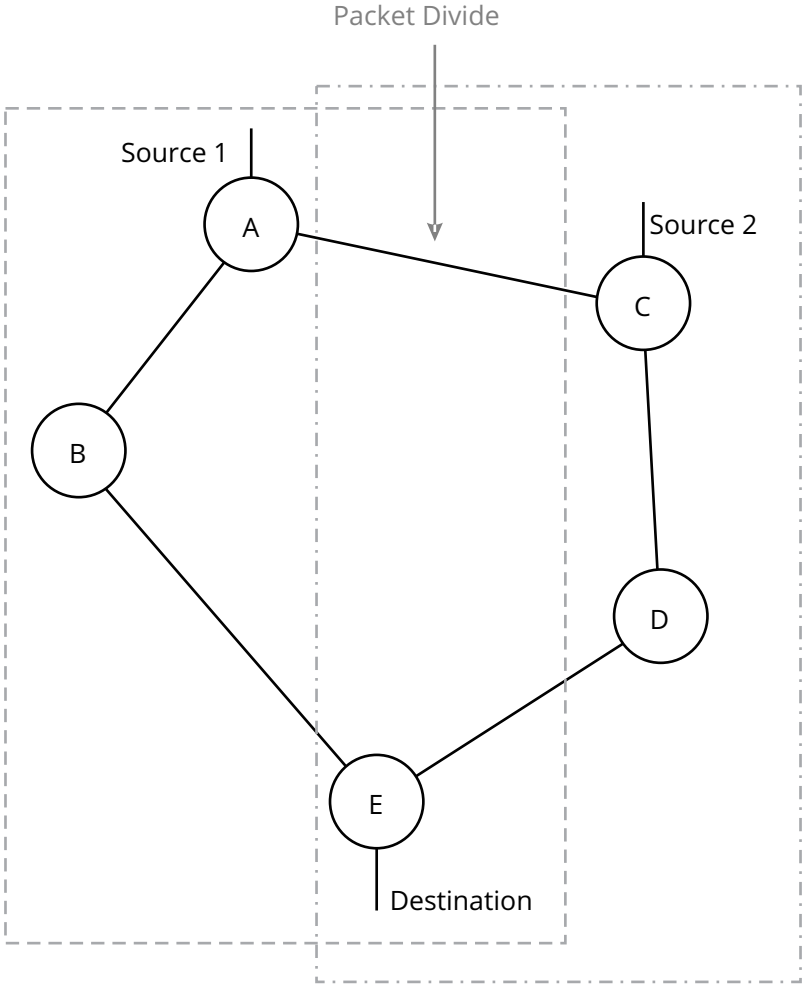


Figure 12-5 *Traffic Flow from a Metric-based Packet Divide*

network at A and is destined beyond E, it will flow down the left side of the ring. To prevent traffic destined beyond E from looping on this ring, one simple thing the control plane can do is either not allow A to advertise the destination to C, or not allow C to advertise the destination to A. Preventing one of these two routers from advertising to the other is called split horizon, because it stops a route from being propagated across a horizon, or rather beyond the point where any particular device knows traffic being passed along a particular link will be looped.

Split horizon is implemented by only allowing a device to advertise reachability through interfaces it is not using to reach the destination in question. In this case:

- D is using E to reach the destination, so it will not advertise reachability toward E
- C is using D to reach the destination, so it will not advertise reachability toward D
- B is using E to reach the destination, so it will not advertise reachability toward E
- A is using B to reach the destination, so it will not advertise reachability toward B

Hence, A blocks B from knowing about the alternate path that it has to the destination through C, and C blocks D from knowing about the alternate path that it has to the destination through A. A Loop-Free Alternate path will cross this split horizon point in the network. In Figure 12-5, A can calculate that C's path cost is less than A's path cost, so any traffic A forwards to C toward the destination will be forwarded along some other path than the one A knows about. C, in LFA terms, is a downstream neighbor of A.

An alternate way to look at the LFA calculation, then, is to find the split horizon point in the ring and determine whether or not the devices on either side of the split horizon point would forward traffic through the packet divide.

P/Q Space

Another model to describe how LFAs work is *P/Q Space;* Figure 12-6 illustrates.

It is easiest to begin with a definition of the two spaces. Assuming the [E,D] link is to be protected from failure:

- Calculate a reverse Shortest Path Tree from E (E uses the cost of the paths toward itself, rather than the costs away from itself, in calculating this tree, because traffic is flowing toward D on this path).
- Remove the [E,D] link, along with any nodes only reachable by passing through the link.
- The remaining nodes that E can reach are the Q space.

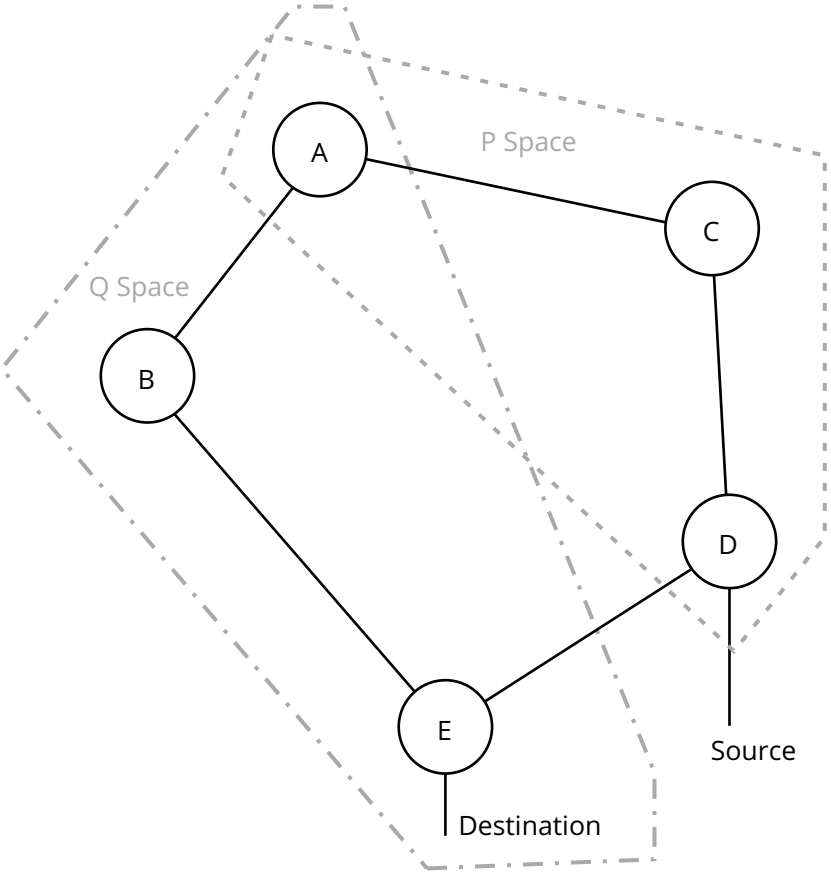


Figure 12-6 *P Space and Q Space*

- Calculate a Shortest Path Tree from D.
- Remove the [E,D] link, along with any nodes only reachable by passing through the link.
- The remaining nodes that D can reach are in the P space.

If D can find a router in the Q space to which to forward traffic if the [E,D] link fails, this is an LFA.

Remote Loop-Free Alternates

What if there is no LFA? It is sometimes possible to find a remote Loop-Free Alternate (rLFA), which can carry the traffic to the destination, as well. The rLFA is not

directly connected to the calculating router, but is rather one or more hops away; this means the traffic must be carried through the routers between the calculating router and the remote next hop; this is normally accomplished by tunneling the traffic.

These models can explain rLFAs without looking at the math required to calculate them. Understanding where a ring will "divide" into P and Q, or into the two halves divided by split horizon helps you quickly understand where an rLFA can be used to work around a failure even if no LFA is present. Returning to Figure 12-6, for instance, if the [E,D] link fails, D must simply wait for the network to converge to begin forwarding traffic toward the destination. The best path from E has been removed from D's tree by the failure, and E has no LFA it can forward traffic to.

Return to the restricted definition of a loop-free path that this section began with—any neighbor to which a device can forward traffic without the traffic being returned. There is no particular reason why the neighbor to which a device sends packets in the case of a local link failure must be locally connected. Chapter 9, "Network Virtualization," describes the ability to create a tunnel, or an overlay topology, that can carry traffic between any two nodes in the network.

Given the ability to tunnel traffic across C, so C does not forward traffic based on the actual destination, but rather on a tunnel header, D can forward traffic directly to A, bypassing the loop. When the [E,D] link fails, then, D can do the following:

1. Calculate the closest point in the network where traffic can be tunneled and will not return to C itself.
2. Form a tunnel to that router.
3. Encapsulate the traffic into the tunnel header.
4. Forward the traffic.

Note

In actual implementations, the rLFA tunnel would be precalculated, rather than calculated at the time of failure. These rLFA tunnels do not necessarily need to be visible to the normal forwarding process, as well. This text is arranged for clarity of *how* this process works, rather than focusing on how it is normally implemented.

D will forward the traffic to the tunnel destination, rather than the original destination; this bypasses C's local forwarding table entry for the original destination, which would loop the traffic back to C. The calculation of such intersection

points will be discussed in the section on Dijkstra's Shortest Path First algorithm in Chapter 13.

Bellman-Ford Loop-Free Path Calculation

Bellman-Ford is one of the simpler protocols to understand, as it is generally implemented by comparing newly learned information about a destination with existing information about the same destination. If the newly discovered route is better than the currently known route, the higher cost route is simply replaced in the path list—as dictated by the shortest path rule for finding loop-free paths through the network. By iterating over the entire topology in this way, a set of shortest paths to each destination is found. Figure 12-7 is used to illustrate the process.

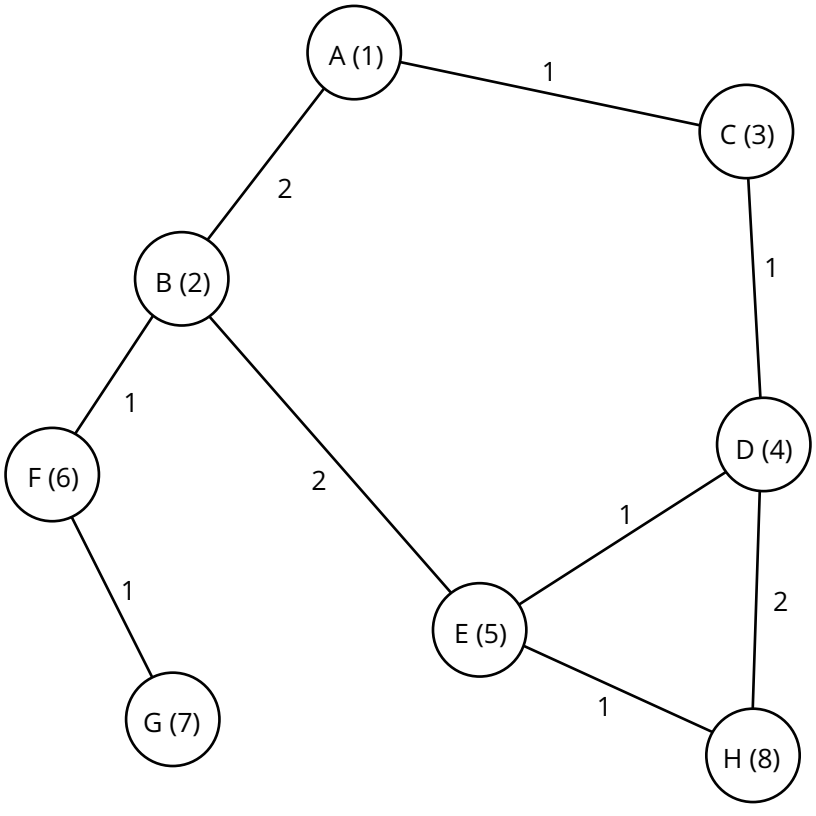


Figure 12-7 *A Sample Network to Run Bellman-Ford*

Note

While Bellman-Ford is mostly known for its distributed variant implemented in widely deployed protocols such as the *Routing Information Protocol (RIP),* it was originally designed as a search algorithm performed on a single structure describing a topology of nodes and edges. Bellman-Ford is discussed as an *algorithm* here. A distributed algorithm similar to Bellman-Ford is discussed in the next section.

Bellman-Ford as an Algorithm

Although first proposed by Alfonso Shimbel in 1955,[1] and again by Edward F. Moore in 1957,[2] this algorithm is named after Richard Bellman, who published it in 1958,[3] and Lester Ford, Jr., who published it in 1956.[4]

Bellman-Ford will calculate a Shortest Path Tree to each reachable destination in a worst case of O(V*E), where *V* is the number of nodes (vertices) in the network, and *E* is the number of links (edges). Essentially, this means the amount of time Bellman-Ford takes to operate over a topology and calculate a Shortest Path Tree is linear against the number of devices and links; doubling the number of either will double the amount of time it takes to run. Doubling both at the same time will increase the run time by a factor of 4.

Bellman-Ford is thus a moderately slow algorithm when used against larger topologies in the worst case, when the nodes in the topology table start out ordered from the farthest from the root to the closest to the root. If the topology table is sorted from the closest to the root to the farthest, Bellman-Ford can terminate in O(E), which is much faster; in the real world, it is difficult to ensure either ordering, so the actual time required to build a Shortest Path Tree is normally somewhere between O(V*E) and O(E).

Bellman-Ford is a greedy algorithm, operating by assuming every node in the network other than the local node is only reachable through an infinite cost, and replacing these infinite costs with actual costs as the topology is walked. Assuming all nodes are infinitely distant is called relaxing the calculation, as it uses an approximate distance for all unknown destinations in the network, replacing them with a real cost once it has been calculated.

1. Shimbel, "Structure in Communication Nets."
2. Moore, "The Shortest Path through a Maze."
3. Bellman, "On a Routing Problem." 87–90.
4. Ford, *Network Flow Theory.*

Note

The actual runtime of any algorithm used for calculating a Shortest Path Tree is normally swamped by the amount of time required to carry information about topology changes through the network; see Chapter 14, "Reacting to Topology Changes," for more information on this topic. Implementations of all of these protocols, particularly in their distributed form, will contain a number of optimizations to reduce their runtime to far below the worst case, so while the worst case is given as a reference point, it often has little (or no) bearing on the performance of each algorithm in actual deployed networks.

To run Bellman-Ford over this topology, it must first be converted into a set of vectors and distances, and stored in a data structure, such as shown in Table 12-1.

Table 12-1 *Topology, or Edges, Represented as a Table for Bellman-Ford*

Row	Source (s)	Destination (d)	Distance (cost)
1	F (6)	G (7)	1
2	E (5)	H (8)	1
3	D (4)	H (8)	2
4	D (4)	E (5)	1
5	B (2)	F (6)	1
6	B (2)	E (5)	2
7	C (3)	D (4)	1
8	A (1)	B (2)	2
9	A (1)	C (3)	1

There are nine entries in this table because there are nine links (edges) in the network. Shortest path algorithms calculate a unidirectional tree (in one direction along the graph). In the network in Figure 12-7, the SPT is shown originating at node 1, and calculation is shown moving away from node 1, which will be the point from which the calculation takes place. The algorithm, in pseudocode, is as follows:

Note

The data structures in this example are 1 referenced (or based), which means the first row is 1 rather than 0, to make the numbering clearer.

```
// create a set to hold the response, with one entry for each node
// the first slot in the resulting structure will represent node 1,
```

```
// the second node 2 etc.
define route[nodes] {
  predecessor // as a node
  cost // as an integer
}

// set the source (me) to 0 cost
// position 1 in the array is the origination point's entry
route[1].predecessor = NULL
route[1].cost = 0
// table 1, above, is held in an array called topo

// walk the topo (edges) table once for each entry in the route
// (results) table, replacing longer entries with shorter ones
i = nodes
while i > 0 {
  j = 1
  while j <= nodes { // iterates over every row in the topology
table
    source_router = topo[j].s
    destination_router = topo[j].d
    link_cost = topo[j].cost

    if route[source_router].cost == NULL {
      source_router_cost = INFINITY
    } else {
      source_router_cost = route[source_router].cost
    }

    if route[destination_router].cost == NULL {
      destination_router_cost = INFINITY
    } else {
      destination_router_cost = route[destination_router].cost
    }

    if source_router_cost + link_cost <= destination_router_cost {
      route[destination_router].cost = source_router_cost + link_
cost
      route[destination_router].predecessor = source_router
    }
    j = j + 1 //or j++ depending on what pseudocode this is
representing
  }
i = i - 1
}
```

This code is deceptive in appearing more complex than it really is. The key line is the comparison *if route[topo[j].s].cost + topo[j].cost < route[topo[j].d].cost;* it is useful to focus on this line through an example. In the first run through the outer loop (which is run once for each entry in the results table, called *route* here):

- For the first line of the topo table:
  - *j* is 1 so *topo[j].s* is node 6 (F), the source of the vector in the edge table
  - *j* is 1, so topo*[j].d* is node 7 (G), the destination of the vector in the edge table
  - *route[6].cost* = infinity, *topo[1].cost* = 1, and *route[7].cost* = infinity
  - infinity + 1 == infinity, so the condition fails and nothing else happens
- Any topo table entry with a source cost of infinity will give the same result, as infinity + anything will always equal infinity; the rest of the rows containing a source with a cost of infinity will be skipped.
- For the eighth line of the topo table (the eighth edge):
  - *j* is 8, so *topo[j].s* is node 1 (A), the source of the vector in the edge table
  - *j* is 8, so topo*[j].d* is node 2 (B), the destination of the vector in the edge table
  - *route[1].cost* = 0, *topo[8].cost* = 2, and *route[2].cost* = infinity
  - 0 + 2 <= infinity, so the condition succeeds
  - route[2].predecessor is set to 1, and route[2].cost is set to 2
- For the ninth line of the topo table (the ninth edge):
  - *j* is 9, so *topo[j].s* is node 1 (A), the source of the vector in the edge table
  - *j* is 9, so topo*[j].d* is node 3 (C), the destination of the vector in the edge table
  - *route[1].cost* = 0, *topo[9].cost* = 1, and *route[3].cost* = infinity
  - 0 + 1 <= infinity, so the condition succeeds
  - *route[3].predecessor* is set to 1, and *route[3].cost* is set to 1

In the second run of the outer loop:

- For the fifth line of the topo table (the fifth edge):
  - *j* is 5, so *topo[j].s* is node 2 (B), the source of the vector in the edge table
  - *j* is 5, so topo*[j].d* is node 6 (F), the destination of the vector in the edge table

- *route[2].cost* = 2, *topo[5].cost* = 1, and *route[6].cost* = infinity
- 2 + 1 <= infinity, so the condition succeeds
- *route[6].predecessor* is set to 2, and *route[6].cost* is set to 3

- For the sixth line of the topo table (the sixth edge):
  - *j* is 6, so *topo[j].s* is 2 (B), the source of the vector in the edge table
  - *j* is 6, so *topo[j].d* is 5 (E), the destination of the vector in the edge table
  - *route[2].cost* = 2, *topo[6].cost* = 2, and *route[5].cost* = infinity
  - 2 + 2 <= infinity, so the condition succeeds
  - *route[5].predecessor* is set to 2, and *route[5].cost* is set to 4
  - The remainder of this run is shown in Table 12-2.

In the third run of the outer loop, node 8 is of particular interest, as there are two paths to this destination.

- For the second line of the topo table (the second edge):
  - *j* is 2, so *topo[j].s* is node 5 (E), the source of the vector in the edge table
  - *j* is 2, so topo*[j].d* is node 8 (H), the destination of the vector in the edge table
  - *route[5].cost* = 4, *topo[2].cost* = 1, and *route[8].cost* = infinity
  - 4+1 <= infinity, so the condition succeeds
  - *route[8].predecessor* is set to 5, and *route[8].cost* is set to 5
- For the third line of the topo table (the third edge):
  - *j* is 3, so *topo[j].s* is node 4 (D), the source of the vector in the edge table
  - *j* is 3, so *topo[j].d* is node 8 (H), the destination of the vector in the edge table
  - *route[4].cost* = 2, *topo[3].cost* = 2, and *route[8].cost* = 5
  - 2+2 <= 4, so the condition succeeds
  - *route[8].predecessor* is set to 4, and *route[8].cost* is set to 4

The interesting point in the third cycle through the *topo* table is the entry for the edge [5,8] is processed first, which sets 8's (H's) predecessor to 5 and cost to 5. When the next line in the *topo* table is processed, however, the [4,8] edge, the algorithm discovers a shorter path to node 8 and replaces the existing one. Table 12-2 shows the state of the *route* table with each pass through the *topo* table.

Table 12-2 *Bellman-Ford Cycles Across the Sample Network*

	A (1)		B (2)		C (3)		D (4)		E (5)		F (6)		G (7)		H (8)	
	P	**C**	**P**	**C**	**P**	**C**	**P**	**C**	**P**	**C**	**P**	**C**	**P**	**C**	**P**	**C**
First Cycle	N	0	1	2	1	1	N	I	N	I	N	I	N	I	N	I
Second Cycle	N	0	1	2	1	1	3	2	2	4	2	3	N	I	N	I
Third Cycle	N	0	1	2	1	1	3	2	2	4	2	3	6	4	4	4

In Table 12-2, the top line represents an entry in the routing table and a node that is reachable in the network. For instance, A (1) represents the best path to A, B (2) represents the best path to B, etc. The *P* column represents the *predecessor,* or the node through which A must pass to reach the destination indicated. The *C* represents the cost to reach this destination. The sample network can be completed in three cycles, given the algorithm is coded to detect the completion of the tree. The pseudocode, as shown, does not have any test for this completion and would run the full 8 cycles (one for each node) anyway.

> **Note**
>
> Bellman-Ford can also support negative cost edges (unlike Dijkstra's algorithm); as these do not normally exist in a network, the process for handling these is not shown here.

Garcia's Diffusing Update Algorithm

The Diffusing Update Algorithm (DUAL) is one of the two algorithms discussed here originally designed to be implemented in a distributed network. It is unique in also having the removal of reachability and topology information contained in the algorithm's state machine. The other algorithms discussed here leave the removal of information to the implementation of the protocol, rather than considering this aspect of the algorithm's operation within the algorithm itself.

The Origins of the Diffusing Update Algorithm

By 1993, Bellman-Ford and Dijkstra had been implemented as distributed algorithms in several routing protocols. The experience gained from these early implementations and deployments led to a "second wave" of research into and thinking around the problem of routing in packet switched networks, resulting in path vector and DUAL. The abstract from the 1993 paper by J. J. Garcia-Luna-Aceves summarizes much of this previous work and experience, and proposes a new distributed computation system (rather than strictly an algorithm) to find the shortest path through a network more efficiently. In the earlier days of network engineering, memory, processor, and network utilization were three primary concerns in designing a protocol; processors were either 6- or 8-bit; memory was measured in kilobytes, and bandwidth was measured in kilobytes per second. Hence, the development of a protocol that was very efficient in these terms was a major breakthrough in deploying large-scale networks. The abstract of the original Garcia-Luna-Aceves paper is worth repeating here:

Abstract—A family of distributed algorithms for the dynamic computation of the shortest paths in a computer network or internet is presented, validated, and analyzed. According to these algorithms, each node maintains a vector with its distance to every other node. Update messages from a node are sent only to its neighbors; each such message contains a distance vector of one or more entries, and each entry specifies the length of the selected path to a network destination, as well as an indication of whether the entry constitutes an update, a query, or a reply to a previous query. The new algorithms treat the problem of distributed shortest-path routing as one of diffusing computations, which was first proposed by Dijkstra and Scholten. They improve on algorithms introduced previously by Chandy and Misra, Jaffe and Moss, Merlin and Segall, and the author. The new algorithms are shown to converge in finite time after an arbitrary sequence of link cost or topological changes, to be loop-free at every instant, and to outperform all other loop-free routing algorithms previously proposed from the standpoint of the combined temporal, message, and storage complexities.[5]

The protocol resulting from this paper, Enhanced Interior Gateway Routing Protocol (EIGRP), was widely deployed at scales that other protocols simply could not attain. DUAL, like Bellman-Ford, is a greedy algorithm, and would run (if it were implemented as a nondistributed algorithm) in O(E*V) in the worst case when calculating the initial set of Shortest Path Trees. Special optimizations, however, allow DUAL to operate very quickly in the case of topology changes.

5. Garcia-Luna-Aceves, "Loop-Free Routing Using Diffusing Computations." 130–41.

As DUAL is designed as a distributed algorithm, it is best to describe its operation across a network; Figure 12-8 and Figure 12-9 are used for this purpose. To explain DUAL, this example will trace the flow of A learning about three destinations and then processing changes in the state of reachability for these same destinations. The first example will consider the case where there is an alternate path, but no downstream neighbor; the second will consider the case there is an alternate path and a downstream neighbor.

Note

While the original DUAL paper refers to neighbor adjacencies, they will not be described in this discussion. Rather, it will simply be assumed such neighbors exist, and hence the transmission of control plane data is reliable.

In Figure 12-8, learning D from A's perspective:

1. A learns two paths to D:
   a. Through H with a cost of 3.
   b. Through C with a cost of 4.

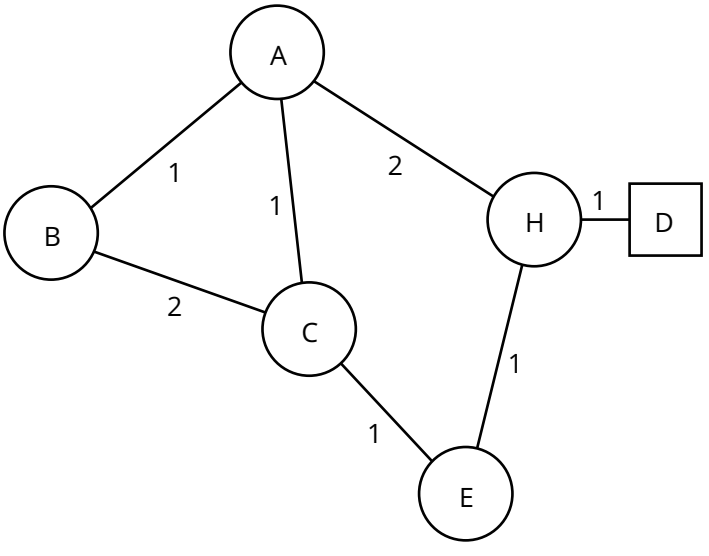


Figure 12-8 *First Network for Demonstrating the Diffusing Update Algorithm*

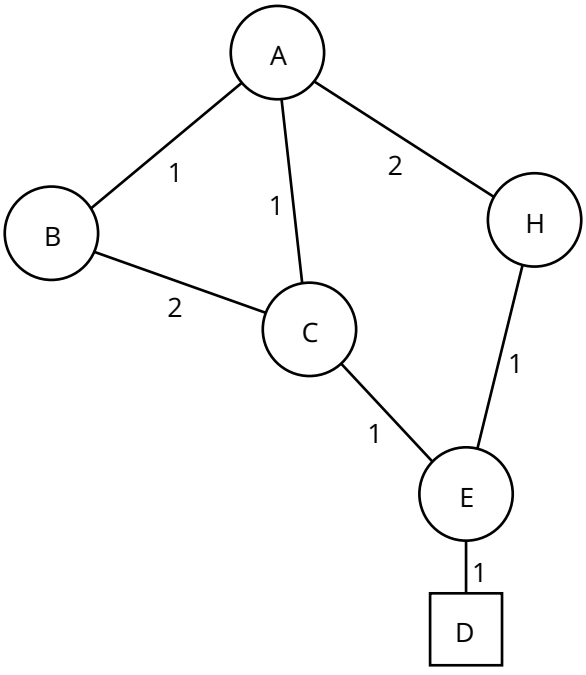


Figure 12-9 *Second Network for Demonstrating the Diffusing Update Algorithm*

2. A will not learn the path through B, because B is using A as its successor:
   a. A is the best path B has to reach D.
   b. As B is using the path through A to reach D (the destination), it will not advertise the route it knows about D (through C) to A.
   c. B will *split horizon* its advertisement of D toward A to prevent possible forwarding loops from forming.
3. A compares the available paths and chooses the shortest path as loop free:
   a. The path through H is marked as the *successor.*
   b. The *feasible distance* is set to the cost along the shortest path, which is 3.
4. A checks the remaining paths to determine if any of them are downstream neighbors:
   a. C's cost is 3.
      A knows this because C advertises the route to D with its local metric, which is 3.
      A saves C's local metric in its *topology table.*
      Hence, A knows the local cost at C and the local cost at A.

   b. 3 (the cost at C) >= 3 (the cost at A), so this route *may* be a loop, Hence, C does not meet the *feasibility condition*.

   c. C is not marked as a downstream neighbor.

Downstream neighbors are called *feasible successors* in DUAL.

Assume the [A,H] link fails. DUAL does not rely on periodic updates, so A cannot simply wait for another update with valid information; rather A must actively pursue an alternate path. This is, therefore, a *diffused* process of alternate path discovery. If the [A,H] link fails, considering just D:

1. A examines its local table for any feasible successors (downstream neighbors).
2. There are no feasible successors, so A must discover an alternate loop-free path to D (if one exists).
3. A sends a *query* to each neighbor to determine if there is some alternate loop-free path to D.
4. At C:
   a. C's successor is E (not A, from whom it received the query).
   b. E's cost is lower than A's cost to D; hence C's path is not a loop.
   c. C *replies* with its current metric of 3 to A.
5. At B:
   a. A is B's current successor.
   b. Through the query, B now discovers its best path to D has failed, and it must also find an alternate path.
   c. *B's processing is not considered here, but rather is left as an exercise for the reader.*
   d. B replies to A that it has no alternate path (responds with an infinite metric).
6. A receives these replies:
   a. The path through C is the only one available, with a cost of 4.
   b. A marks the path through C as its successor.
   c. There are no other paths to D; hence there is no feasible successor (downstream neighbor).

In Figure 12-9, the destination (D) has been moved from H to E; this will be used for the second example.

In this example, there is a feasible successor (downstream neighbor). Learning D from A's perspective:

1. A learns two paths to D:
   a. Through H with a cost of 4.
   b. Through C with a cost of 3.
2. A will not learn any path through B:
   a. B has two paths to D.
   b. Through both C and A with a cost of 4.
   c. B is using both A and C as its successors in this case.
   d. B will *split horizon* its advertisement of D toward A because A is marked as a successor.
3. A compares the available paths and chooses the shortest path as loop free:
   a. The path through C is marked as the *successor.*
   b. The *feasible distance* is set to the cost along the shortest path, which is 3.
4. A checks the remaining paths to determine if any of them are downstream neighbors:
   a. H's cost is 2.
   b. 2 (the cost at H) <= 3 (the cost at A), so this route *cannot* be a loop; hence H does meet the *feasibility condition*.
   c. H is marked as a feasible successor (downstream neighbor).

If the [A,C] link fails just considering A:

1. A will examine its local topology table for a feasible successor.
2. A feasible successor exists through H.
3. A switches its local table to H as the best path.
   a. No diffusing update has been run, so no paths have been verified or recalculated.
   b. Hence, the feasible distance cannot be changed; it remains at 3.
4. A sends an update to its neighbors noting its cost to reach D has changed from 3 to 4.

   The impact of this update is not described here, but consider that B is using A as a successor.

As you can see, processing when a feasible successor exists is much faster and simpler than without. In networks where a routing protocol using DUAL (specifically EIGRP) has been deployed, one primary design goal will be limiting the scope of any queries generated in the case where there is no feasible successor. Query scope is the primary determinant of how quickly the DUAL algorithm completes and hence how quickly the network converges.

Figure 12-10 illustrates a basic DUAL finite state machine.

Things included in *route gets worse* could include

- Failure of a connected link or neighbor
- Receiving an update for a route with a higher metric
- Receiving a query from the current successor

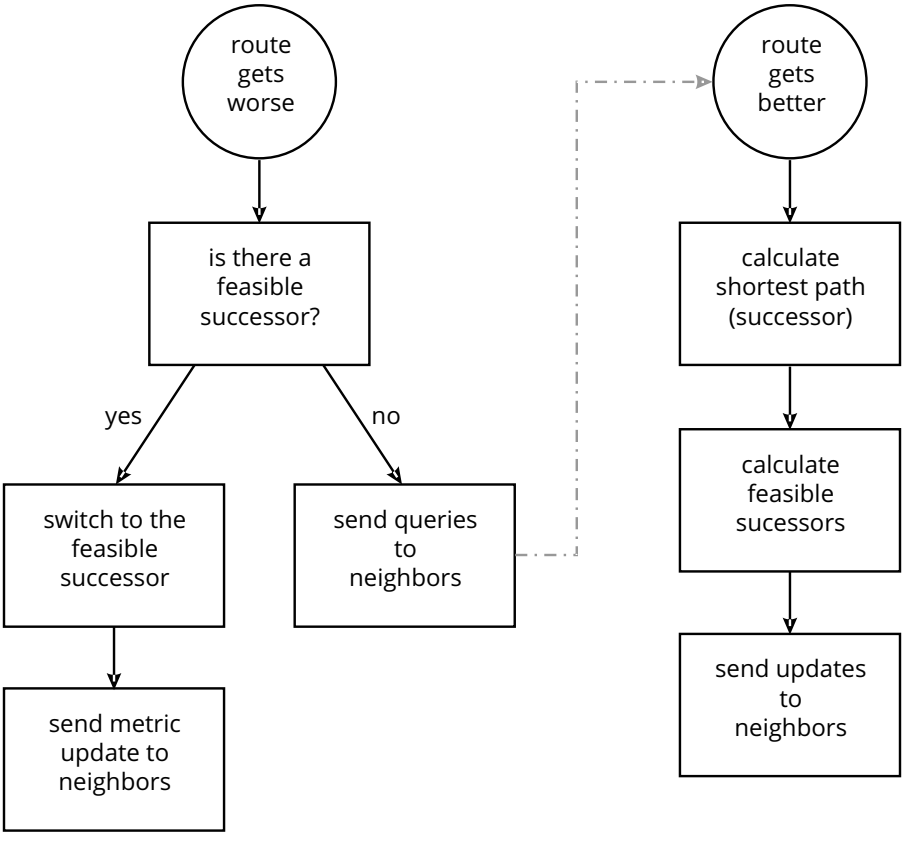


Figure 12-10 *A Simple DUAL Finite State Machine*

Things included in *route gets better* could include

- A new route learned from a neighbor
- A new neighbor discovered, along with the routes this neighbor can reach
- Receiving all queries sent to neighbors when a *route gets worse*

Final Thoughts

This chapter is the first of two discussing calculating loop-free paths through a network. The shortest path rule is the foundation of most calculation mechanisms; Bellman-Ford and DUAL, the foundation of most widely deployed *distance-vector* protocols (classifications of protocols are considered in more depth in Chapters 15 through 17, which discuss distributed and centralized control planes). The next chapter considers one more algorithm that relies on the shortest path rule, and then turns to path vector, and finally disjoint paths.

Further Reading

Bellman, Richard. "On a Routing Problem." *Quarterly of Applied Mathematics* 16 (1958): 87–90.

"Enhanced Interior Gateway Routing Protocol (EIGRP) Wide Metrics White Paper." *Cisco*. Accessed January 28, 2017. http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/enhanced-interior-gateway-routing-protocol-eigrp/whitepaper_C11-720525.html.

Ford, L. R. *Network Flow Theory*. Santa Monica, CA: RAND Corporation, 1956.

Garcia-Luna-Aceves, J. J. "Loop-Free Routing Using Diffusing Computations." *IEEE/ACM Transactions on Networking* 1, no. 1 (February 1993): 130–41.

Hendrick, C. *Routing Information Protocol*. Request for Comments 1058. RFC Editor, 1988. doi:10.17487/rfc1058.

Malkin, Gary S. *RIP Version 2*. Request for Comments 2453. RFC Editor, 1998. doi:10.17487/rfc2453.

Malkin, Gary S., and Robert E. Minnear. *RIPng for IPv6*. Request for Comments 2080. RFC Editor, 1997. doi:10.17487/rfc2080.

Moore, Edward F. "The Shortest Path through a Maze." In *Proceedings of the International Symposium on Switching Theory 1957, Part II*. Cambridge MA: Harvard University Press, 1959.

Perlman, Radia. "An Algorithm for Distributed Computation of a Spanningtree in an Extended LAN." *SIGCOMM Computer Communication Review* 15, no. 4 (September 1985): 44–53. doi:10.1145/318951.319004.

———. *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*. 2nd edition. Reading, MA: Addison-Wesley Professional, 1999.

Retana, Alvaro, Russ White, and Don Slice. *EIGRP for IP: Basic Operation and Configuration*. 1st edition. Boston, MA: Addison-Wesley Professional, 2000.

Russ White. "CAP Theorem and Routing." *Rule 11 Reader*, March 25, 2016. https://rule11.tech/cap-theorem-routing/.

———. "Ordered FIB." *Packet Pushers*, March 25, 2014. http://packetpushers.net/ordered-fib/.

———. "Video: Do Remote LFAs Really Solve Microloops?" *Rule 11 Reader*, September 11, 2017. https://rule11.tech/video-remote-lfas-really-solve-microloops/.

Savage, Donnie, Steven Moore, James Ng, Russ White, Donald Slice, and Peter Paluch. *Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)*. Request for Comments 7868. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7868.txt.

Shimbel, A. "Structure in Communication Nets." In *Proceedings of the Symposium on Information Networks*. New York: Polytechnic Press of the Polytechnic Institute of Brooklyn, n.d., 199–203.

Review Questions

1. Explain the relationship between the calculation of shortest paths and loop-free paths through a network.
2. What are the conditions an alternate path must meet to be considered a Loop-Free Alternate?
3. Explain the difference between the waterfall and P/Q space models of understanding where loops will form using a network diagram containing seven routers in a ring and a single destination reachable through one of these routers.
4. When is an algorithm for solving the problem of loop-free paths called "greedy"?

5. Compare the state machine given in the chapter for DUAL to the state machine given in the EIGRP RFC. What is left out, what is combined, etc.? What are the advantages and disadvantages of having more or less detailed state machine diagrams? When would you prefer one or the other?
6. Draw a small network of around 10 or 11 nodes, and walk through the process of running the Bellman-Ford and Diffusing Update algorithms on it. Will DUAL find any Loop-Free Alternates in this network? Are there any places where a remote Loop-Free Alternate can be calculated?
7. In the network from question 6, assume a single link has failed; trace the reaction of DUAL to this event? Will queries be required? Why or why not?

This page intentionally left blank

Chapter 13

Unicast Loop-Free Paths (2)

Learning Objectives

After reading this chapter, you should be able to understand:

- Dijkstra's Shortest Path First algorithm
- The computation of Loop-Free Alternates using Dijkstra's algorithm
- Suurballe's Disjoint Path algorithm
- DFS numbering for calculating disjoint paths
- Maximally redundant trees for calculating disjoint paths

The preceding chapter discussed the shortest path rule and two algorithms (or perhaps systems) to find loop-free paths through a network. There is a wide range of such systems—far too many to cover in a few chapters of a larger book—but it is important for network engineers to be familiar with at least a few of these systems. This chapter considers Dijkstra's Shortest Path First, Path Vector, and two different disjoint path algorithms: Suurballe's and Maximally Redundant Trees (MRTs). Finally, this chapter will consider one other problem that control planes need to solve: ensuring two-way connectivity through the network.

Dijkstra's Shortest Path First

Dijkstra's Shortest Path First (SPF) algorithm is, perhaps, the most widely recognized and understood system for discovering loop-free paths through a network. It is

used by two widely deployed routing protocols, and in many other everyday systems such as software designed to find the shortest path through a road network, or to discover connections and connection patterns in social networks.

The History of Dijkstra's Algorithm

Edsger Dijkstra, a theoretical physicist, published his shortest path algorithm in 1959.[1] This greedy algorithm, in its original form, ran in the worst case in $O(n^2)$, but has been optimized through the use of self-balancing and ordered heaps, to $O(|E|+|V| \log |V|)$, where *E* is the number of edges (or links) in the network, and *V* is the number of vertices (or nodes). In most real-world networks, various assumptions are made that allow the algorithm to run faster than this, computing a tree across a large set of nodes and edges in tens to hundreds of milliseconds.

Dijkstra originally designed this algorithm as a "toy problem" to demonstrate the abilities of a 6-bit computer and then reused it to find the minimum amount of wire required to connect several computers. The algorithm is often considered a version of *Prim's Universal Shortest Path First* algorithm. Prim published a version in 1957, a version of which was apparently known by Jarnik in the 1920s.

This intertwined history of algorithms is common in the field of computer science; it is often difficult to unwind the true history of any particular algorithm, as many were discovered (in theory) by very early Greek or other classical mathematicians.

1. Dijkstra, "A Note on Two Problems in Connexion with Graphs."

Dijkstra's algorithm, in pseudocode, uses two data structures. The first is the tentative list, or the TENT; this list contains the set of nodes under consideration for inclusion in the Shortest Path Tree. The second is the PATH; this list contains the set of nodes (and therefore links, as well), which are on the Shortest Path Tree.

```
01 move "me" to the TENT
02 while TENT is not empty {

03   sort TENT
04   selected == first node on TENT

05   if selected is in PATH {
```

```
06     *do nothing*
07     }
08   else {
09     add selected to PATH

10     for each node connected to selected in TOPO
11       v = find node in TENT
12       if (!v)
13         move node to TENT
14       else if node.cost < v.cost
15         replace v with node on TENT
16       else
17         remove node from TOPO

18     }
19 }
```

As always, the algorithm is less complex than it appears on initial inspection; the key is the sorting of the two lists and the order in which nodes are processed off the TENT list. Here are some notes on the pseudocode before walking through an example:

1. The process starts with a copy of the topology database, called TOPO here; this will be clearer in the example, but it is simply a structure containing the source nodes, the destination nodes, and the cost of the link between them.
2. The TENT is the list of nodes that may, tentatively, be considered the shortest path to any particular node.
3. The PATH is the Shortest Path Tree (SPT), a structure containing a loop-free path to each node, and the next hop from "me" to that node.
4. The first crucial point in this algorithm is keeping only nodes already somehow connected to a node on the PATH list on the TENT; this means the shortest path on the TENT is the next shortest path in the network.
5. The second crucial point in this algorithm is the comparison between any existing nodes on the TENT that connect to the same node; this, combined with the sorting of the TENT and the separation of the TENT from the PATH, executes the shortest path rule.

With these points in mind, Figures 13-1 through 13-9 are used to illustrate the operation of Dijkstra's SPF algorithm.

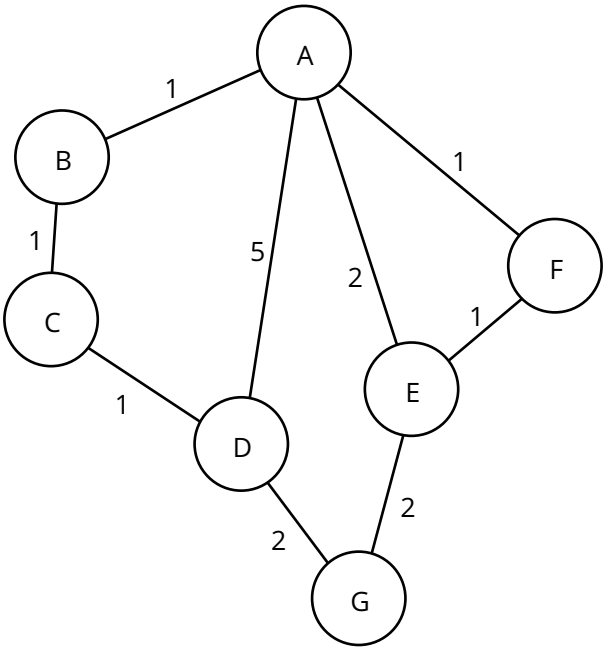
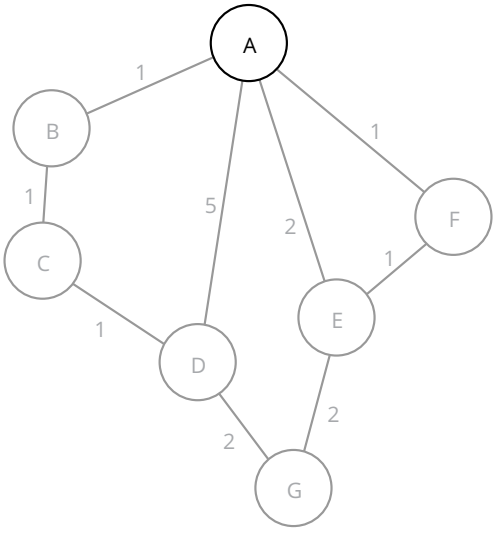


Figure 13-1 *A Small Network for Demonstrating Dijkstra's SPF Algorithm*

Each of the following illustrations, along with the accompanying description, will show one step in the SPF algorithm on this network, beginning with Figure 13-2.



TOPO

A	**A**	**0**
A	B	1
A	D	5
A	E	2
A	F	1
B	C	1
C	D	1
E	F	1
D	G	2
E	G	2

TENT

A	0

PATH

A	0

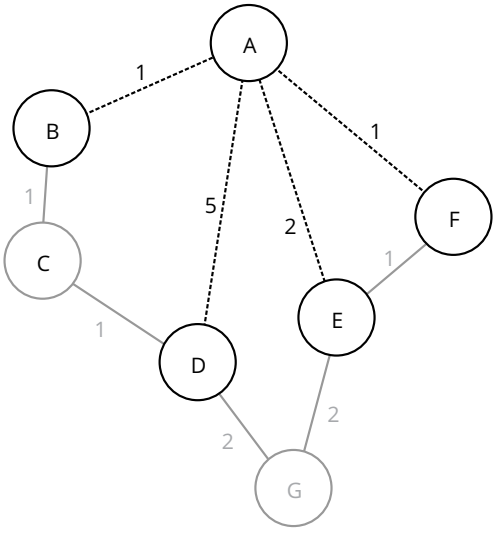
Figure 13-2 *The First Step in Dijkstra's SPF Calculation*

At the point illustrated in Figure 13-2, A has been moved from the TOPO into the TENT and then into the PATH. The cost of the origin node to itself is always 0; this link is included to start the SPF calculation. This represents lines 01 through 09 in the pseudocode shown earlier. Figure 13-3 illustrates the second step in the SPF calculation.

In Figure 13-3, each node connected to A has been moved from the TOPO to the TENT; this represents lines 10 through 17 in the pseudocode shown earlier. When this step began, there was only A in the TENT, so there are no existing nodes in the TENT that would have caused any metric comparisons. The TENT is now sorted, and execution continues with line 03 in the pseudocode. Figure 13-4 illustrates.

In Figure 13-4, one of the two shortest cost paths—to B and F, each with a cost of 1—has been chosen and moved to the PATH (lines 05–09 in the pseudocode shown earlier). When B is moved from the TENT to the PATH, any nodes with an origin of B in the TOPO are moved to the TENT (lines 10–17 in the pseudocode). Note C was not already in the TENT before being drawn on through B's move to the PATH, so no metric comparison is done. The cost to C is the sum of the cost of its predecessor in the PATH (which is B, with a cost of 1), and the link between the two nodes; hence C is added to the TENT with a cost of 2. The TENT is sorted (line 3 of the pseudocode), so the process is ready to begin again. Figure 13-5 illustrates the next step in the process.

In Figure 13-5, the shortest path on the TENT has been chosen, and F moved from the TENT to the PATH. There is a link between F and E (shown in previous illustrations as [E,F]), but the path through F to E is the same cost as the path [A,E], so this



TOPO

B	C	1
C	D	1
E	F	1
D	G	2
E	G	2	

TENT

B	1
F	1
E	2
D	5

PATH

A	0

Figure 13-3 *The Second Step in Dijkstra's SPF Calculation*

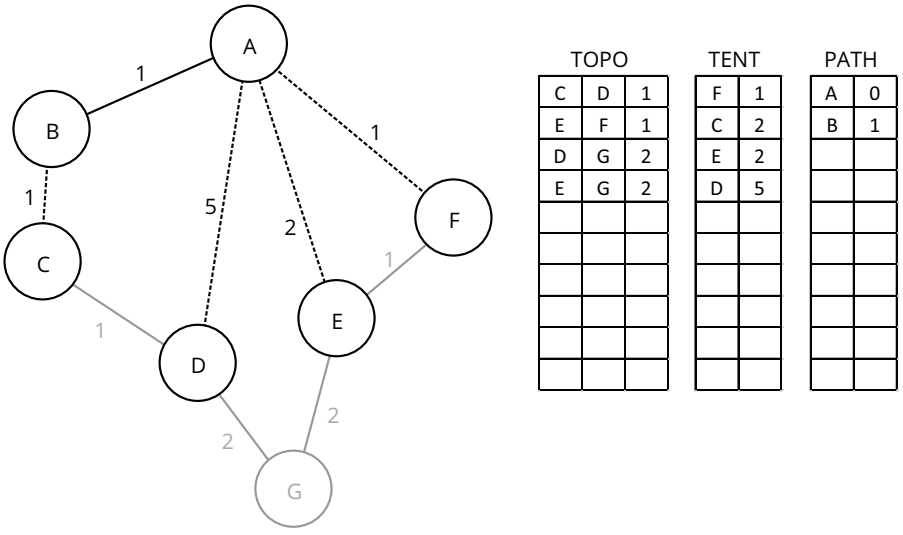


Figure 13-4 *The Third Step in Dijkstra's SPF Calculation*

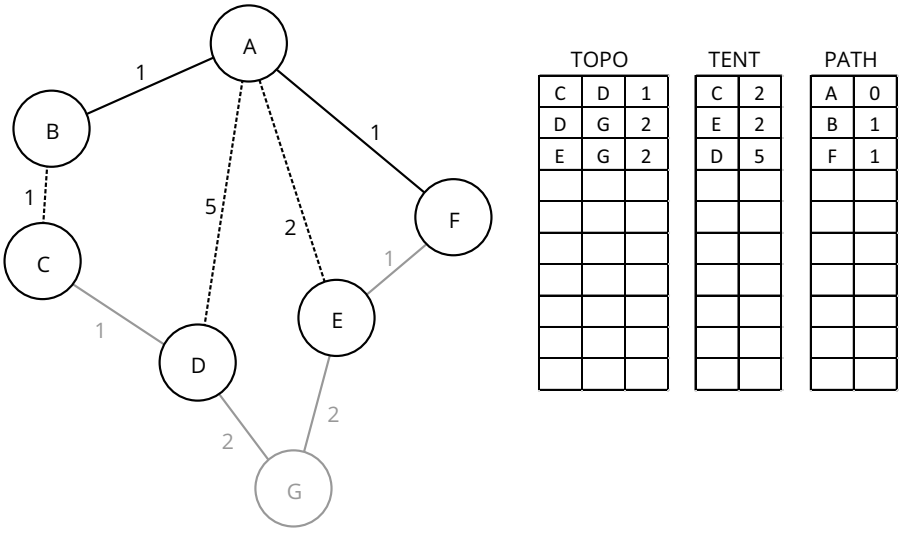
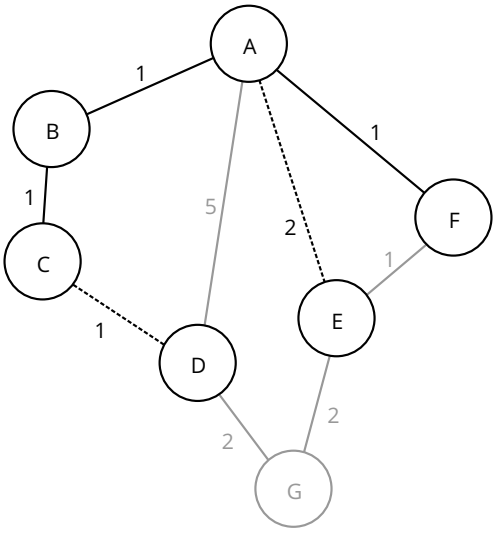


Figure 13-5 *The Fourth Step in Dijkstra's SPF Calculation*

link is not added to the TENT. Rather, it remains grayed out, as not being considered for inclusion in the SPT, and is removed from the TOPO. Figure 13-6 illustrates the next step in the process, which will move one of the metric 2 paths into the PATH.



TOPO

D	G	2
E	G	2

TENT

E	2
D	3

PATH

A	0
B	1
F	1
C	2

Figure 13-6 *The Fifth Step in Dijkstra's SPF Calculation*

Note

Most real-world implementations support carrying multiple equal cost paths from the TENT into the PATH, so they can forward traffic across all links with the same metric. This is called equal cost multipath, or ECMP. There are a number of different ways to accomplish this, but they are not covered here.

In Figure 13-6, the path to C through B, with a cost of 2, has been moved to the PATH, and the path to D through [A,B,C,D] has been moved to the TENT. In moving this path to the TENT, however, line 11 in the pseudocode finds an existing path to D on the TENT, the [A,D] path, with a cost of 5. The metric through the new path, 3, is lower than the metric through the existing path, 5, so the [A,D] path is removed from the TENT when the [A,B,C,D] path is added (line 15 in the pseudocode). Figure 13-7 shows the next step, where the remaining cost 2 link is moved from the TENT to the PATH.

In Figure 13-7, the path to E, with a cost of 2, has been moved from the TENT to the PATH. G has been moved to the TENT with a cost of 4 (the sum of [A,E] and [E,G]). E's other neighbor, F, is explored, but it is already on the PATH, so it is not considered for inclusion in the TENT. Figure 13-8 illustrates the next step, which moves D onto the PATH.

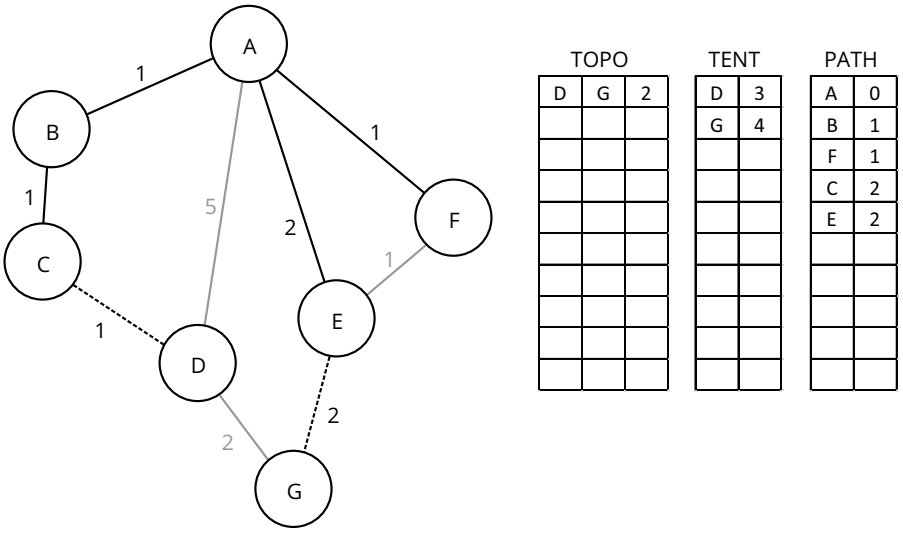


Figure 13-7 *The Sixth Step in Dijkstra's SPF Calculation*

In Figure 13-8, D, with a total cost of 3, has been moved from the TENT to the PATH. This brings D's neighbor, G—the last entry in TOPO—into consideration for the TENT. However, there is already a path to G with a total cost of 4 through [A,E,G], so line 14 in the pseudocode fails, and the path [D,G] is removed from the TOPO. This is the final SPT.

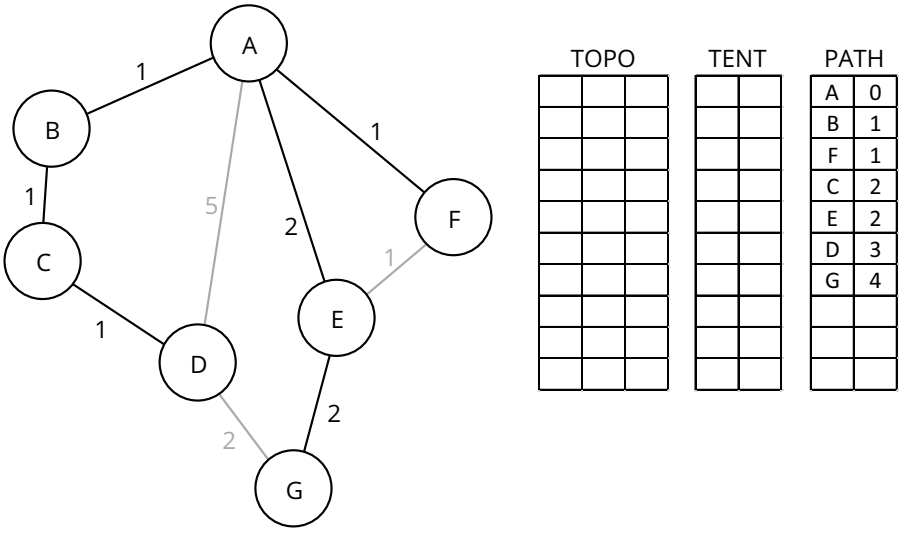


Figure 13-8 *The Seventh Step in Dijkstra's SPF Calculation*

The primary difficulty in understanding Dijkstra's algorithm is the shortest path rule isn't executed in one place (or on one router), as it is with Bellman-Ford or the Diffusing Update Algorithm (DUAL). The shortest path is (apparently) checked only when moving nodes from the TOPO to the TENT—but in reality, the sorting of the TENT itself executes another portion of the shortest path rule, and checking against the PATH for existing nodes constitutes another step in the process, making the process three steps:

1. If the path to the node is longer than any on the TENT, then the one on the TENT is a shorter path across the entire network.
2. A path that has risen to the top of the TENT through sorting is the shortest to that node in the network.
3. If the path moves to the PATH from the top of the TENT, it is the shortest path to that node in the network, and any other entries in the TOPO to that node should be discarded.

With the base algorithm in place, it is useful to look at some optimizations, and the calculation of Loop-Free Alternates (LFAs) and remote Loop-Free Alternates (rLFAs).

Partial and Incremental SPF

There is no particular reason that the entire SPT must be rebuilt each time there is a change to the network topology or reachability information; Figure 13-9 is used to explain.

Assume G loses its connection to 2001:db8:3e8:100::/64; device A does not need to recalculate its path to any of the nodes in the network. The reachable destination is just a leaf on the tree, even if it is a set of hosts connected to a single wire (such as an Ethernet). There is no reason to recalculate the entire SPT when a single

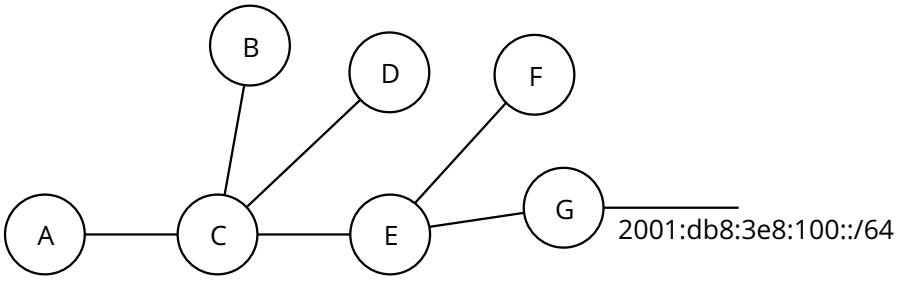


Figure 13-9 *Partial and Incremental SPF*

leaf (or any set of leaves) is disconnected from the network. In this case, only the leaf (the Internet Protocol [IP] address or the reachable destination) itself would need to be removed from the network (or rather, the destination can be removed from the database without any change to the network). This is a *partial* recalculation of the SPT.

Assume the [C,E] link fails. What does A do in this case? Again, there is no change to the topology of C, B, and D, so there is no reason for A to recalculate the entire tree. It is possible, in this case, for A to remove the entire tree beyond E. To compute just the changed portion of the graph, do the following:

- Remove the failed node and all nodes that A passes through E to reach.
- Recalculate the tree just from C's predecessor (in this case, A) to determine if there are alternate paths to reach nodes previously reachable through E before the [C,E] link failed.

This is called an *incremental* SPF.

Calculating LFAs and rLFAs

Chapter 12, "Unicast Loop-Free Paths (1)," considered the theory behind LFAs and rLFAs. Bellman-Ford does not calculate either downstream neighbors or LFAs, and does not appear to have the information required to do so. DUAL calculates downstream neighbors by default and uses them during convergence. What about protocols based on Dijkstra (and, by extension, similar SPF algorithms)? Figure 13-10 illustrates a simple mechanism that these protocols can use to find LFAs and downstream neighbors.

The definition of a downstream neighbor is one where the neighbor's cost to reach a destination is less than the local cost to reach the destination. From A's perspective:

- A knows the local cost to reach the destination, based on the SPT built by running Dijkstra's SPF.
- A knows B's and C's cost to reach the destination, by subtracting the cost of the [A,B] and [A,C] links from the locally calculated cost.

Hence, A can compare the local cost with the cost from each neighbor to determine if any neighbor is downstream in relation to any particular destination. The definition of an LFA is

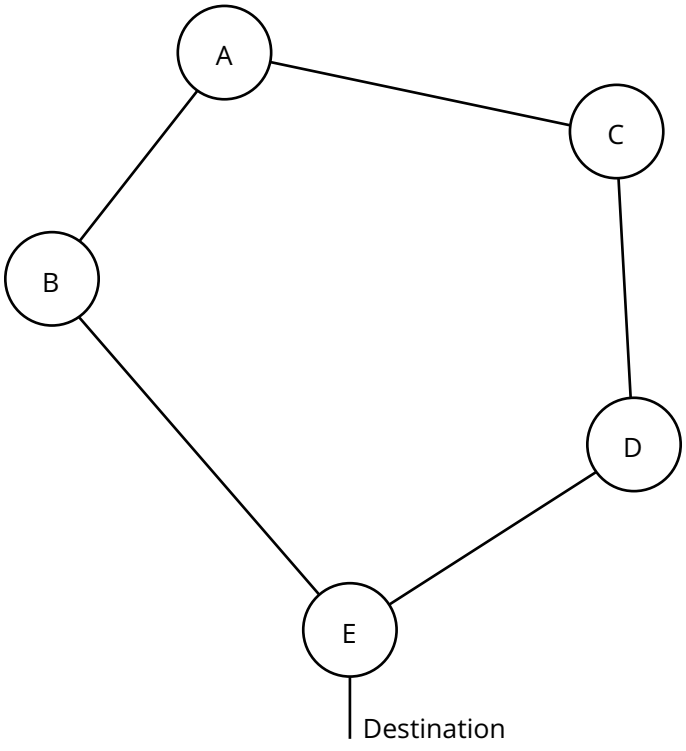


Figure 13-10 *Calculating LFAs and Downstream Neighbors with Dijkstra's Algorithm*

If the neighbor's cost to "me" plus the neighbor's cost to reach the destination is lower than the local cost, the neighbor is an LFA.

Or rather, given

- NC is the neighbor's cost to the destination.
- BC is the neighbor's cost to me.
- LC is the local cost to the destination.

If NC + BC < LC, then the neighbor is an LFA. In this case, A knows the cost of the [B,A] and [C,A] links from the perspective of the neighbor (it would be contained in the topology table, although it is not used in computing the SPT using Dijkstra's algorithm). So LFAs and downstream neighbors require very little additional work to calculate, but what about remote LFAs? The P/Q Space model

provides the simplest way for Dijkstra-based algorithms to compute downstream neighbors and LFAs. Figure 13-11 is used to illustrate from within the P/Q Space (see Chapter 12).

The definition of the P space is the set of nodes reachable from one end of the protected link, and the definition of Q space is the set of nodes reachable without traversing the protected link. This should suggest a moderately simple way to calculate these two spaces using Dijkstra:

Calculate an SPT from the perspective of the device connected to one end of the link; remove the link without recalculating the SPT. The remaining nodes are reachable from this end of the link.

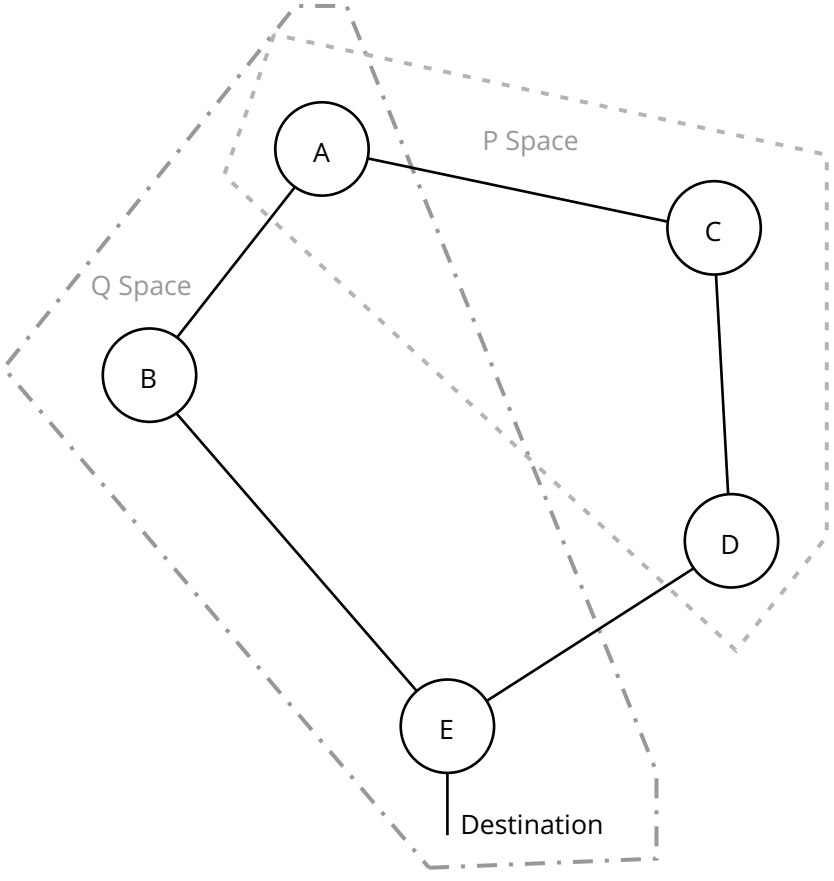


Figure 13-11 *P/Q Space and Calculating Remote LFAs with Dijkstra's Algorithm*

In Figure 13-11, E can

- Calculate the Q space by removing the [E,D] link from a copy of the local SPT, and all nodes that E uses D to reach.
- Calculate the P space by calculating an SPT from D's perspective (using D as the root of the tree), removing the [D,E] link, and then all nodes that D uses E to reach.
- Find the closest node reachable from both E and D with the [E,D] link removed.

Dijkstra's SPF is a versatile, widely used algorithm for computing Shortest Path Trees through a network.

Path Vector

Path vector relies on keeping a list of the nodes through which a path passes. Any node that receives an update with itself in the path will just discard the update, as it is not a viable path. Figure 13-12 is used for an example.

In Figure 13-12, each device advertises information about destinations to each neighboring device; for the destination attached to E:

1. E will advertise F with itself in the source, so with a path of [E], to both B and D.
2. From B:

   B will advertise F to A with a path of [E,B].
3. From D:

   D will advertise F to C with a path of [E,D].
4. From C:

   C will advertise F to A with a path of [E,D,C].

> **Note**
>
> Path vector was not developed as a theory or algorithm, but rather as a protocol; it is unique among the algorithms discussed here in this regard.

Which path will A prefer? In a path vector system, there can be a number of metrics, including the length of the path, policy preferences, etc. For instance, assume

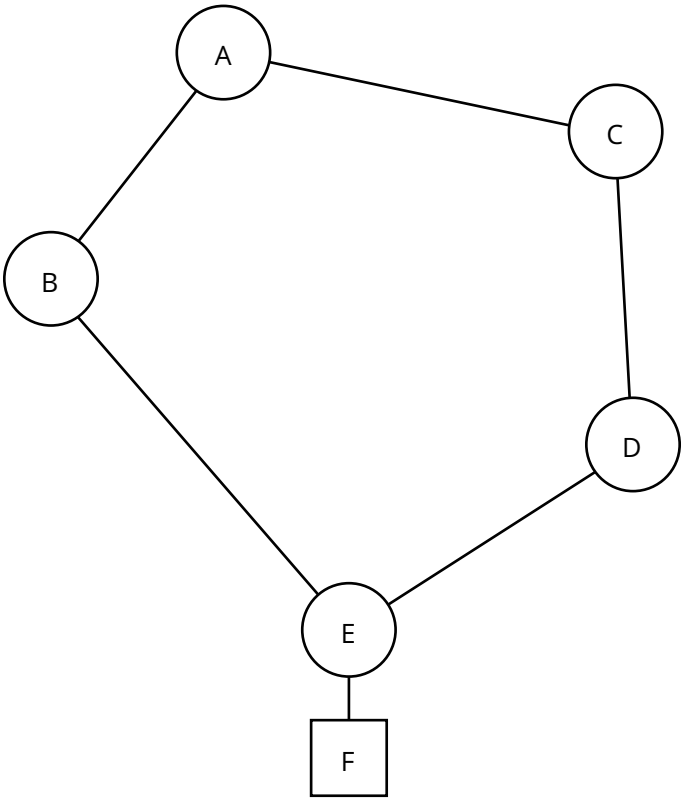


Figure 13-12 *Path Vector Operation Example*

there is a metric that is set locally at each node carried with each route. This local metric is carried between nodes but not summed in any way as it passes through the network, and each node can set this metric independently of the other nodes (so long as the node uses the same metric toward every neighbor). For instance, E's local metric is advertised to B, which then sets its own local metric for this destination and advertises the resulting route to A, etc.

To determine the best path, each node can then

- *Discard* any destination with the local node identifier in the path.
- *Compare the metric,* choosing the highest local metric among those it has received.
- *Compare the length of the path,* choosing the shortest path among those it has received.
- Advertise only the path being used to forward traffic.

Note

It does not matter if each node chooses the highest or the lowest metric; it only matters that each node does the same thing throughout the entire network. If comparing paths, however, the node must always choose the shorter path.

If every node in the network always follows these three rules, no loop will form. For instance:

- E advertises F to B with a path of [E] and a metric of 100.
- B advertises F to A with a path of [E,B] and a metric of 100.
- E advertises F to D with a path of [E] and a metric of 100.
- D advertises F to C with a path of [E,D] and a metric of 100.
- C advertises F to A with a path of [E,D,C] and a metric of 100.

A has two paths, both with the same metric, and hence will use the second rule to choose one, which is the shorter path. In this case, A will choose the path through [E,B]. A will advertise the route it is using toward C, but if C is following the same set of rules, it will also have two paths with a metric of 100 available, one with the path [E,B,A], and the second with a path of [E,D,C]. In this case, there must be a tie breaker that C uses internally to choose between the two routes. It isn't important what this tie breaker is, so long as it is consistently applied within the node; no matter which path C chooses, the traffic toward F will not loop.

Assume, however, a slightly different set of circumstances:

- E advertises F to B with a path of [E] and a metric of 100.
- B advertises F to A with a path of [E,B] and a metric of 100.
- E advertises F to D with a path of [E] and a metric of 50.
- D advertises F to C with a path of [E,D] and a metric of 50.
- C advertises F to A with a path of [E,D,C] and a metric of 50.

A has two paths, one with a metric of 100, and another with a metric of 50. Therefore:

- A will choose the higher of the two metrics, the path through [E,B], and advertise this route to C.

- C will choose the higher of the two metrics, the path through [E,B,A], and advertise this route to D.
- D will choose the higher of the two metrics, the path through [E,B,A,C], and advertise this route to E.
- E will discard this route, as E itself is already in the path.

Hence, even if the metric overrides the path length at (almost) every node, no loop will form.

The Multiple Metric Problem

Every algorithm discussed to this point has used a single metric to compute loop-free paths except path-vector, and path-vector uses two metrics in a very constrained way, with one always preferred over the other. The path, in fact, can be seen as a "tie breaker" that comes into play only when the primary metric, which does not relate to the path in any way (because it is not summed hop by hop in the network) fails to prevent a loop. Some protocols can use multiple metrics, but they will always combine these metrics in some way so only a single combined metric is used to find loop-free paths. Why?

In mathematical terms, all methods used to find a set of loop-free (or shortest) paths through a network are solvable in polynomial, or nonexponential, time—or rather, they are considered problems of the class *P.* There is a broader class of problems, containing *P,* that contains any problem solvable using a (theoretical) nondeterministic Turing machine. Among the NP problems, there is a set of problems considered NP-complete, which means there is no known efficient way to solve the problem; in other words, to solve the problem, every possible combination must be listed, and the best possible solution chosen from among this set.

The multiple metric problem is classified as NP-complete, and hence—while solvable—it is not solvable in any way lending itself to use in near-real-time communication networks.

Disjoint Path Algorithms

Consider the problem of a medical procedure executed by a robot following the hands of a live surgeon halfway across the world. It is possible that making such a system work requires packets to be delivered from the sensors on the surgeon's hands

to the robot in near real time, in order, with little or no jitter, and absolutely no packets can be dropped. This example, of course, can be expanded to many different situations, including financial systems and other mechanical control systems where near-real-time packet delivery with no failures is required.

What is often needed in these situations is to transmit two copies of each packet and then allow the receiver to choose the packet best fitting the Quality of Service (QoS) and packet loss characteristics needed to support the application. All of the systems discussed so far, however, can find only one loop-free path, and potentially an alternate path (an LFA and/or an rLFA). The problem being solved, then, by disjoint path algorithms, is this:

> How can paths be built through a network in such a way as to make certain they use the smallest number of overlapping resources (devices and links) as possible (hence are maximally disjoint, or maximally redundant)?

This section will begin by describing the concept of a two-connected network, and then consider two different (but seemingly related) ways of calculating disjoint topologies on two-connected networks.

Two-Connected Networks

A two-connected network is any network in which there are at least two paths between a source and destination that do not use the same devices (nodes) or links (edges). There are points to pay attention to here:

- A network is two-connected in relation to a specific set of sources and destinations; most networks are not two-connected for every source and every destination.
- Small blocks of any given network may be two-connected for some sources and destinations, and these blocks may be interconnected by narrow one- or two-connected *choke points*.

Note

Choke points will play a major role in many different areas of network design, a topic considered in Part III, "Network Design."

It is often easiest to understand two-connectedness through an actual example; Figure 13-13 shows a network marked out in blocks.

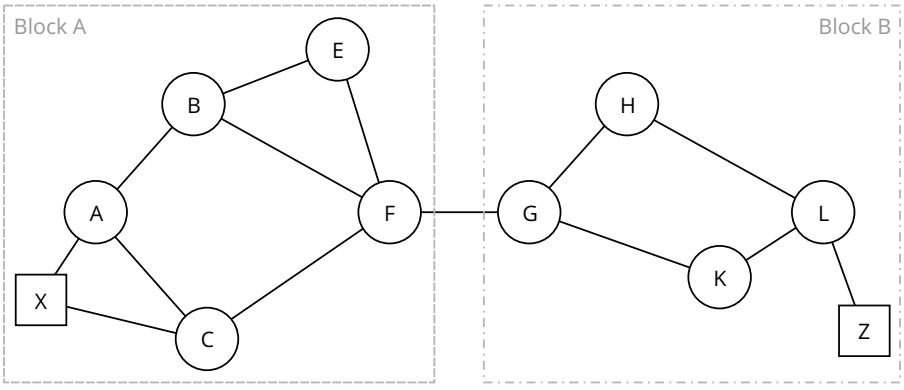


Figure 13-13 *Two-Way Connected Network Example*

In *block A,* there are at least two different disjoint paths between X and F:

- [X,A,B,E,F] and [X,C,F]
- [X,A,B,F] and [X,C,F]

In *block B,* there is one pair of disjoint paths from G to L: [G,K,L] and [G,H,L]. There are no disjoint paths to Z, as this node is *single connected*. There are also no disjoint paths between F and G, as these two are single connected. The [F,G] link can be considered a choke point between these two topology blocks. It is not possible, in the network illustrated in Figure 13-13, to compute two disjoint paths between X and Z.

Suurballe's Disjoint Path Algorithm

In 1974, J. W. Suurballe published a paper describing how to use multiple runs of Dijkstra's SPF algorithm to find multiple disjoint topologies in a network.[2] The algorithm essentially computes SPF once, removes a subset of the links in use on the SPT, and then computes a second SPF across the remaining links. Suurballe's algorithm is harder to explain than to illustrate in an example because of its reliance on the directional nature of the links computed through SPT; Figure 13-14 through Figure 13-18 are used as examples.

Figure 13-14 shows the state of the operations after the first SPF run has completed and the initial SPT is computed. Note the directional arrows on the links; it is

2. Suurballe, "Disjoint Paths in a Network."

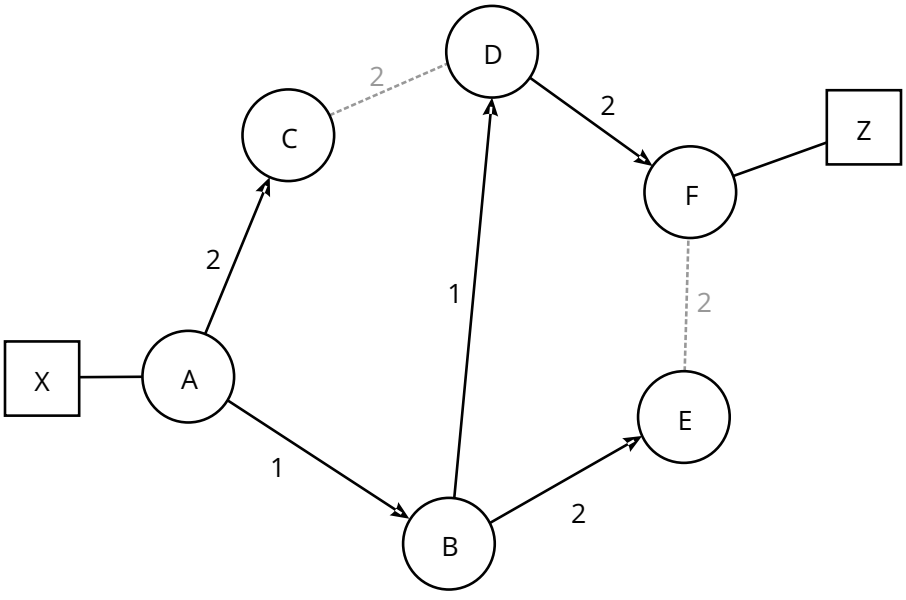


Figure 13-14 *Using Suurballe's Algorithm for Finding Disjoint Paths, Step 1*

not common to think about an SPT as being directional, but in reality it is, with each link oriented away from the source, or the root of the tree. When F computes a tree back toward X, it would also produce a directional tree with the arrows pointing in the opposite direction.

Edges (or links) on the SPT are called tree edges, and edges (or links) not on the resulting SPT are called nontree edges. In Figure 13-14, the tree edges are marked in solid black with directional arrows, and the nontree edges are lighter gray dashed lines.

The second step is shown in Figure 13-15.

Figure 13-15 shows each link with modified costs; each link that was a part of the original SPT (each tree edge, shown as a solid line) has two costs, one in each direction, while links not originally part of the SPT (nontree edges, shown as dashed lines) have their original costs. Note the arrows showing the direction of the cost in each case; this will be important in the next stage of the calculation. To calculate the costs of the two directional links for each tree edge:

1. Call one end of the link *u* and the other end of the link *v;* note the equation is being run in both directions.
2. Subtract the cost from the source to *v* from the cost of the link from *u* to *v*.
3. Add the cost from the source to *u*.

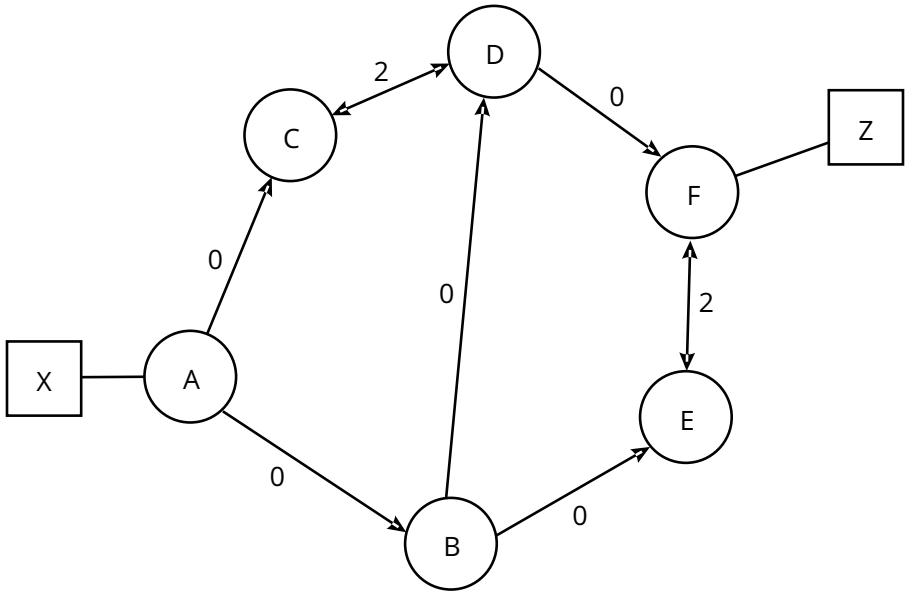


Figure 13-15 *Using Suurballe's Algorithm for Finding Disjoint Paths, Step 2*

If the source is *s:*

$d[sp](u,v) = d(u,v) - d(s,v) + d(s,u)$

This essentially sets the cost of tree edges to 0, as can be seen by doing the math for the [B,E] link:

- B is *u*, E is *v*, A is *s*
- $d(u,v) = 2$, $d(s,v) = 3$, $d(s,u) = 1$
- $2 - 3 + 1 = 0$

All of the nontree edges, however, will be set to some (generally larger) nonzero cost. For the network in Figure 13-15:

- For the [B,A] link (note [A,B] is not a link in the directional tree being calculated):

  B is *u*, A is *v*, A is *s*

  $d(u,v) = 0$, $d(s,v) = 0$, $d(s,u) = 1$

  $0 - 0 + 1 = 1$

- For the [E,B] link:

  E is *u*, B is *v*, A is *s*

  $d(u,v) = 2$, $d(s,v) = 1$, $d(s,u) = 3$

  $2 - 1 + 3 = 4$

- For the [C,A] link:

  C is *u*, A is *v*, A is *s*

  $d(u,v) = 2$, $d(s,v) = 0$, $d(s,u) = 2$

  $2 - 0 + 2 = 4$

- For the [F,D] link:

  F is *u*, D is *v*, A is *s*

  $d(u,v) = 1$, $d(s,v) = 4$, $d(s,u) = 5$

  $1 - 4 + 5 = 2$

- For the [D,B] link:

  D is *u*, B is *v*, A is *s*

  $d(u,v) = 1$, $d(s,v) = 1$, $d(s,u) = 2$

  $1 - 1 + 2 = 2$

The next step, shown in Figure 13-16, is to remove all the directional edges pointing toward the source that lies along the original SPT toward the specific destination (Z, in this case), reverse the direction of the zero-cost edges (links) along this same path, and then run Dijkstra's SPF again, creating a second SPT on the same topology.

Returning to the original SPT, the path from X to Z was along the path [A,B,D,F]. Hence, the four nonzero-cost edges (the dashed lines) pointing back toward the source, A, along this path have been removed. Along the same path, [A,B,D,F], the direction of each edge has been reversed; for instance, [A,B] originally pointed from A toward B and now points from B toward A. The next step is to run SPF across this graph, remembering *traffic cannot flow against the direction of the link*. The resulting tree is shown in Figure 13-17.

Figure 13-17 shows the original tree and the newly calculated tree overlaid on the original topology as two different dashed lines. The two topologies still share the

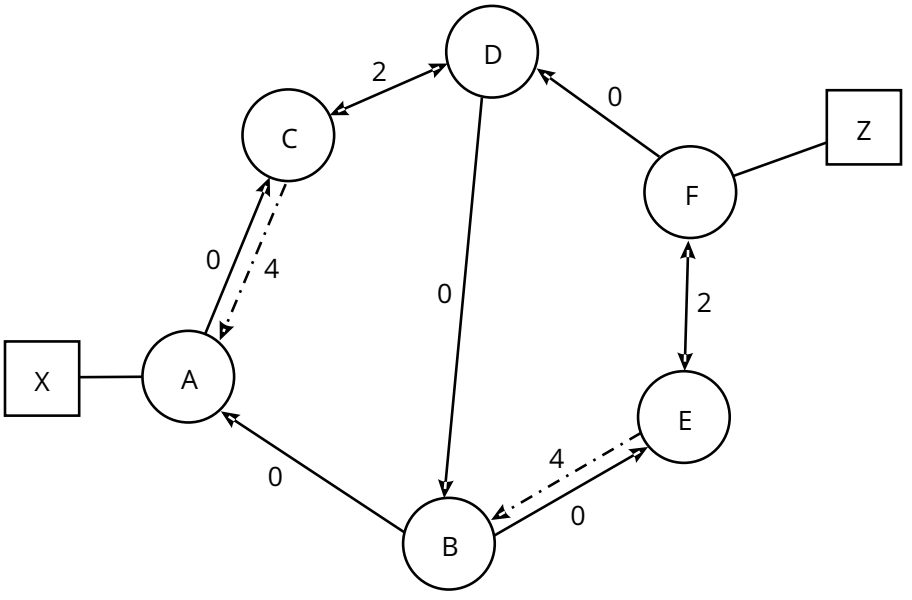


Figure 13-16 *Using Suurballe's Algorithm for Finding Disjoint Paths, Step 3*

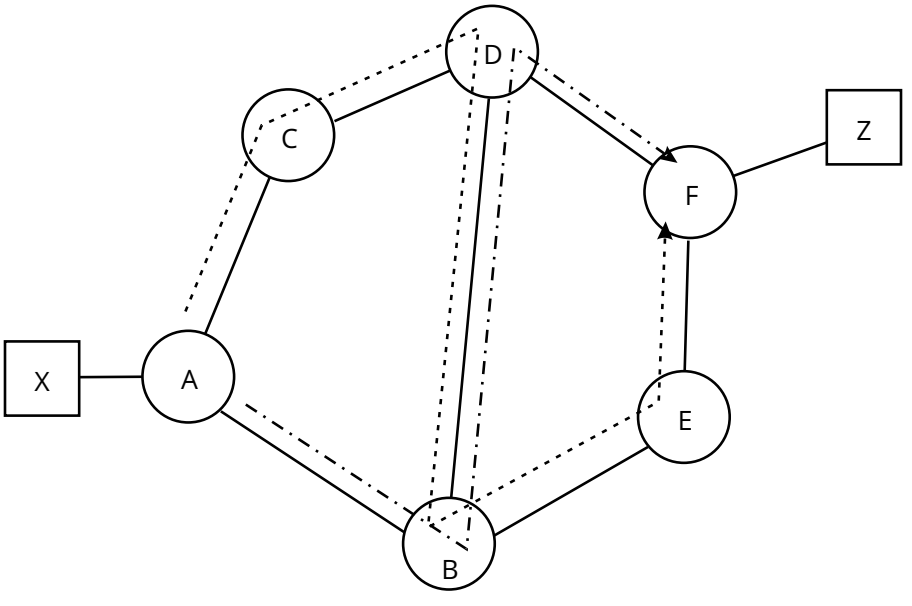


Figure 13-17 *Using Suurballe's Algorithm for Finding Disjoint Paths, Step 4*

[B,D] link in common, so they are not truly disjoint yet. At this point, there are two shortest paths from X to Z:

- [A,B,D,F]
- [A,C,D,B,E,F]

These two graphs are merged to form a set of edges, and any links that are included in both graphs, but in opposite directions, are discarded; the combined set looks like this:

[A->B, B->E, E->F, A->C, C->D, D->F]

Note the directionality of each link again—it is crucial to paring out the overlapping link, which would be listed both as [B->D] and [D->B]. With this subset of possible edges on the graph, it is possible to see the correct set of shortest paths are [A,B,E,F] and [A,C,D,F].

Suurballe's algorithm is complex, but shows the principal points of calculating disjoint trees—including how difficult they are to compute.

Maximally Redundant Trees

A simpler alternative to Suurballe's algorithm to calculate disjoint trees is computing Maximally Redundant Trees (MRTs). The best place to begin in understanding MRTs is with the humble Depth First Search (DFS), particularly the numbered DFS. Figure 13-18 is used as an illustration.

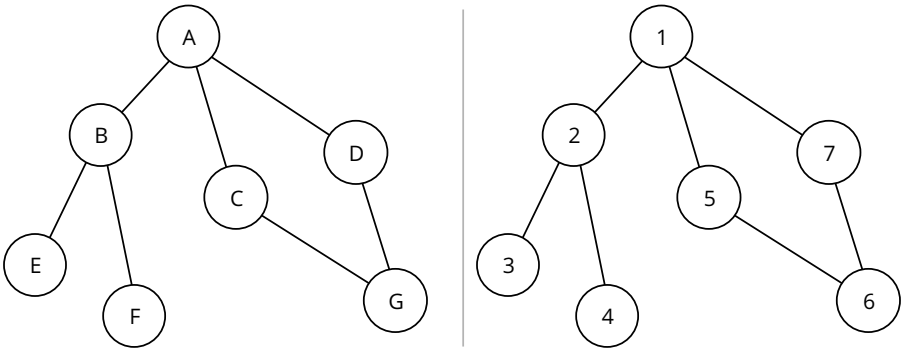


Figure 13-18 *A Depth First Search Tree Example*

In Figure 13-18, the left side represents a simple topology; the right, the same topology that has been numbered using a DFS. Assuming the DFS algorithm used to "walk" the tree always chooses the left node over the right, the process would look something like this:

```
01 main {
02   dfs_number = 1
03   root.number = dfs_number

04   recurse_dfs(root)
05 }

06 recurse_dfs(current) {

07   for each neighbor of current {
08     child = left most neighbor (not visited)
09     if child.number == 0 {
10       dfs_number++
11       child.number = dfs_number
12       if child.children > 0 {
13         recurse_dfs(child)
14       }
15     }
16   }
17 }
```

The best way to understand this code is to walk through the recursion a few times to see how it works. Using Figure 13-18:

- In the first call into *recurse_dfs,* A, or *root,* is set as the current node.
- Once inside *recurse_dfs,* the leftmost node of A is chosen, or B.
- B does not have a *number* when the loop is entered, so the *if* statement on line 09 is true.
- B is assigned the next DFS number (line 11).
- B has children (line 12), so *recurse_dfs* is called again with B as the current node.
- Once inside the (second level of) *recurse_dfs,* the leftmost neighbor of B is chosen, which is E.
- E does not have a DFS number, so the *if* statement on line 09 is true.
- E is assigned the next DFS number (3).

- E does not have children, so the processing winds back to the top of the loop.
- F is now the leftmost neighbor of B that has not been visited, so it is assigned to *child*.
- F does not have a number, so the *if* statement on line 09 is true.
- F is assigned the next DFS number (4).
- B has no more children, so the *for* loop at line 07 fails, and the *recurse_dfs* exits.
- However, *recurse_dfs* does not actually exit—it just "falls back" to the previous recursion level, which is line 14; this level of recursion is still processing A's neighbors.
- C is the next neighbor of A that has not been touched, so *child* is set to C.
- And so on.

Examining the numbers of the nodes on the right side of Figure 13-18 leads to the following interesting observations:

- If A always follows an increasing number to reach D, it will follow the path [A,C,G,D].
- If D always follows a decreasing DFS number to reach A, it will follow the path [D,A].
- These two paths are, in fact, disjoint.

This property holds for all topologies that have been assigned numbers through a DFS search: a path that follows always-increasing numbers will always be disjoint with a path that always follows decreasing numbers. This is precisely the property MRTs rely on to build disjoint paths. The problem with DFS numbering, however, is it is difficult to do in near real time. There must be some sort of elected *root,* traffic is suboptimal at a local level (much like a Minimum Spanning Tree, or MST, might be), and any changes to the topology require the entire DFS numbering scheme to be rebuilt.

To work around these problems, MRT builds disjoint topologies using the same principle but in a different way. Figure 13-19 is used to explain.

The first step in building an MRT is to find a short loop through the topology from a root (generally these loops are found using Dijkstra's SPF algorithm). In this case, A will be chosen as the root, and the loop will be [A,B,C,D]. This first loop will be used as the first of the two topologies, say the *red* topology. Reversing the loop to [A,D,C,B] generates a disjoint topology, say the *blue* topology. This first pair of topologies through this short loop is called an *ear.*

To expand the range of the MRT, a second *ear* is added to the first. To do this, a second loop is discovered, this time through [A,D,F,E,B], and the disjoint topology

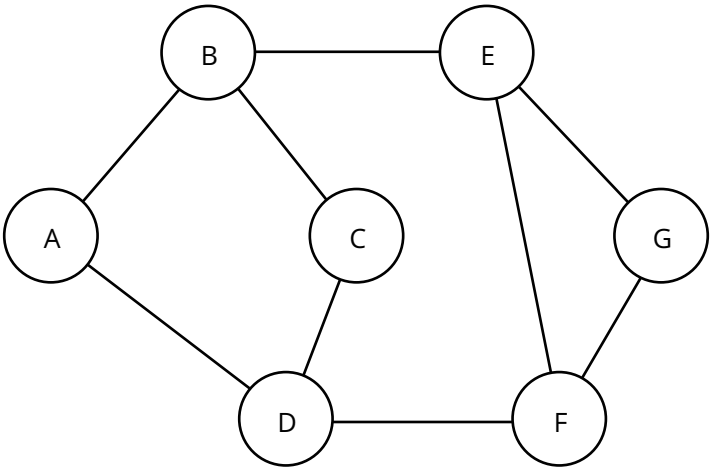


Figure 13-19 *Sample Network for Building an MRT*

is [A,B,E,F,D]. The question is: which of these two topology extensions should be added to the *red* topology, and which should be added to the *blue*? This is where a form of DFS numbering comes into play.

Each device in the network must already have an identifier assigned, either by the administrator, or through some other mechanism. These identifiers must be unique per device. Within the DFS numbering scheme there is also the concept of a *low point,* which indicates where on a particular tree this node attaches, and also what nodes attach to the tree through this node.

Given these unique identifiers and the ability to calculate a *low point,* each node in the network can be *ordered* just like it were given a number through a DFS numbering process. The key is to know how the ordering corresponds to the existing *red* and *blue* topologies. Assume B's low point is higher than C's, if the [A,B,C,D] topology is part of the *red* topology. For any other *ear* or loop in the topology, which passes through B and C, the direction of the *ear* in which B is less than C should be placed on the *red* topology. The loop in the opposite direction should be place on the *blue* topology.

This explanation is rather cursory, but it does give you the sense of how MRTs form disjoint topologies. Refer to the "Further Reading" section at the end of this chapter for more information on MRTs and their construction.

Two-Way Connectivity

This chapter and the preceding one have described a number of different ways to compute a loop-free path (or a set of disjoint paths) through a network. In each of these cases, the path computed is unidirectional—from the root of the tree to the

edges, or reachable destinations. It is, in fact, possible, for *no return path to exist*. In other words, a source may be able to reach a destination along a loop-free path, but there may be no return path from the destination to the source. This can be an uncommon failure mode in some link types, a result of filtering reachability information, or a number of other situations in the network.

Note

Two-way connectivity is not always desired; consider the case of a submarine, for instance, that needs to receive information about its current mission but cannot transmit any information without revealing its current position. The ability to send packets to devices located on the submarine, even though there is no two-way connectivity to them, would be desirable. Control planes either must be modified or specially designed to handle this kind of uncommon case, as the common case is for two-way connectivity to be required for proper network operation.

One other problem control planes must contend with in the area of computing paths is ensuring end-to-end two-way connectivity exists.

There are a number of ways a control plane can solve this problem:

- Some control planes just ignore this problem, which means they assume some other protocol, such as a transport protocol, will detect this condition.
- The control plane can check for this problem during route calculation. It is possible, for instance, when calculating routes using Dijkstra's algorithm, to perform a back link check while computing loop-free paths. Performing this back link check at each step of the computation can ensure two-way connectivity exists.
- The control plane can assume two-way connectivity between neighbors ensures end-to-end two-way connectivity. Control planes that perform explicit two-way connectivity checks on a per neighbor basis can (generally) safely assume any path through those neighbors is also capable of two-way communications.

Final Thoughts

These two chapters have covered a lot of ground, beginning with the shortest path rule and its importance in the process of computing loop-free paths through a network. Bellman-Ford, in its original form, was discussed next, then Garcia's DUAL.

Routing protocols built on these two protocols are considered *distance-vector* protocols, a term you will encounter in following chapters. Dijkstra's SPF was considered next; protocols built on this algorithm are considered *link state*. Then the path-vector solution was discussed, and finally disjoint paths.

Most of these algorithms can be used either by a distributed control plane or a centralized one. The primary point is to know how the loop-free path problem can be solved, so you can recognize it in its many forms and understand how it is being solved, no matter what protocol or controller you are looking at.

Further Reading

Chandra, Ravi, and John Scudder. *Capabilities Advertisement with BGP-4*. Request for Comments 5492. RFC Editor, 2009. doi:10.17487/rfc5492.

Chen, Enke, Tony J. Bates, and Ravi Chandra. *BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)*. Request for Comments 4456. RFC Editor, 2006. doi:10.17487/rfc4456.

Chen, Enke, John Scudder, Alvaro Retana, and Daniel Walton. *Advertisement of Multiple Paths in BGP*. Request for Comments 7911. RFC Editor, 2016. doi:10.17487/rfc7911.

Chen, Enke, and Quaizar Vohra. *BGP Support for Four-Octet AS Number Space*. Request for Comments 4893. RFC Editor, 2007. doi:10.17487/rfc4893.

Chunduri, Uma, Wenhu Lu, Albert Tian, and Naiming Shen. *IS-IS Extended Sequence Number TLV*. Request for Comments 7602. RFC Editor, 2015. doi:10.17487/rfc7602.

Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs." *Numerische Mathematik* 1, no. 1 (1959): 269–71. doi:10.1007/BF01386390.

Doyle, Jeff, and Jennifer DeHaven Carroll. *Routing TCP/IP, Volume 1*. 2nd edition. New Delhi, India: Cisco Press, 2005.

Ferguson, Dennis, Acee Lindem, and John Moy. *OSPF for IPv6*. Request for Comments 5340. RFC Editor, 2008. doi:10.17487/rfc5340.

Ginsberg, Les, Stephane Litkowski, and Stefano Previdi. *IS-IS Route Preference for Extended IP and IPv6 Reachability*. Request for Comments 7775. RFC Editor, 2016. doi:10.17487/rfc7775.

Heitz, Jakob, Keyur Patel, Job Snijders, Ignas Bagdonas, and Nick Hilliard. "BGP Large Communities." Internet-Draft. Internet Engineering Task Force, January 2017. https://tools.ietf.org/html/draft-ietf-idr-large-community-12.

"Intermediate System to Intermediate System Intra-Domain Routing Information Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service." Standard. Geneva: International Organization for Standardization, 2002. http://standards.iso.org/ittf/PubliclyAvailableStandards/.

Katz, Dave. "OSPF and IS-IS: A Comparative Anatomy." Presented at the NANOG19, Albuquerque, NM, June 12, 2000. https://nanog.org/meetings/abstract?id=1084.

McPherson, Danny R., and Keyur Patel. *Experience with the BGP-4 Protocol*. Request for Comments 4277. RFC Editor, 2006. doi:10.17487/rfc4277.

Meyer, David, and Keyur Patel. *BGP-4 Protocol Analysis*. Request for Comments 4274. RFC Editor, 2006. doi:10.17487/rfc4274.

Mirtorabi, Sina, Abhay Roy, Acee Lindem, and Fred Baker. "OSPFv3 LSA Extendibility." Internet-Draft. Internet Engineering Task Force, October 2016. https://tools.ietf.org/html/draft-ietf-ospf-ospfv3-lsa-extend-13.

Moy, John. "OSPF Version 2." Request for Comments 2328. RFC Editor, April 1998. doi:10.17487/RFC2328.

Parker, Jeff. *Recommendations for Interoperable Networks Using Intermediate System to Intermediate System (IS-IS)*. Request for Comments 3719. RFC Editor, 2004. doi:10.17487/rfc3719.

Przygienda, Dr. Antoni B. *Optional Checksums in Intermediate System to Intermediate System (ISIS)*. Request for Comments 3358. RFC Editor, 2002. doi:10.17487/rfc3358.

Ramachandra, Srihari S., and Yakov Rekhter. *BGP Extended Communities Attribute*. Request for Comments 4360. RFC Editor, 2006. doi:10.17487/rfc4360.

Raszuk, Robert, Christian Cassar, Bruno Decraene, Stephane Litkowski, Kevin Wang, and Erik Aman. "BGP Optimal Route Reflection (BGP-ORR)." Internet-Draft. Internet Engineering Task Force, January 2017. https://tools.ietf.org/html/draft-ietf-idr-bgp-optimal-route-reflection-13.

Rekhter, Yakov, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. Request for Comments 4271. RFC Editor, 2006. doi:10.17487/rfc4271.

Retana, Alvaro, and Russ White. "BGP Custom Decision Process." Internet-Draft. Internet Engineering Task Force, February 2017. https://tools.ietf.org/html/draft-ietf-idr-custom-decision-08.

Roy, Abhay, Yi Yang, and Alvaro Retana. *Hiding Transit-Only Networks in OSPF*. Request for Comments 6860. RFC Editor, 2013. doi:10.17487/rfc6860.

Shand, Mike, Stefano Previdi, Les Ginsberg, and Danny R. McPherson. *Simplified Extension of Link State PDU (LSP) Space for IS-IS*. Request for Comments 5311. RFC Editor, 2009. doi:10.17487/rfc5311.

Suurballe, J. W. "Disjoint Paths in a Network." *Networks* 4, no. 2 (1974): 125–45. doi:10.1002/net.3230040204.

Vohra, Quaizar, and Enke Chen. *BGP Support for Four-Octet Autonomous System (AS) Number Space*. Request for Comments 6793. RFC Editor, 2012. doi:10.17487/rfc6793.

Walton, Daniel, Alvaro Retana, Enke Chen, and John Scudder. *Solutions for BGP Persistent Route Oscillation*. Request for Comments 7964. RFC Editor, 2016. doi:10.17487/rfc7964.

Wang, Lili, Zhaohui (Jeffrey) Zhang, and Nischal Sheth. *OSPF Hybrid Broadcast and Point-to-Multipoint Interface Type*. Request for Comments 6845. RFC Editor, 2013. doi:10.17487/rfc6845.

"What Are the Differences between NP, NP-Complete and NP-Hard? Stack Overflow." Accessed September 24, 2017. https://stackoverflow.com/questions/1857244/what-are-the-differences-between-np-np-complete-and-np-hard.

White, Russ. *Intermediate System to Intermediate System (IS-IS) Routing Protocol LiveLessons*. Video. LiveLessons. Cisco Press, 2016. http://www.ciscopress.com/store/intermediate-system-to-intermediate-system-is-is-routing-9780134465326?link=text&cmpid=2017_02_02_CP_RussWhiteVideo.

White, Russ. "iSPF Versus PRC." *Rule 11 Reader*, June 7, 2017. https://rule11.tech/ispf-verse-prc/.

White, Russ, Danny McPherson, and Srihari Sangli. *Practical BGP*. Boston, MA: Addison-Wesley Professional, 2004.

White, Russ, and Alvaro Retana. *IS-IS: Deployment in IP Networks*. 1st edition. Boston, MA: Addison-Wesley, 2003.

Review Questions

1. Read through the additional material on DFS numbering systems. The concept of the low point was left out of the main text for brevity. Can you expand on this concept and the importance of finding the low point in determining disjoint paths through the network?

2. Compare the operation of Bellman-Ford and Dijkstra in a network with negative cost links; draw a small network of six or seven routers, set one of the links

so it has a negative cost in both directions, and determine the set of loop-free paths through the network using both algorithms. You do not need to run the algorithm to do this in a formal way; just describe which paths the Dijkstra algorithm will have a problem with and how the Bellman-Ford algorithm will react to these same paths.

3. Run MRT across the network shown in Figure 13-19 and show the resulting disjoint topologies.
4. Run Dijkstra's SPF across the network shown in Figure 13-19 from the perspective of A, using a cost of 1 for each link. Show the resulting shortest paths.
5. Using the network shown in Figure 13-19, assuming all link costs are 1, what would be the best path toward G? Would D have an LFA or rLFA in this network?
6. Is there any way you can think of to ensure connectivity exists with a unidirectional connectivity problem, such as the submarine example given in the note?

This page intentionally left blank

Chapter 14

Reacting to Topology Changes

Learning Objectives

After reading this chapter, you should be able to understand:

- The four steps of the convergence process
- Using polling and event-driven mechanisms for topology change detection
- Bidirectional Forwarding Detection
- Record-level replication and flooding
- Microloops
- Hop-by-hop control plane state distribution
- The CAP theorem and its interaction with control planes

You might have noticed that very few of the mechanisms described in Chapter 12 "Unicast Loop-Free Paths (1)," and Chapter 13, "Unicast Loop-Free Paths (2)," considered changes in the topology. Most of these solutions are focused on computing loop-free paths through an apparently stable network, as discovered by the mechanisms described in Chapter 11, "Topology Discovery." But what happens when the topology changes? Returning to the introduction of Part II:

> **How do network devices build the tables needed to forward packets along loop-free paths through the network?**

Now it is time to consider one more of the subproblems of this overarching problem:

How do control planes detect and react to changes in the network?

This question will be answered by examining two components of the convergence process in a control plane. The convergence process in a network can be described in four stages. Figure 14-1 is used for reference in describing these four stages.

Once the [C,E] link fails, the four stages that must occur are *detection, distribution, computation,* and *installation*.

1. **Detecting the change:** Whether the inclusion of a new device or link, or the removal of a device or link, regardless of the reason, the change must be detected by any connected devices. In Figure 14-1, devices C and E must detect the failure of the [C,E] link; when the link is brought back up, they must also detect the inclusion of this (apparently new) link in the topology.
2. **Distributing information about the change:** Each device participating in the control plane must learn about the topology change in some way. In Figure 14-1, devices A, B, and D must somehow be notified of the failure of the [C,E] link; when the link is brought back up, they must again be notified of the inclusion of this (apparently new) link in the topology.
3. **Computing a new loop-free path to the destination:** These algorithms are discussed in Chapters 12 and 13. In Figure 14-1, B and C must compute some alternate path to reach destinations behind E (or perhaps E itself).
4. **Installing the new forwarding information into the relevant local tables:** In Figure 14-1, B and C must install the newly computed loop-free paths to destinations beyond E into their local forwarding tables, so traffic can be switched along the new path.

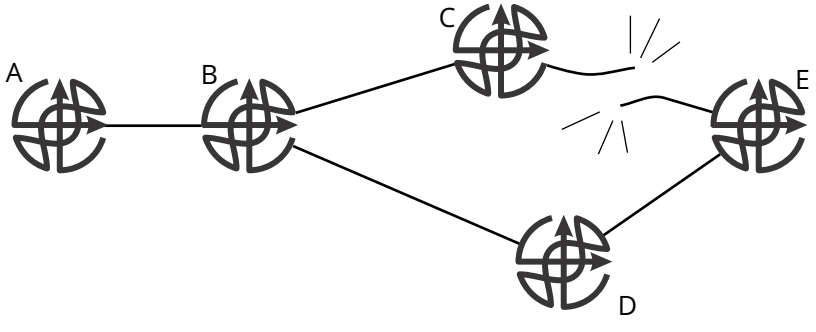


Figure 14-1 *The Convergence Process*

The following sections will focus on the first two of the four steps described in the preceding list, beginning with some thoughts on detecting topology changes. Some examples of protocols specializing in detecting topology changes will be considered. The distribution of topology and reachability information will take up the final half of this chapter. As this problem is, essentially, a distributed database problem, it will be addressed from that perspective.

Detecting Topology Changes

The first step in reacting to a change in the network topology is to detect the change. Returning to Figure 14-1, how should the two devices connected to the link, C and E, detect the link has failed? The solution to this problem is not as simple as it might first appear for two reasons: information overload and false positives.

Information overload occurs when the control plane receives so much information it simply cannot distribute information about topology changes, and/or compute and install alternate paths into the relevant tables at each device, fast enough to keep the state of the network consistent. In the case of quick, persistently occurring changes, such as a link disconnecting and connecting every few milliseconds, the control plane can be overwhelmed with information, causing the control plane itself to consume enough network resources to cause the network to fail. It is also possible for a series of failures to trigger a positive feedback loop, in which case the control plane "folds in" on itself, either reacting very slowly or failing altogether. The solution to information overload is to hide the true state of the topology from the control plane until the rate of change is within the bounds the control plane can support.

False positives are the second sort of problem; if a link drops one packet out of every 100, and the single packet dropped each time just happens to be a control plane packet used to monitor the link's state, the link will appear to go down and come back up (flap) quite frequently—even though other traffic is being forwarded across the link without problem.

There are two broad classes of solutions to the event detection problem:

- Implementations can send packets periodically to determine the state of a link, device, or system. This is *polling*.
- Implementations can trigger a reaction to a change in the state of a link or device off some physical or logical state within the system. This is *event driven*.

There are, as always, different tradeoffs with these two solutions, and subcategories of each one.

Polling to Detect Failures

Polling can be performed *remotely,* or *out of band;* or *locally,* or *in band;* Figure 14-2 illustrates.

In Figure 14-2, A and B are sending a *hello,* or some other form of polling packet, periodically across the same link they are connected through, and the same link across which they are forwarding traffic. This is *in band polling,* which has the advantage of tracking the state of the link over which traffic is being forwarded, reachability information is being carried, etc. On the other hand, D is polling A and B for some information about the state of the [A,B] link from another location in the network. For instance, D could be checking the state of the two interfaces on the [A,B] link on a periodic basis, or perhaps sending a packet along the [C,A,B,C] path on a periodic basis, etc. The advantage here is information about the state of a large number of links can be centralized, making network management and troubleshooting easier. Both kinds of polling are often used in real-world network deployments.

Polling mechanisms often use two separate timers to operate:

- A timer to determine how often the poll is transmitted; this is often called the polling interval in the case of out of band polling, and is often called the hello timer in the case of in band polling
- A timer to determine how long to wait before declaring a link or device down, or to raise some sort of alarm; this is often called a dead interval or dead timer in the case of in band polling

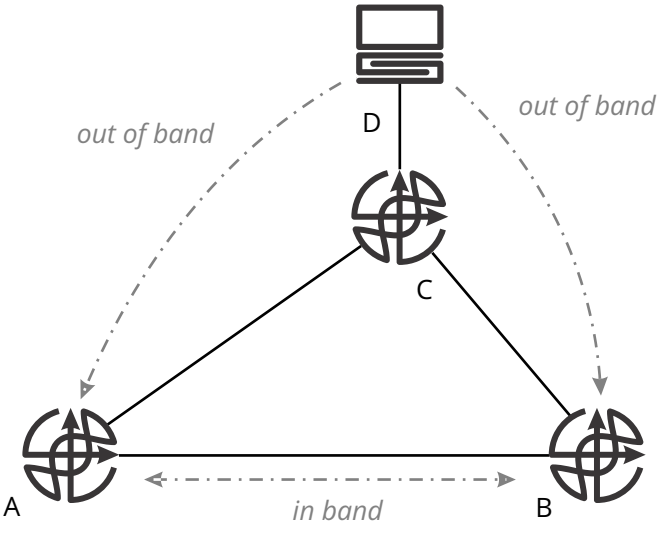


Figure 14-2 *In Band and Out of Band Polling*

The objective of in and out of band polling is often different. Out of band polling to discover changes in network state is often (but not always—specifically in the case of a centralized control plane) used to monitor the network state, and allows for centralized reactions to changes in state. In band polling is most often used (as you might expect) to detect changes in state locally, to drive the reaction of distributed control planes.

Event-Driven Failure Detection

Event-driven failure detection relies on some local, measurable event to determine the status of a particular link or device. Figure 14-3 illustrates.

> **Note**
>
> This is just an example; not all router implementations follow this model.

In Figure 14-3, which shows one possible implementation of the architecture elements between the physical interface and the routing protocol, there are four steps:

1. The link between the two physical interface (*phy*) chips located at either end of the link fails. Physical interface chips are normally optical to electrical handoffs. Most physical interface chips also perform some level of decoding on the inbound information, converting the individual bits on the wire to packets (deserialization), and packets into bits (serialization). Information is encoded by the physical interface onto a carrier, which is supplied by the two physical chips connected to the physical media. If the link fails, or one of the two interfaces is disconnected for any reason, the physical interface chip on the other end of the link will see the carrier drop in near real time—usually based on the speed of light and the length of the physical media. This condition is called loss of carrier.
2. The physical interface chip will, on detecting loss of carrier, send a notification toward the routing table (RIB) on the local device. This notification normally starts life as an interrupt, which is then translated into some form of Application Programming Interface (API) call into the RIB code, which results in the

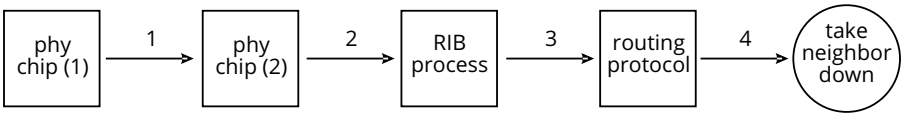


Figure 14-3 *Event-Driven Detection Example*

routes reachable through the interface, and any next hop information through the interface, being marked stale or being removed from the routing table. This signal may, or may not, pass through the Forwarding Information Base (FIB) along the way, depending on the implementation.

3. The RIB will notify the routing protocol about the routes it just removed from the local table based on the interface down event.
4. The routing protocol can then remove any neighbors reachable through the indicated interfaces (or rather through the connected routes).

There is no point in Figure 14-3 in which there is a periodic process checking the state of anything, nor are there any packets moving across the wire. The entire process is based on the physical interface chip losing carrier on the connected media; hence this process is event driven.

It is often the case that event-driven and polled status are combined. For instance, in Figure 14-3, if there were a management station polling the status of the interface in the local RIB on a periodic basis, the process from the physical interface chipset to the RIB would be event driven, while the process from the RIB to the management station would be driven by polling.

Comparing Event-Driven and Polling-Based Detection

Table 14-1 summarizes the advantages and disadvantages of each event detection mechanism.

Table 14-1 *Comparison of Polling and Event-Driven Detection*

	Out of Band Polling	In Band Polling	Event Driven
Status Distribution	Status is driven from a centralized system; the centralized system has a *bigger picture* view of the overall network state	Status is driven by local devices; gathering a *bigger picture* view of the state of the entire network requires gathering information from each individual network device	Status is driven by local devices; gathering a *bigger picture* view of the state of the entire network requires gathering information from each individual network device

Ties forwarding state to link or device state	Link and/or device state can be falsely reported; does not directly test forwarding capability	Link and/or device state can be directly tied to forwarding capability (barring failures within the state checking mechanism)	Link and/or device state can be directly tied to forwarding capability (barring failures within the state checking mechanism)
Speed of Detection	Must have some waiting interval before declaring a link or device failed to prevent false positives; slows reporting of network changes	Must have some waiting interval before declaring a link or device failed to prevent false positives; slows reporting of network changes	Some timer before reporting failures might be desirable to reduce the reporting of false positives, but this timer can be very short, and backed with a double-check of the state of the system itself; generally much faster at reporting network changes
Scaling	Must transmit periodic polls, consuming bandwidth, memory, and processing cycles; scales within these limits	Must transmit periodic polls, consuming bandwidth, memory, and processing cycles; scales within these limits	Small amounts of current local state; tends to scale better than polling mechanisms

While it may appear event-driven detection should always be favored, there are some specific situations where polling can solve problems that event-driven mechanisms cannot. For instance, one of the main advantages of polling-based systems, particularly when deployed in band, is to "see" the state of otherwise invisible boxes. For instance, in Figure 14-4, there are two routers connected through a third device, identified as a repeater in the illustration.

In Figure 14-4, device B is a simple physical repeater; whatever it receives on the [A,B] link it retransmits, just as it received it, on the [B,C] link. There is no control plane of any sort running on this device (at least not that A and C are aware of). Neither A nor C can detect this device, as it does not change the signal in any way A or C could measure.

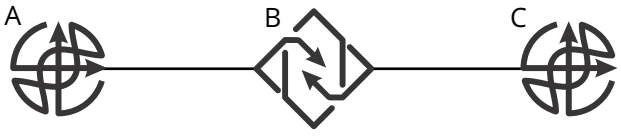


Figure 14-4 *Signal Repeaters and Loss of Carrier*

What happens if the [A,B] link fails if A and B are using an event-driven mechanism to determine link state? A will lose carrier, of course, because the physical interface at B will no longer be reachable. However, C will continue to receive carrier and hence will not detect the link failure at all. If it is possible for A and C to somehow communicate with B, this situation can be resolved. For instance, if B tracks all the Address Resolution Protocol (ARP) requests it receives, it can, when the [A,B] link fails, somehow send an "inverse ARP" notifying B that A is no longer reachable. The other solution available in this situation is some sort of polling between A and C that verifies reachability across the entire link, including the state of B (even though A and C are not aware that B exists).

From a complexity perspective, event-driven detection increases the interaction surfaces between the systems in a network, while polling tends to keep state within a system. In Figure 14-3, there must be some sort of interface between the physical interface chipset, the RIB, and the routing protocol implementation. Each of these interfaces represents a place where information that might be better hidden through an abstraction is transferred between systems, and an interface that must be maintained and managed. Polling, on the other hand, can often be contained within a single system, completely ignoring the underlying mechanisms and technologies in place.

An Example: Bidirectional Forwarding Detection

It will be useful, at this point, to spend a few pages examining an example of a protocol designed specifically to detect link state in a network. Neither of these protocols is part of a larger system (such as a routing protocol), but rather interact with other protocols through programming interfaces and status indicators.

Bidirectional Forwarding Detection (BFD) is grounded in a single observation: there are many control planes running on a typical network device, each with its own failure detection mechanism. It would be more efficient to run a single shared detection mechanism among all the different control planes. In most applications, BFD does not replace existing hello protocols used in each control plane, but rather augments them. Figure 14-5 illustrates.

In the BFD model, there are likely to be *at least* two different polling processes running over the same logical link (there could be more, if there are logical links layered on top of other logical links, as BFD can be used across various network

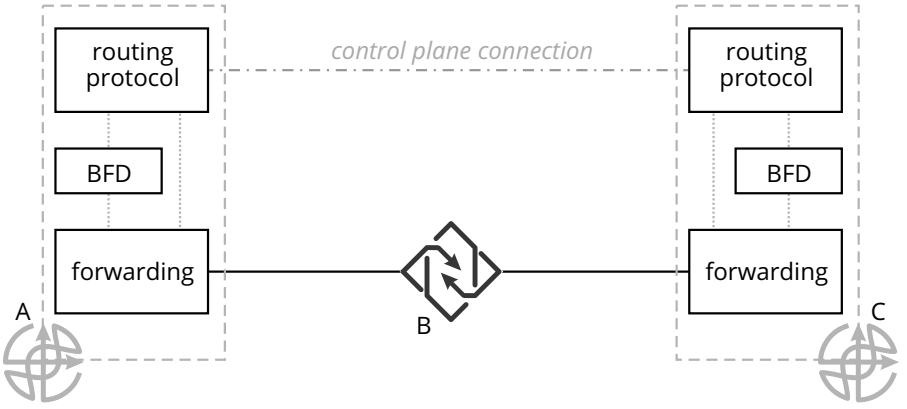


Figure 14-5 *Bidirectional Forwarding Detection*

virtualization technologies, as well). Control plane polling will use hellos to discover adjacent devices running the same control plane process, to exchange capabilities, determine the Maximum Transmission Unit (MTU), and, finally, to make certain the control plane process on the adjacent device is still running. These hellos are run across the *control plane connection* in Figure 14-5, which can be seen as a sort of "virtual link" passing through the physical link.

BFD polling will run *underneath* the control plane connection, as shown, verifying the operation of the physical connection and forwarding planes on the two connected devices. This two-layered approach allows BFD to operate much more quickly, even as a polling mechanism, than any routing protocol-based detection mechanism.

BFD can operate in four distinct modes:

- **Asynchronous mode:** In this mode, BFD acts like a lightweight hello protocol. The BFD process at A, potentially running on a distributed process (or even in an Application-Specific Integrated Circuit [ASIC]), sends hello packets to C; the BFD process at C acknowledges these hello packets. This is a fairly traditional use of polling through hellos.
- **Asynchronous mode with echo:** In this mode, the BFD process in A will send hello packets to C so the hello packets will be processed only through the forwarding path, hence allowing only the forwarding path to be polled. To accomplish this, A sends hello packets to C formed in such a way that they will be forwarded back to A. For instance, A can send a packet to C with A's own address as the destination; C can pick this packet up and forward it back to A. In this mode, the hellos transmitted by A are completely different from the hellos transmitted by C; there is no acknowledgment, just the two systems sending independent hellos that test the link bidirectionally from each end.

- **Demand mode:** In this mode, the two BFD peers agree to send hellos just when connectivity needs to be validated, rather than periodically. This is useful in the case where there is some other way to determine link status—for instance, if the [A,C] link is an Ethernet link, which means carrier detect is available to detect link failure—but when the alternate method is not necessarily trusted to provide accurate connectivity status in all situations. For instance, in the case of "switch in the middle," where B is disconnected from A but not C, C could send a BFD hello on noting any problem with the connectivity to verify its connection with A is still good. In demand mode, some event, such as a lost packet, can cause a local process to trigger a BFD detection event.
- **Demand mode with echo:** This mode is like demand mode—regular hellos are not transmitted between the two devices running BFD. When a packet is transmitted, it is sent in such a way as to cause the other device to forward the hello packet back to the sender. This reduces the amount of processor load on both devices, allowing much faster timers to be used for BFD hellos.

Regardless of the mode of operation, BFD calculates different polling (hello) and detection (dead) timers separately across the link. The best way to explain the process is through an example. Assume A sends a BFD control packet with a proposed polling interval of 500ms, and C sends a BFD control packet with a proposed polling interval of 700ms. The higher number, or rather the slower polling interval, is chosen for the relationship; the rationale for this is the slower system must be able to keep up with the polling interval to prevent false positives.

The polling rate is modified in actual use to prevent synchronization of hello packets across multiple systems on the same wire. If there were four or five systems deploying the Border Gateway Protocol (BGP) on a single multiaccess link, and every system sets its timer to send the next hello packet based on the receipt of the last packet, it is possible for all five systems to synchronize their hello transmission so all the hellos on the wire are transmitted at precisely the same moment. Since BFD normally operates with timers less than one second in length, this could result in a device receiving hellos from multiple devices at the same time, and not being able to process them quickly enough to prevent a false positive.

The specific modification used is to jitter the packets; each transmitter must take the base polling timer and subtract some random amount of time that is between 0% and 25% of the polling timer. For instance, if the polling timer is 700ms, as in the example given, A and C would transmit each hello packet sometime between around 562 and 750ms after the transmission of the last hello.

The final point to consider is the amount of time A and C will wait before declaring the link (or neighbor) down. In BFD, each device can calculate its own dead timer, normally expressed as a multiple of the polling timer. For instance, A could choose to consider the link (or C) down after two BFD hellos are missed, while C might decide to wait for three BFD hellos to be missed.

Change Distribution

Once a change in the network topology has been detected, it must be distributed in some way to all the devices participating in the control plane. Each item in a network topology can be described as either

- A link, or edge, including the nodes or reachable destinations attached to this link
- A device, or node, including the nodes, links, and reachable destinations connected to this device

This rather restricted set of terms lends itself to being held in a table, or database, often called the *topology table* or *topology database*. The question of distributing changes in the network topology to all the devices participating in the control plane, then, can be described as the process of distributing changes to specific rows in this table or database throughout the network.

The way in which information is distributed through a network depends on the design of the protocol, of course, but there are three commonly used kinds of distribution: hop-by-hop distribution, flooded distribution, and a centralized store of some sort.

Flooding

In flooding, each device participating in the control plane receives, and stores, a copy of every piece of information about the network topology and reachable destinations. While there are a number of ways to synchronize a database, or table, only one is normally used in control planes: record-level replication. Figure 14-6 illustrates.

In Figure 14-6, each device will flood the information it knows to each neighbor, who will then reflood the information to each neighbor. For instance, A knows two specific things about the network topology: how to reach 2001:db8:3e8:100::/64 and how to reach B. A floods this information to B, which, in turn, floods this information to C. Each device in the network ultimately ends up with a copy of all the topology information available; A, B, and C have *synchronized* topology databases (or tables).

In Figure 14-6, C's connectivity to D is shown as an item in the database; not all control planes would include this information. Instead, C may just include connectivity to the 2001:db8:3e8:102::/64 range of addresses (or subnet), which contains D's address.

Note

In larger networks, it is impossible for the entire description of a device's connections to fit into a single MTU-sized packet, and connection information needs to be timed out and reflooded on a regular basis to ensure freshness.

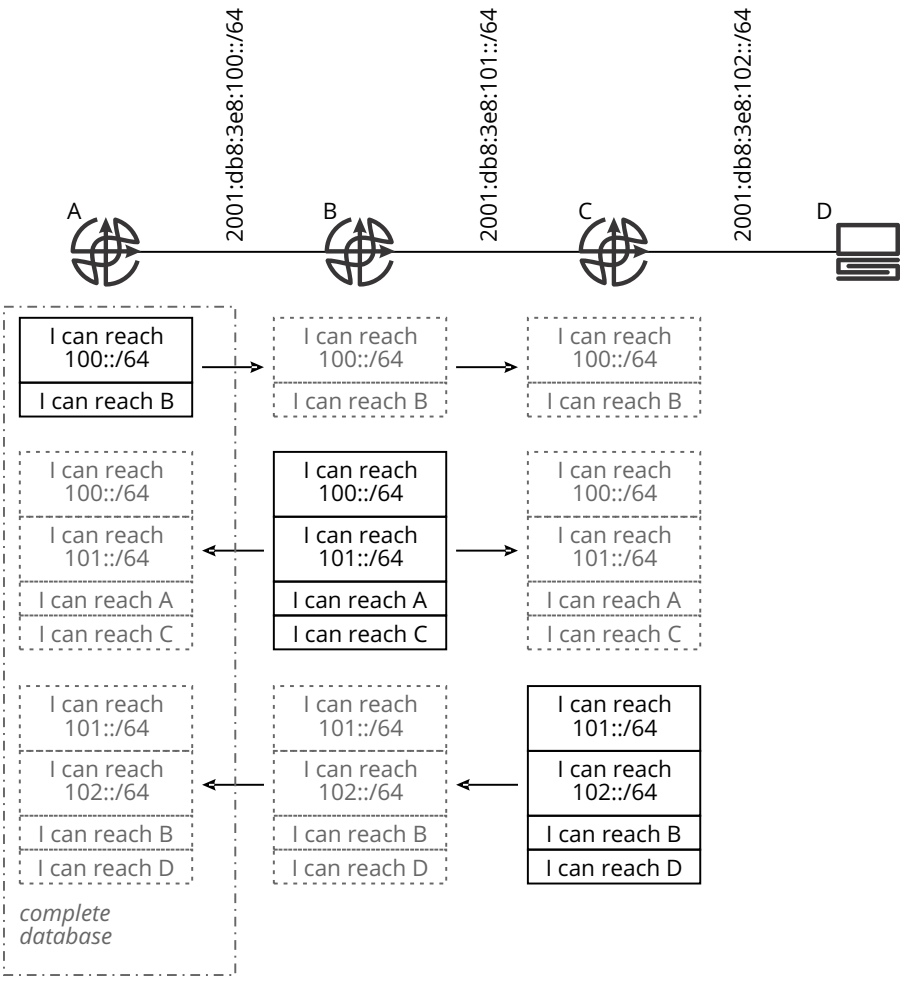


Figure 14-6 *Flooding Between Network Devices*

An interesting problem arises in flooded distribution mechanisms that can cause temporary routing loops, called microloops; Figure 14-7 illustrates.

In Figure 14-7, assume the [E,D] link fails. Consider the following chain of events, including some roughly possible times for each event:

1. **Start:** A is using E to reach D; C is using D to reach E.
2. **100ms:** E and D discover the link failure.
3. **500ms:** E and D flood information about the topology change to C and A.

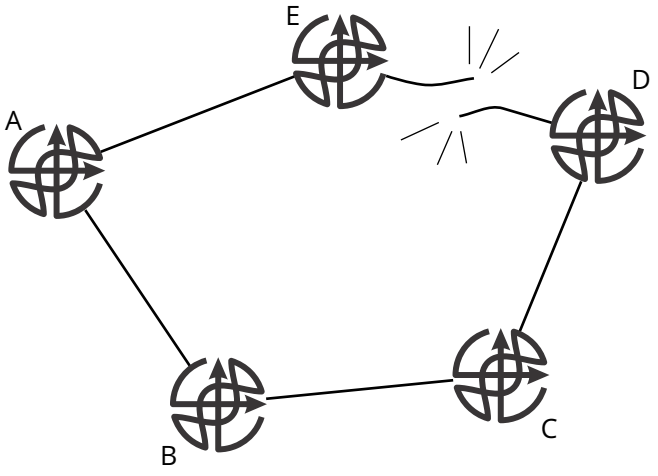


Figure 14-7 *Microloops in Flooded Database Distribution*

4. **750ms:** C and A receive the updated topology information.
5. **1,000ms:** E and D recompute their best paths; E selects A as its best path to reach D, D selects C as its best path to reach E.
6. **1,250ms:** A and C flood information about the topology change to B.
7. **1,400ms:** A and C recompute their best paths; A selects B to reach D, C selects B to reach E.
8. **1,500ms:** B receives the updated topology information.
9. **2,000ms:** B recomputes its best paths; it chooses C to reach D, and A to reach E.

While the times and ordering might vary slightly in any particular network, the ordering of discovery, advertisement, and recomputing will almost always follow a similar pattern. In this example, a microloop forms between steps 5 and 7; for 400ms, A is using E to reach D, and E is using A to reach D. Any traffic entering the ring at either A or D during the time between E's recalculation of the best path to D and A's recalculation of the best path to D *will loop*. A more formal definition of this problem will be considered in the later section, "Consistency, Accessibility, and Partitionability." One solution to this problem is to precompute Loop-Free Alternates or remote Loop-Free Alternates (both discussed in discussed in Chapter 13).

Microloops and Ordered FIB

If the routers receiving new topology information and calculating new best paths could be induced to order the installation of the new best paths, the microloop can be avoided. In Figure 14-7, if C could somehow be configured to wait to install its new best path until B has computed and installed its new best path, and if D could be configured to wait to install its new best path until C has computed and installed its new best path, the microloop can be avoided. In other words, one way to avoid the microloop is to somehow ensure the Forwarding Information Base (FIB) entries in the network devices are installed in the order B->(C,A)->(D,E).

There is actually a technique for computing the order of installs called *ordered FIB,* or *oFIB.*[1] This mechanism assumes some form of Shortest Path First (SPF) algorithm, such as Dijkstra's, to compute the best path. To compute the order in which the FIB entries should be installed to avoid the loop, begin with the assumption that all the new topology information available will be received on every network device before the best path calculation is undertaken by any network device. This requires measuring the largest (typical) amount of time flooding updated topology information through the network, and setting some timer so that no network device will calculate a new best path until the timer has expired.

Once all the topology information is received, each device calculates two best paths: one from the local node based on the new topology information and one from the *affected node* using the topology information from before the change. The affected node is either the end of the failed link farthest from the local node (D, from A's perspective in Figure 14-7), or the actual node that has failed. This can be discovered by examining the report of the failure from both of the devices connected to the failure point (D and E in Figure 14-7).

The number of devices relying on the local node to forward to the affected device can be computed using the shortest path from the perspective of the affected router. For the network in Figure 14-7, A and B rely on E to reach D, and B relies on A to reach D. Given this information, the device can wait to install any new forwarding information until after it is certain any devices that once

1. Bryant et al., Framework for Loop-Free Convergence Using the Ordered Forwarding Information *Base (oFIB) Approach*.

depended on it—who might still be using the local device to forward traffic to the affected device—have installed their new forwarding information. To do this, an installation wait time (called *wait-time* here) is agreed on throughout the network (before the failure!). This wait time is how long, per dependent node, any device should wait before installing any new forwarding information after a failure. It is best, of course, to base this wait time on real measurements of how long it takes to compute and install a new best path in the slowest node in the network.

Given this wait time, E can note that two devices were (potentially) forwarding traffic to E to reach D. Hence, when the [D,E] link fails, E should wait *2*wait-time* before installing any new path to D. A can note that one device was (potentially) forwarding traffic to A to reach D; hence it should wait *1*wait-time* before installing any new path to D. This process will ensure the new forwarding information is installed in the correct order to prevent the microloop.

What is interesting about oFIB is it not only prevents microloops in the case of link or node failures, but it also prevents microloops in the case of metric increases at one or more links in the network, and even the microloops that form in the case of metric decreases, or the insertion of new links or nodes in the network. The tradeoff is the two additional wait times that must be introduced to allow oFIB to operate: The computation of the new best paths must not take place until all the reports of the topology change are received by every node in the network, including the additional time each device must wait before installing new forwarding information.

Hop by Hop

In hop-by-hop distribution, each device computes a local best path and sends just the best path to its neighbors. Figure 14-8 illustrates.

In Figure 14-8, each device advertises information about what it can reach to each of its neighbors. D, for instance, advertises reachability to E, and B advertises reachability to C, D, and E toward A. It is interesting to consider what happens when A advertises its reachability toward E through the link along the top of the network. Once E receives this information, it will have two paths to B, for instance: one through D and one through A. In the same way, A will have two paths to B: one

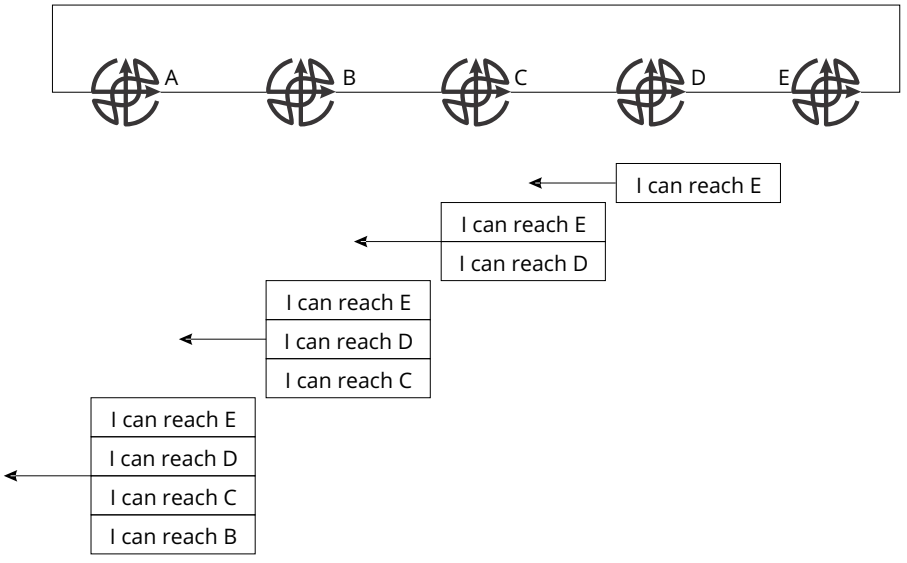


Figure 14-8 *Hop-by-Hop Distribution*

directly to B and another through E. Any of the shortest path algorithms discussed in previous chapters can determine which of these paths to use, but is it possible for microloops to form with a flooded distribution mechanism? Consider:

1. E chooses the path through A to reach B.
2. The [A,B] link fails.
3. A detects this failure, and switches to the path through E.
4. A then advertises this new path to E.
5. E receives the changed topology information and calculates a new best path through D.

During the time between steps 3 and 5, A will point to E as its best path to B, while E will point to A as its best path to B—a microloop. Most hop-by-hop distribution systems resolve this through *split horizon* or *poison reverse*. Defined, these are as follows:

- The split horizon rule states: a device should not advertise reachability toward a destination it is using to reach the destination.
- The poison reverse rule states: a device should advertise destinations toward the adjacent device it is using to reach the destination with an infinite metric.

If split horizon is implemented in Figure 14-8, E would not advertise reachability to B, as it is using the path through A to reach B. Alternatively, E could poison the route to B through A, which would have the effect of ensuring A has no path through E to B.

A Centralized Store

In a centralized system, each network device reports information about changes to the topology and reachability to a controller, or rather some collection of off box services and devices acting as a controller. While centralization often evokes the idea of a single device (or virtual device) to which all information is reported, and which feeds the correct forwarding information to all the packet processing devices in the network, this is an oversimplification of what a *centralized control plane* really means. Figure 14-9 illustrates.

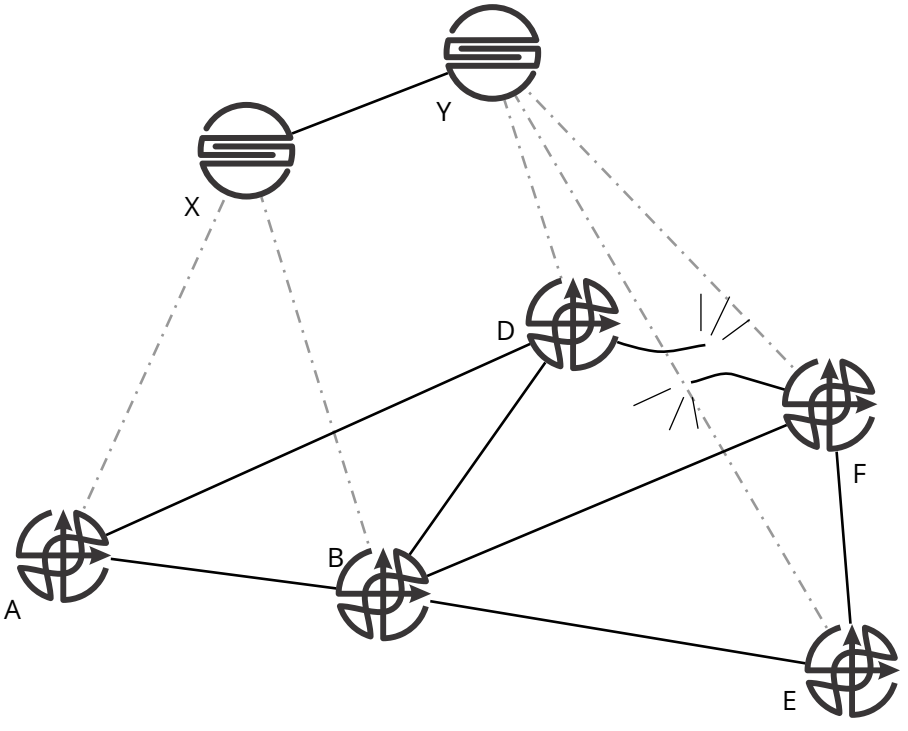


Figure 14-9 *Topology Changes and Centralized Control Planes*

In Figure 14-9, when the link between D and F fails:

1. D and F both report the topology change to the controller, Y.
2. Y forwards this information to the other controller, X.
3. Y computes the best path to each destination without the [D,F] link and sends it to each affected device in the network.
4. Each device installs this new forwarding information into its local table.

A specific instance of step 3 is Y computing a next best path to E without the [D,F] link, and sending it to D to install in its local forwarding table. Can microloops form in a centralized control plane?

- The databases in X and Y need to be synchronized for both controllers to compute the same loop-free paths through the network.
- Synchronizing these databases will involve the same challenges, and (probably) use the same solutions, as the solutions discussed thus far in this chapter.
- There will be some time required for the connected devices to discover the change in topology and report the change to the controller.
- There will be some time required for the controller to compute new loop-free paths.
- There will be some time required for the controller to notify the affected devices of the new loop-free paths through the network.

During the timing intervals described here, it is still possible for the network to form microloops. A centralized control plane most often translates to *the control plane is not running on the devices forwarding traffic*. Although they may seem radically different, centralized control planes actually use many of the same mechanisms to distribute topology and reachability, and the same algorithms to compute loop-free paths through the network, as distributed control planes.

Sharding and Control Planes

One interesting idea to reduce the state carried on any individual device, whether using a distributed or centralized control plane, is to *shard* the information in the topology table (or database). Sharding is splitting up the information in a single table based on some property of the data itself, and storing each resulting shard, or piece of the database, on a separate device. Figure 14-10 illustrates.

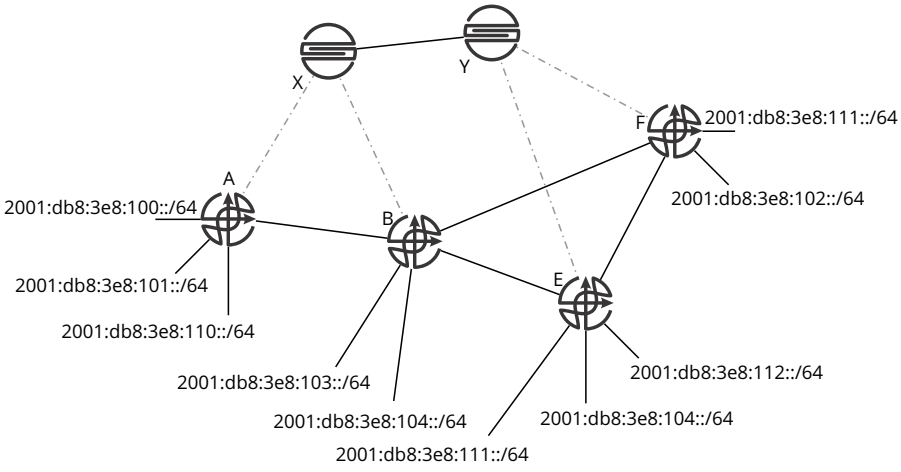


Figure 14-10 *Sharding Reachability Information*

In the network in Figure 14-10, assume both controllers, X and Y, have topology information for all the nodes (devices) and edges (links) in the network. However, to scale the size of the network, the reachable destinations have been sharded across the two controllers. There are many possible sharding schemes—anything able to divide the database (or table) into somewhat equally sized pieces will work. A hash is often used, as hashes can be quickly modified on every device where a shard is stored to rebalance the shard sizes.

In this case, assume the sharding scheme is something a bit simpler: a range of Internet Protocol (IP) addresses. Specifically, there are two ranges of IP addresses represented in the illustration: 2001:db8:3e8:100::/60, which contains 100::/64 through 10f::/64; and 2001:db8:3e8:110::/60, which contains 110::/64 through 11f::/64. Each of these address ranges is sharded onto a single controller; X will hold information about 2001:db8:3e8:100::/60, and Y will hold information about 2001:db8:3e8:110::/64. It doesn't matter where these reachable destinations are attached to the network. For instance, the information that 2001:db8:3e8:102::/64 is connected to F will be held at controller X, and the information that 2001:db8:3e8:110::/64 is connected to A will be held at controller Y. To build reachability information about 2001:db8:3e8:102::/64, Y will need to retrieve the information about where this destination is connected from X. This will be less efficient in terms of calculating shortest paths, but it will be more efficient in terms of storing the information needed to calculate the shortest paths. In fact, it is possible,

if the information is stored correctly (rather than in the trivial way used in this example), for several devices to calculate different parts of the shortest path and then exchange just the resulting tree with one another; this would distribute not only the storage, but also the processing.

There are a number of ways in which control plane information can be split up, stored, and calculations run across it to find a set of loop-free paths through a network. All of these systems face the same challenges discussed in the chapters in this section: discovering the topology, calculating a set of loop-free paths, distributing topology and reachability information, and reacting to changes in the topology.

Consistency, Accessibility, and Partitionability

In all three distribution systems discussed in this chapter—flooding, hop by hop, and centralized stores—the problem of microloops arises. Protocols implementing these techniques have various systems, such as split horizon and Loop-Free Alternates, to work around these microloops, or they allow the microloop to occur, assuming the results will not be too great on the network. Is there a unifying theory or model that will allow engineers to understand the problems inherent in the distribution of data through a network and the various tradeoffs involved?

There is: the *CAP theorem*.

In 2000, Eric Brewer, working on both theoretical and practical pursuits, postulated there are three qualities to a distributed database: Consistency, Accessibility, and Partition tolerance (CAP). Between these three, there is always a tradeoff such that you can choose two of the three in any system design. This conjecture, later proved true mathematically, is now known as the CAP theorem. The three terms are defined as

- **Consistency:** Every reader sees a consistent view of the contents of the database. If some device C writes to the database moments before two other devices, A and B, read from the database, the two readers will receive the same information. In other words, there is no lag between the writing of the database and both of the readers, A and B, being able to read the information that was just written.

- **Accessibility:** Every reader has access to the database when required (in near real time). The response to a read may be delayed, but every read will receive a response. Another way to put this is every reader has access to the database all the time; there is no time during which a reader would receive the answer "you cannot query this database right now."
- **Partition tolerance:** The ability of the database to be copied, or partitioned onto multiple devices.

It is simpler to *see* the CAP theorem in a small network; Figure 14-11 is used for this.

Assume A contains a single copy of a database that both C and D must access. Assume C writes some information to the database and then immediately after C and D both read the same information. The only processing that must take place to make certain C and D receive the same information is on A itself. Now, replicate the database, so there is a copy on E and another copy on F. Now assume K writes to the replica on E, and L reads from the replica on F. What will happen?

- F could return the value it currently has, even though it is not the same value K just wrote. This means the database returns an inconsistent reply, so *consistency* has been sacrificed by *partitioning* the database.

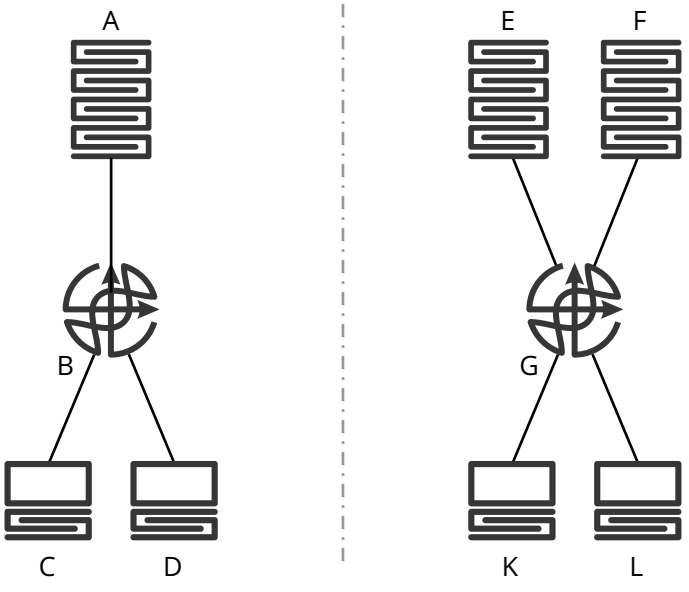


Figure 14-11 *The CAP Theorem Illustrated*

- If the two databases are synchronized, the reply will eventually be the same, of course, but it will take some time to package the change up (marshal the data), transfer it to F, and integrate the change into F's local copy. F could *lock* the database, or a specific part of the database, while the synchronization is taking place. In this case, when L reads the data, it may receive a reply that the record is locked. In this case, *accessibility* is lost, but consistency and the partitioning of the database are preserved.
- If the two databases are merged, then *consistency* and *accessibility* can be preserved, at the cost of *partitioning*.

There is no way to work it out so all three are preserved because of the time required to synchronize the information between the two copies of the database. The same problem holds true for a sharded database.

How does this apply to control planes? In a distributed control plane, the database from which the control plane draws information to calculate loop-free paths is partitioned across the entire network. Further, the database is locally readable at any time in order to calculate loop-free paths. Given the partitioning and accessibility required of the distributed database used in a control plane, you should expect consistency to suffer—and it does, resulting in microloops during convergence. A centralized control plane does not "solve" this problem; rather it just moves the problem around, or allows the designer to make different choices in the tradeoffs. A centralized control plane running on a single device will always be consistent, but it will not always be accessible, and the lack of partitioning will present an issue in the resilience of the network.

The three poles—consistency, accessibility, and partition tolerance—are not as clear-cut as they have been presented here, of course. There are often situations where less partitioning can result in more consistency, or short-term losses in availability will yield large increases in consistency. In other words, the CAP theorem does not really describe a set of three absolute poles, but rather a set of extreme points across a range of possibilities. In this way it is much like the *state, optimization, surface* triad found in an analysis of network complexity.[2]

The CAP theorem is a useful way to think about the performance of the database used in control planes.

Final Thoughts

The problem of detecting and distributing information about topology changes is second only to the problem of calculating shortest paths over a network in the space

2. For more information on complexity theory and control planes, see White and Tantsura, *Navigating Network Complexity*.

of network engineering. Breaking the problem down into four steps—detection, reporting, calculation, and installation—provides a framework you can use to assess the various options and think through the way a network really converges. Two broad classes of solutions are available, event driven and polling, each with a different set of tradeoffs; control planes normally use some form of record-level replication to carry topology information through the network in the case of a change.

The problems of loops and microloops have been particularly thorny in link state protocols, mirrored by dropped packets in distance vector protocols. These problems have occasioned years of research on the part of the best minds in protocol design; ultimately, however, all these solutions run up against the three-way tradeoff of the CAP theorem. The CAP theorem will show up again when considering centralized control planes.

The next two chapters will consider the three basic kinds of widely deployed control planes—distance vector, link state, and path vector. The material in this chapter and the two chapters on unicast loop-free paths should enable you to more readily understand the operation of the examples given in the following chapters. Overall, understanding what problems a control plane needs to solve, and the solutions available, will help you ask the right questions of any control plane and quickly understand its operation.

Further Reading

Bhatia, Manav, Carlos Pignataro, Sam Aldrin, and Trilok Ranganath. *OSPF Extensions to Advertise Seamless Bidirectional Forwarding Detection (S-BFD) Target Discriminators*. Request for Comments 7884. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7884.txt.

Bryant, Stewart, Stefano Previdi, Clarence Filsfils, Pierre Francois, Mike Shand, and Olivier Bonaventure. *Framework for Loop-Free Convergence Using the Ordered Forwarding Information Base (oFIB) Approach*. Request for Comments 6976. RFC Editor, 2013. https://rfc-editor.org/rfc/rfc6976.txt.

Gilbert, Seth, and Nancy A. Lynch. "Perspectives on the CAP Theorem." *Computer* 45 (2011): 30–36. doi:doi.ieeecomputersociety.org/10.1109/MC.2011.389.

Huang, Peng, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. "Gray Failure: The Achilles' Heel of Cloud-Scale Systems." In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, 150–155. HotOS '17. New York, NY, USA: ACM, 2017. doi:10.1145/3102980.3103005.

Katz, Dave, and David Ward. *Bidirectional Forwarding Detection (BFD)*. Request for Comments 5880. RFC Editor, 2010. https://rfc-editor.org/rfc/rfc5880.txt.

Pignataro, Carlos, David Ward, Manav Bhatia, Nobo Akiya, and Juniper Networks. *Seamless Bidirectional Forwarding Detection (S-BFD)*. Request for Comments 7880. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7880.txt.

White, Russ, and Jeff Tantsura. *Navigating Network Complexity: Next-Generation Routing with SDN, Service Virtualization, and Service Chaining*. Indianapolis, IN: Addison-Wesley Professional, 2015.

Review Questions

1. Consider the concept of information overload in reporting topology changes within the context of the State/Optimization/Surface (SOS) model. What are some of the tradeoffs in sending information more quickly versus slow topology change information more slowly, in terms of optimization versus state?
2. Consider polling and event-driven notification within the context of the state, optimization, surface model. List at least one or two, more if possible, positive and negative aspects of each kind of solution in each of the three realms of the model (SOS).
3. Read the paper "Gray Failure: The Achilles' Heel of Cloud-Scale Systems." Do you think a polling-based or event-driven solution would be best for solving the kinds of problems described in the paper? Why?
4. Explain why jitter is introduced in BFD sessions.
5. One alternative to record-level replication is binary-level replication of two files. For instance, rsynch uses binary replication to synchronize two files or databases. Why would network control planes not use binary replication?
6. What is the relationship between network topology and microloops? Will microloops form in any topology or only rings? Does the size of the ring impact whether or not microloops will form?

Chapter 15

Distance Vector Control Planes

Learning Objectives

After reading this chapter, you should understand:

- The classification of control planes into link state and distance vector
- The operation of the Spanning Tree Protocol
- The operation of the Routing Information Protocol
- How the Routing Information Protocol reacts to topology changes
- The operation of the Enhanced Interior Gateway Protocol
- How the Enhanced Interior Gateway Routing Protocol reacts to topology changes

The previous several chapters have considered three broad areas of problems every control plane for a packet switched network must solve and a range of solutions for each of those problems. The first problem considered was discovering the network topology and reachability. The second was calculating loop-free (and, in some cases, disjoint) paths through the network. The final problem, reacting to topology changes, is really a set of problems, including detecting and reporting changes to the network across the control plane.

This chapter will consolidate these problems and solutions by examining a few implementations of distributed control planes used for unicast forwarding in packet switched networks. The implementations here are not chosen because they are widely used, but rather because they represent a range of implementation choices among the solutions outlined in the previous chapters. The basic operation of each

protocol is considered in each case; later chapters in this part of the book will delve into information hiding and other more advanced topics in control planes, so they will not be covered here.

Rather than diving directly into protocol operation, the first section of this chapter will begin by defining various broad classes of control planes. Once these broad definitions are out of the way, six distributed unicast control planes will be considered.

Control Plane Classification

Control planes are typically classified by two characteristics. First, they are divided based on where the loop-free paths are calculated, whether on the forwarding device or off. Control planes in which the actual switching devices directly participate in the calculation of loop-free paths are then divided up based on the kind of information they carry about the network. There is no classification based on the algorithm used to calculate loop-free paths, although this is often intimately tied to the kind of information carried by the control plane.

While *centralized control planes* are often related to a few (or one, conceptually) controllers gathering the reachability and topology information from each switching device, calculating the set of loop-free paths, and downloading the resulting forwarding table to the switching devices, the concept is much less strict. A centralized control plane more generally just means calculating some part of the forwarding information someplace other than the actual forwarding device. This may mean a single device or a set of devices; it may mean a set of processes running in a virtual machine; it may mean calculating all of the required forwarding information or (perhaps) most of it.

Distributed control planes are generally marked by three general characteristics:

- A protocol running on each device, and that implements the various mechanisms required to transport reachability and topology information between devices
- A set of algorithms implemented on each device, used to compute a set of loop-free paths to known destinations
- The ability to detect and react to changes in reachability and topology locally at each device

In distributed control planes, not only is each packet switched hop by hop, but each hop determines the set of loop-free paths to reach any particular destination locally. Distributed control planes are generally divided into three broad classes of protocols: link state, distance vector, and path vector.

In link state protocols, each device advertises the state of each connected link, including reachable destinations and neighbors attached to the link. This information forms a topology database containing every link, every node, and every reachable destination in the network, across which an algorithm such as Dijkstra's or Suurballe's can be used to calculate a set of loop-free or disjoint paths. Link state protocols typically flood their databases so each forwarding device has a copy that is synchronized with every other forwarding device.

In distance vector protocols, each device advertises a set of *distances* to known reachable destinations. This reachability information is advertised by a particular neighbor that provides the *vector* information, or rather the direction through which the destination can be reached. Distance vector protocols typically implement either Bellman-Ford, Garcia-Luna's DUAL, or some similar algorithm to calculate loop-free paths through the network.

In path vector protocols, the path to reach the destination is recorded as the routing advertisement passes through the network, on a node-by-node basis. Other information may be added, such as metrics, to express some form of policy, but the primary loop-free nature of each path is calculated based on the actual paths advertisements take when passing through the network.

Figure 15-1 illustrates these three kinds of distributed control planes.

In Figure 15-1:

- In the link state example, at the top, each device advertises what it can reach to every other device in the network. Hence, A advertises reachability to B, C, and D; at the same time, D advertises reachability to 2001:db8:3e8:100::/64 and to C, B, and A.
- In the distance vector example, in the middle, D advertises reachability to 2001:db8:3e8:100::24 to C with its local cost, which is 1. C adds the [D,C] cost and advertises reachability to 2001:db8:3e8:100::64 with a cost of 2 to B.
- In the path vector example, at the bottom, D advertises reachability to 2001:db8:3e8:100::/24 through itself. C receives this advertisement and adds itself to [D,C].

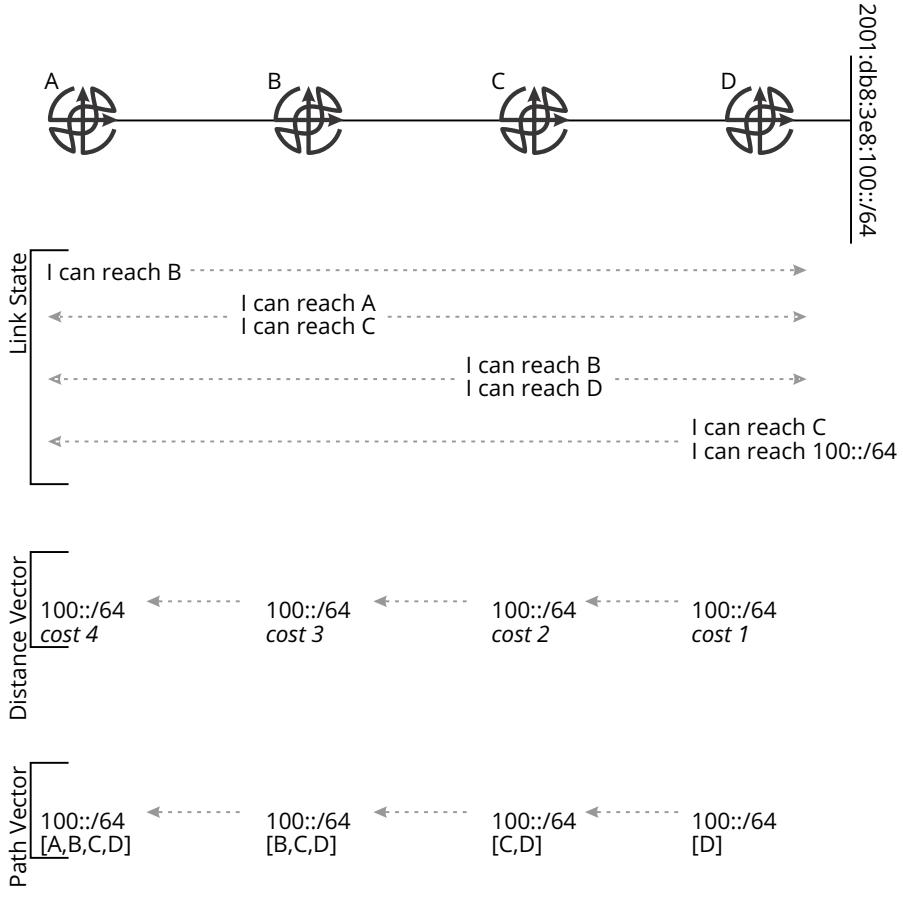


Figure 15-1 *Distributed Control Plane Classifications*

Control planes do not *always* neatly fit into one category or another, particularly when you move into various forms of information hiding. Some link state protocols, for instance, use distance vector principles with aggregated information, and path vector protocols often use some form of distance vector metric arrangement to augment the path in calculating loop-free paths. These classifications—centralized, distance vector, link state, and path vector—are important for understanding and encountering the network engineering world.

Packets Don't Follow Advertisements

A common mistake in assessing and understanding path vector protocols, a mistake that can be easily expanded to distance vector protocols, is to associate the *path of an advertisement* to the *path of a packet* through the network. Not only does the traffic flow in the opposite direction of the advertisement. This mistake shows up particularly in design when thinking through traffic engineering, in troubleshooting when trying to understand why a particular flow of packets is acting a particular way, and in securing the control plane to provide greater data plane security. The key point to remember is the control plane provides a loop-free path, but each forwarding device chooses which among a number of possible loop-free paths to use when forwarding traffic. Figure 15-2 illustrates one situation where the path of the packet does not follow the path of the update in a path vector system.

Assume, in the network shown in Figure 15-2, D and E are advertising /64's. At C, the /64 being advertised by D is being aggregated into a covering /60. This /60, plus the /64 being advertised by E, are both being advertised to B. At B, there is some policy in place that recognizes the overlapping prefixes, passing the /60 through to A, and blocking the /64. Forwarding is not impacted by this configuration; traffic transmitted through A toward either 2001:db8:3e8:100::/64 or 2001:db8:3e8:101::/64 will be delivered correctly.

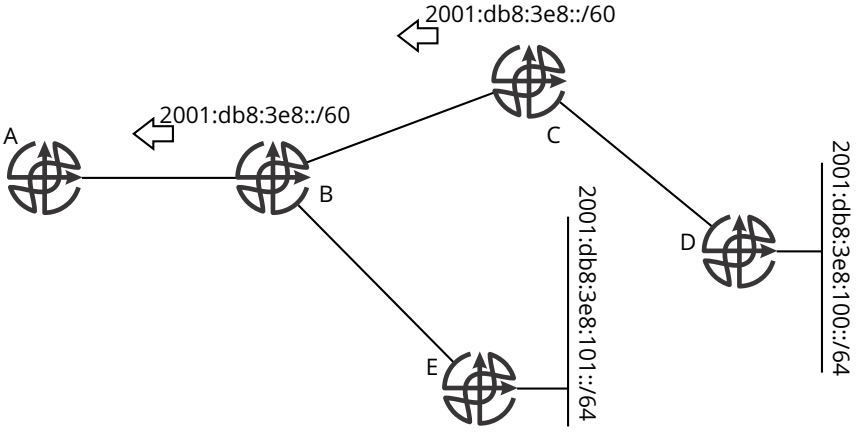


Figure 15-2 *Update versus Packet Path through a Network*

Assume the path vector system in place treats an aggregate in the same way it treats any other reachable destination. At C, then, when the 2001:db8:3e8::/60 aggregate route is created, C advertises the route toward B with a path originating at C itself. Hence, when A receives this aggregated route, the path through the network is [A,B,C]. From A's perspective, then, traffic being transmitted toward 2001:db8:3e8:100::/64 and 2001:db8:3e8:101::/64 will both leave the network at C. This is not true.

Looking at the routing (or topology) table at A, then, will give you apparently wrong information about how the traffic is forwarded through the network to reach either of these destinations. While routing is still correct, using the routing table to see how traffic will be forwarded through the network—either for troubleshooting, design, or security—is of limited value. This kind of situation can arise with any routing protocol; forwarding is performed hop by hop through the network, with each device making an independent decision about where to send each individual packet.

Spanning Tree Protocol

The Spanning Tree Protocol (STP) was originally designed by Radia Perlman, and first described in 1985 in *An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN*.[1] STP is unique in the list of control planes considered here because it was originally designed to support switching rather than routing. In other words, STP was designed to support forwarding on packets without a Time to Live (TTL), and without a per hop header swap by the switching device. Packets switched based on the STP are carried through the network without change.

Building a Loop-Free Tree

The process of building a loop-free tree is as follows:

1. Each device places all ports in blocked mode so that no port will forward any traffic, and begins advertising Bridge Protocol Data Units (BPDUs) out each port. This BPDU contains
   a. The ID of the advertising device, which is a priority combined with a local interface Media Access Control (MAC) address.

1. Perlman, "An Algorithm for Distributed Computation of a Spanningtree in an Extended LAN."

   b. The ID of the candidate root bridge. This is the bridge with the lowest ID the local device knows about. If every device on the network starts at the same moment, then each device would advertise itself as the candidate root bridge until it learned of other bridges with a lower bridge ID.

2. On receiving a BPDU on an interface, the root bridge ID contained in the BPDU is compared with the locally stored lowest root bridge ID. If the root bridge ID contained in the BPDU is lower, then the locally stored root bridge ID is replaced with the newly discovered bridge with a lower ID.
3. After a few rounds of advertisements, every bridge should have discovered the bridge with the lowest bridge ID in the network and declared this bridge to be the root bridge.
   a. This should occur while all the ports on all the devices are still in a blocked state (not forwarding traffic).
   b. To make certain this does happen while all the ports are still blocked, a timer is set long enough to allow the root bridge to be elected.
4. Once the root bridge is elected, the shortest path to the root bridge is determined.
   a. Each BPDU also contains a metric to reach the root bridge. This metric may be a hop count, but the cost of each hop can vary based on administrative variables as well, such as the bandwidth of the link.
   b. Each device determines the port through which it has the lowest cost path to the root bridge; this is marked as the root port.
   c. If there is more than one path to the root bridge with the same cost, a tie breaker is used; this is normally the port identifier.
5. For any link on which two bridges are connected
   a. The bridge with the lowest cost path to the root bridge is elected to forward traffic off the link toward the root bridge.
   b. The port connecting the elected forwarder to the link is marked as the designated port.
6. Ports marked as either root or designated ports are allowed to forward traffic.

The result of this process is a single tree over which every destination in the network is reachable. Figure 15-3 is used to show how STP works in an actual topology.

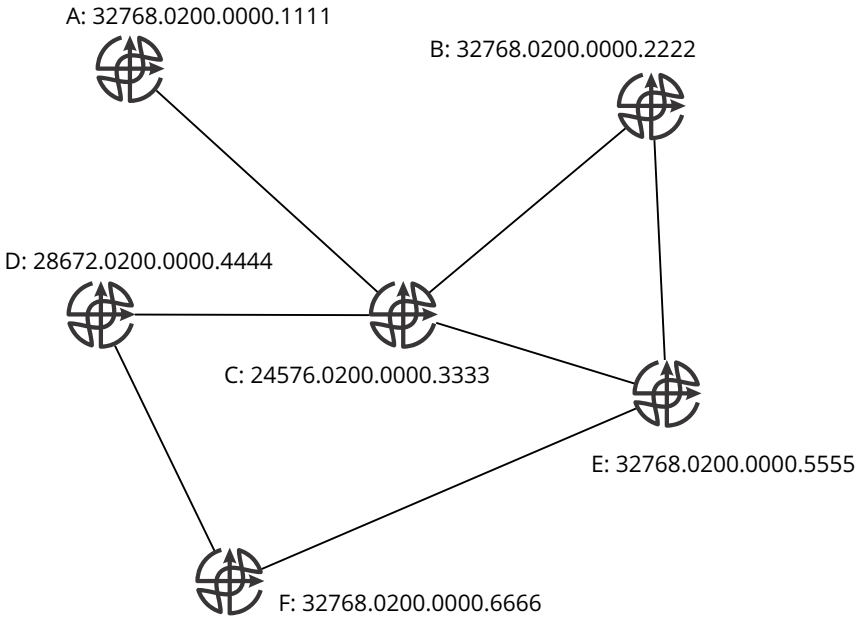


Figure 15-3 *Spanning Tree Protocol Operation*

Assume all the devices in Figure 15-3 were turned on at the same moment. There are a number of variations possible in timing, but the process of building a set of loop-free paths through the network would look, from F's perspective, something like this:

1. Elect the root bridge:
   a. F advertises a BPDU to E and D with an ID and a candidate root bridge of 32768.0200.0000.6666.
   b. D (assuming D has not received any BPDUs) advertises a BDPU with an ID and a candidate root bridge of 28672.0200.0000.4444.
   c. E (assuming E has not received any BPDUs) advertises a BPDU with an ID and a candidate root bridge of 32768.0200.0000.5555.
   d. At this point, F will elect D as the root bridge, and start advertising BPDUs with its local ID and the candidate root bridge set to D's ID.
   e. At some point, D and E will both receive BPDUs from C, which has a lower bridge ID (24576.0200.0000.3333). On receiving this BPDU, they will both set their candidate root bridge ID to C's ID and send new BPDUs to F.

f. On receiving these new BPDUs, F will note the new candidate root bridge ID is lower than its previous candidate root bridge ID, and it will then elect C as the root bridge.

g. After several rounds of BDPUs, all the bridges in the network will elect C as the root bridge.

2. Mark the root ports by finding the shortest path to the root:

   a. Assume each link is a cost of 1.

   b. D will receive a BDPU from C with a local ID and root bridge ID of 24576.0200.0000.3333 and a cost of 0.

   c. D will add the cost of reaching C, a single hop, advertising it can reach the root bridge with a cost of 1 to F.

   d. E will receive a BDPU from C with a local ID and root bridge ID of 24576.0200.0000.3333 and a cost of 0.

   e. E will add the cost of reaching C, a single hop, advertising it can reach the root bridge with a cost of 1 to F.

   f. F now has two advertisements toward the root bridge with equal cost; it must break the tie between these two available paths. To do so, F examines the bridge ID of the advertising bridges. D's bridge ID is lower than E's, so F will mark its port toward D as its root port.

3. Marking the designated ports on each link:

   a. F's only other port is toward E. Should this port be blocked?

   b. To determine this, F compares its local bridge ID with E's bridge ID. The priorities are the same, so the local port addresses must be compared to make the decision. F's local ID ends in 6666, while E's ends in 5555, so E's is lower.

   c. F does *not* mark the interface toward E as a designated port; instead, it marks this port as *blocked*.

   d. E does the same comparison and marks its port toward F as a designated port.

   e. D compares its cost toward the root with F's cost toward the root.

   f. D's cost is lower, so it will mark its port toward D as a designated port.

Figure 15-4 illustrates the blocked, designated, and root ports once these calculations are completed.

The ports in Figure 15-4 are marked with bp for blocked port, rp for root port, and dp for designated port. The result of the process is a tree that can reach any segment in the network, and hence the hosts connected to any segment in the network. One interesting point about STP is the result is a *single* tree across the entire topology, anchored at the root bridge. If some host connected to E sends a packet to a host connected to B or F, the packet must travel through C, the root bridge, because one of the two ports on the [F,E] and [E,B] links is blocked. This is not the most efficient use of bandwidth, but it does prevent looping packets during normal forwarding.

How is neighbor discovery handled in STP? Neighbor discovery is not addressed from the perspective of the reliable transport of information through the network at all. Each device in the network builds its own BPDUs; these BPDUs are not carried *through* any device, so there is no need for end-to-end reliable transport in the control plane. Neighbor discovery is used, however, to elect a root bridge and to build a loop-free tree across the entire topology using BPDUs. What about dropped and missed packets? Any device running STP retransmits its BPDUs on every link periodically (according to a *retransmission timer);* it takes a few dropped packets (according to a *dead timer)* for a device running STP to assume its neighbors have

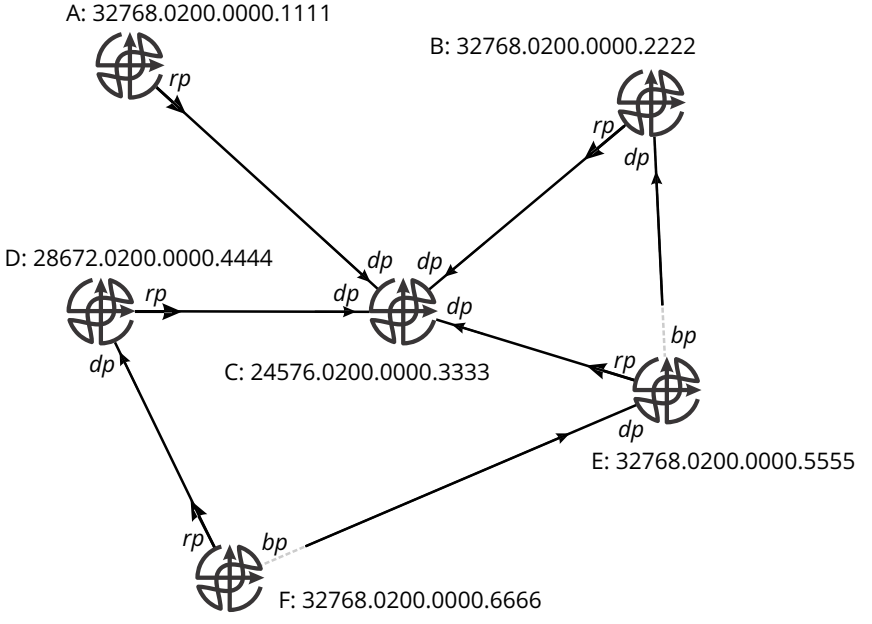


Figure 15-4 *The Result of the Spanning Tree Process*

failed, and hence to restart calculating the root bridge and port statuses. There is no two-way connectivity check in STP, either on a per neighbor basis or across the entire path. Nor is there any Maximum Transmission Unit (MTU) check of any kind. STP learns about the topology by combining BPDUs with local link information on a per node basis; there is no single node in the network with a table describing the entire topology, however.

Learning about Reachable Destinations

How does STP enable forwarding? More specifically, how do devices running STP learn about reachable destinations? Figure 15-5 is used to explain.

Figure 15-5 shows the state of the network with the spanning tree calculated and each port marked as a designated or root port. There are no blocked ports in this topology because there are no loops. Assume B, C, and D have no information about attached devices; A sends a packet toward E. What happens at this point?

1. A transmits the packet onto the [A,B] link. As B has a designated port on this link, it will accept the packet (switches accept all packets on designated ports) and examine the source and destination addresses.
2. B can determine that A is reachable through this designated port because it has received a packet from A on this port. Based on this, B will insert A's MAC address as reachable in its forwarding table through its interface onto the [A,B] link.
3. B does not have any information about E; therefore it will flood this packet out every one of its nonblocked ports. In this case, the only other port B has is its root port, so B will forward this packet toward C. This flooding is called Broadcast, Unknown, and Multicast (BUM) traffic; BUM traffic is something every control plane that learns destinations during the forwarding process must manage in some way.
4. When C receives this packet, it will examine the source address and discover that A is reachable through the designated port attached to [B,C]. It will insert this information into its local forwarding table.

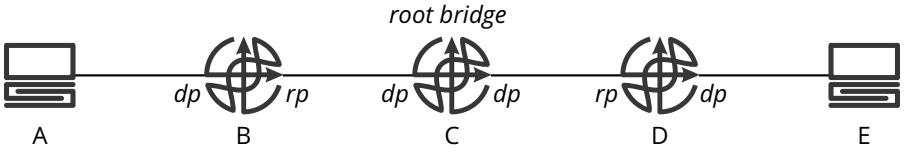


Figure 15-5 *STP Reachability Discovery*

5. C also has no information about where E is located on the network, so it will simply flood the packet on all nonblocked ports. In this case, the only other port C has is onto the [C,D] link.
6. D repeats the same process B and C have followed, learning that A is reachable through its root port onto the [C,D] link and flooding the packet onto the [D,E] link.
7. When E receives the packet, it processes the information and sends a reply back toward A.
8. When D receives this reply packet from E, it will examine the source address and discover E is reachable on its designated port onto the [D,E] link. D does know the path back to A, as it discovered this information in processing the first packet in the flow traveling from A to E. It will look up A in its forwarding table and transmit the packet onto the [C,D] link.
9. C and B will repeat the process D and C have used to discover the location of E and to forward the return traffic back to A.

In this way—learning the source address from incoming packets, and either flooding or forwarding packets onto outgoing links—every device in the network can learn about every reachable destination. Because STP relies on learning reachable destinations in reaction to packets being transmitted on the network, it is classified as a reactive control plane. Note this learning process is at the host level; subnets and Internet Protocol (IP) addresses are not learned, but rather the physical address of the host interface. If a single host has two physical interfaces onto the same wire, it will appear as two different hosts to the STP control plane.

How is information removed from the forwarding tables on each device? Through a timeout process. If a forwarding entry has not been used in a specific time (a hold timer), the entry is removed from the table. Hence, STP relies on cached forwarding information.

Concluding Thoughts on the Spanning Tree Protocol

STP is clearly not a link state protocol, nor is it a path vector protocol. Is it a distance vector protocol? Any confusion over how to classify the protocol stems from the initial selection of a root bridge before the shortest paths are calculated. Removing this first step, it is easier to classify STP as a distance vector protocol using a distributed form of the Bellman-Ford algorithm to calculate loop-free paths across the topology. What should be done with the initial root bridge calculation? This part of the process ensures there is just one Shortest Path Tree across the entire network. So STP can

be classified as a *distance vector protocol* that uses the *Bellman-Ford algorithm* to compute a single set of shortest paths for *all* destinations across the entire network. Another way to put this is STP computes a Shortest Path Tree across the *topology*, rather than across the *destinations*.

Why is it important that a *single* tree be calculated across the entire network? This is related to the way in which STP learns reachability information: STP is a reactive control plane, learning reachability in response to actual packets flowing through the network. If each device built a separate tree rooted at itself, this reactive process would lead to an inconsistent view of the network topology and hence to forwarding loops.

STP and Broadcast Storms

Broadcasts are an important part of service discovery in most applications. For instance, in Figure 15-6, how can A discover the presence of a particular service on F?

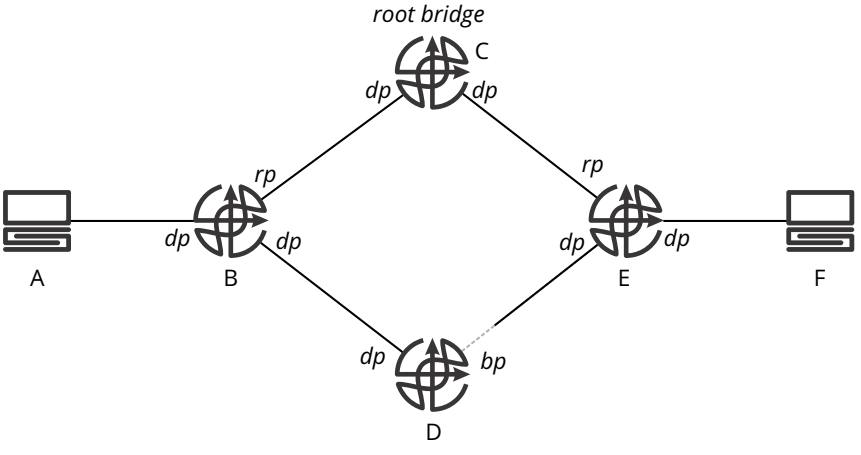


Figure 15-6 *Broadcast Storms and the Spanning Tree Protocol*

The easiest thing for A to do in this situation is to send some sort of packet that will be delivered to each host connected to the network and await a reply from the host running the service in question. So A sends a broadcast asking about a particular service or device. How should B, C, D, and E treat this broadcast? As the broadcast is not a "learnable" address (every device on

every segment should receive the broadcast), the best thing for the switches to do is to forward the packet on every nonblocked port.

What happens if A sends a *lot* of broadcasts? What happens if a host sends enough broadcasts to cause BPDUs to be dropped? In this case, STP itself will become confused and will likely create a forwarding loop in the topology. Such a forwarding loop will, of course, forward broadcast packets *forever;* there is no TTL to drop packets after they have traversed the network a specific number of times. Each broadcast transmitted by A, in this situation, will remain in the network forever, looping, perhaps, among the switches B, C, D, and E. And each broadcast added to the load of the network will, of course, prevent BPDUs from being successfully transmitted or received, preventing STP from converging.

Hence, the traffic on the network prevents STP from converging, and the lack of convergence increases the traffic load on the network itself—a positive feedback loop causing havoc throughout the network. These events are called broadcast storms, and are common enough in STP-based networks to cause wise network designers and operators to limit the scope of any STP domain. The existence of broadcast storms has also driven a number of modifications to the operation of STP, such as simply replacing the base protocol with a true link state control plane.

The Routing Information Protocol

The Routing Information Protocol (RIP) was originally specified in RFC1058, *Routing Information Protocol,* published in 1998.[2] The protocol was updated in a series of more recent RFCs, including RFC2435, *RIP version 2,*[3] and RFC2080, *RIP Next Generation for IPv6*.[4] Figure 15-7 is used to explain RIP operation.

The operation of RIP is deceptively simple. In Figure 15-7:

1. A discovers 2001:db8:3e8:100::/64 because it is configured on a directly attached interface.
2. A adds this destination to its local routing table with a cost of 1.

2. Hendrick, *Routing Information Protocol*.
3. Malkin, *RIP Version 2*.
4. Malkin and Minnear, *RIPng for IPv6*.

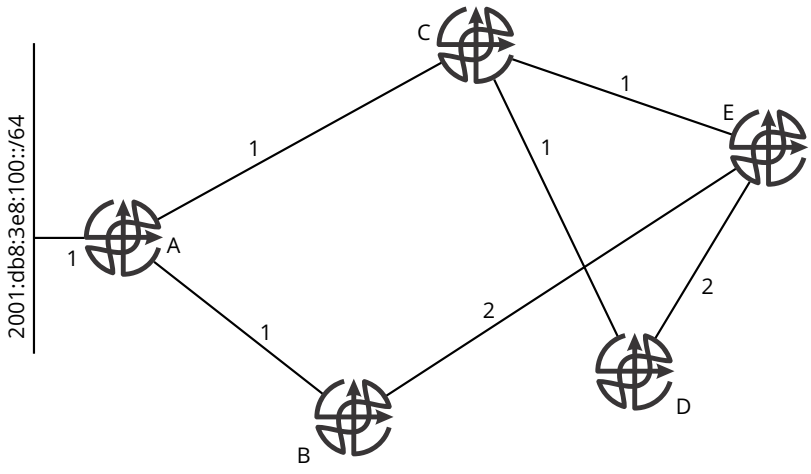


Figure 15-7 *RIP Operation Example*

3. As 100::/64 is installed in the local routing table, A will advertise this reachable destination *(route)* to B and C.
4. When B receives this route, it will add the cost of the inbound interface so that the path through A has a cost of 2, and examine its local table for any lower-cost routes to this destination. As B has no other path to 100::/64, it will install the route in its routing table and advertise the route to E.
5. When C receives this route, it will add the cost of the inbound interface so that the path through A has a cost of 2, and examine its local table for any lower-cost routes to this destination. As C has no other path to 100::/64, it will install the route in its routing table and advertise the route to D and E.
6. When D receives this route, it will add the cost of the inbound interface from C so that the path through C has a cost of 3, and examine its local table for any lower-cost routes to this destination. As D has no other path to 100::/64, it will install the route into its routing table and advertise the route to E.
7. E will now receive three copies of the same route; one through C with a cost of 3, one through B with a cost of 4, and one through D with a cost of 5. E will choose the path through C with a cost of 2, installing this path to 100::/64 into its local routing table.
8. E will not advertise any path to 100::/64 toward C, because it is using C as its best path to reach this specific destination. Thus, E will *split horizon* its advertisement of 100::/64 toward C.

9. While E will advertise its best path, through C, to both D and B, neither will choose the path through E, as they already have better paths available toward 100::/64.

RIP advertises a set of destinations and costs one hop at a time through the network; hence it is considered a *distance vector* protocol. The process that RIP uses to find a set of loop-free paths through the network is considered a distributed form of the Bellman-Ford algorithm, but it is not obvious how the process that RIP is using is related to Bellman-Ford.

Tying Bellman-Ford to RIP

To see the connection, it is best to think of each hop in the network as a single row in the topology table; this is illustrated in Figure 15-8.

Chapter 12, "Unicast Loop-Free Paths (1)," describes Bellman-Ford operating across a topology table, arranged as a set of columns and rows. Using the row numbers indicated in Figure 15-8, you can build a similar topology table for this network, as shown in Table 15-1.

Table 15-1 *A Topology Table Built from the Network in Figure 15-8*

Row	Source (s)	Destination (d)	Distance (cost)
1	100::/64	A	1
2	A	B	1
3	B	C	2
4	C	D	2

Assume each row of the table is run through the Bellman-Ford algorithm by a different node. For instance, A computes Bellman-Ford across the first row and passes the result on to B. Likewise, B computes Bellman-Ford across the relevant rows and passes the result on to C. Bellman-Ford would still be the algorithm used to compute the set of loop-free paths through the network; it would simply be distributed across the nodes in the network. This, in fact, is how RIP operates. Consider the following:

1. A computes the first row in the table, setting the predecessor for 100::/64 to A and the cost to 1. A passes this result on to B for the second round of processing.

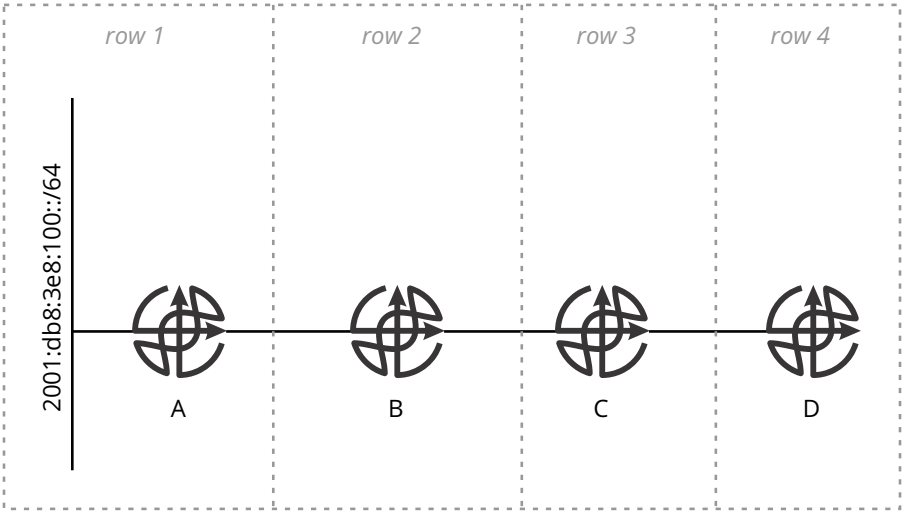


Figure 15-8 *RIP and Bellman-Ford*

2. B processes the second row in the table, setting the predecessor for 100::/64 to B and the cost to 2. B passes this result on to C for the third round of processing.
3. C processes the second row in the table, setting the predecessor for 100::/64 to C and the cost to 2. C passes this result on to D.

The Bellman-Ford distributed processing is more difficult to see in more complex topologies, because there is more than one "result table" being passed around the network. These "result tables" will eventually merge at the source node, however. Figure 15-9 illustrates.

In Figure 15-9, A would compute a provisional result table as the first "round" of the Bellman-Ford algorithm, passing the result on to both B and E. B would compute a provisional result based on local information, passing this on to C, and then C to D. In the same way, E would compute a provisional result table based on local information, passing this on to F, and then F to D. At D, the two provisional results are combined into a final table from D's perspective. Of course, the provisional table is considered final for the device at each hop. From E's perspective, the table it computes based on locally available information plus the advertisement from A is the *final* table of loop-free paths to reach 100::/64.

The entire distributed process has the same effect as walking across every row in the topology table the same number of times as entries in the topology table itself, slowly sorting the predecessor and cost fields for each entry based on newly set predecessors in the previous round of computation.

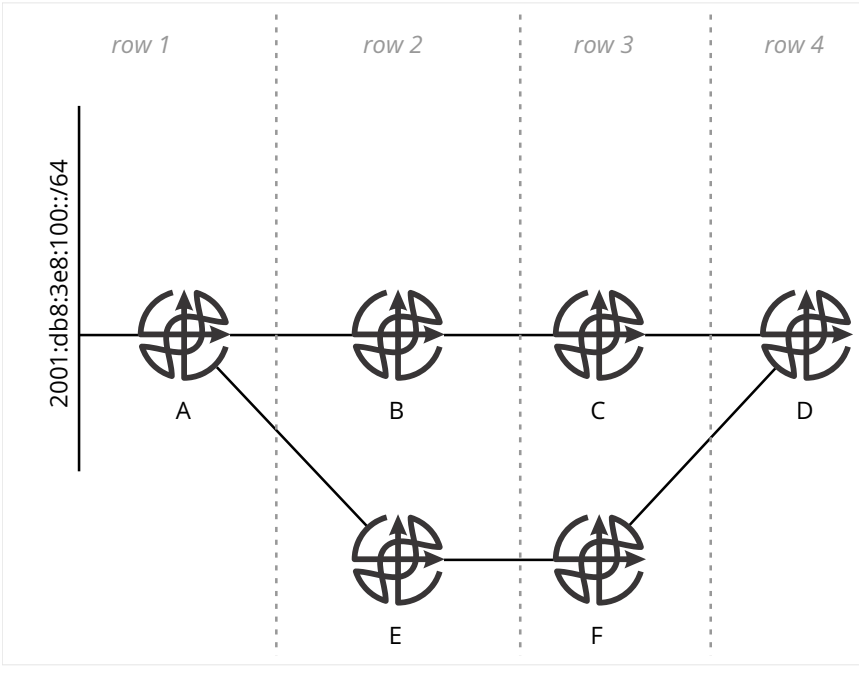


Figure 15-9 *Distributed Bellman-Ford in Parallel*

Reacting to Topology Changes

How does RIP remove reachability information from the network in the case of a node or link failure? Figure 15-10 is used to explain.

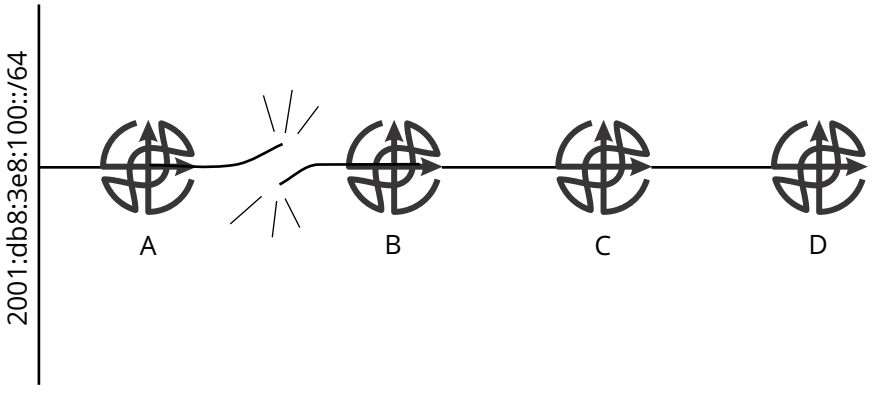


Figure 15-10 *Link Failure in a RIP Network*

There are two different possible reactions to the loss of the [A,B] link, depending on the version and configuration of RIP running in this network. The first possible reaction is to simply let the information about 100::/64 time out. Assuming the *invalid timer* (a form of hold timer) for any given route is 180 seconds (a common setting in RIP implementations):

- B would notice the failed link immediately, as it is directly connected, and remove 100::/64 from its local routing table.
- B would stop advertising reachability to 100::/64 toward C.
- C will remove reachability to this destination from its local routing table and stop advertising reachability toward 100::/64 to D 180 seconds after B stops advertising reachability to 100::/64.
- D will remove reachability to this destination from its local routing table 180 seconds after C stops advertising reachability to 100::/64.

At this point, the network has converged on the new topology information. This is obviously a rather slow process, as each hop must wait for every router closer to the destination to time the destination out before discovering the loss of connectivity.

To speed up this process, most RIP implementations also include *triggered updates*. If triggered updates are implemented and deployed in this network, when the [A,B] link fails (or is removed from service), B will remove reachability to 100::/64 from its local table and send a triggered update to C, informing C of the failed reachability toward the destination. This triggered update generally takes the form of an advertisement with an infinite metric, or rather what is known as a *poison reverse*. Triggered updates are often *paced,* so a flapping link will not cause the triggered updates themselves to overwhelm either a link or a neighboring router.

Two other timers are specified in RIP for use during convergence: the *flush timer* and the *hold-down timer*. When a route times out (as described above), it is not immediately removed from the local routing table. Rather, another timer is set that determines when the route will be flushed from the local table. This is the flush timer. Further, there is a separate time period during which any route with a worse metric than the previously known metric will not be accepted. This is the hold-down timer.

Concluding Thoughts on RIP

RIP carries information about locally reachable destinations to neighbors, along with a cost for each destination; hence it is a distance vector protocol. Reachable destinations are learned through local information at each device, and carried

through the network by the protocol regardless of traffic flow; hence RIP is a *proactive* control plane.

RIP does not form adjacencies for the reliable transmission of data through the network; rather, RIP relies on periodically transmitted updates to ensure information has not become out of date or has been accidentally dropped. The amount of time any piece of information is kept is based on a *hold timer,* and the frequency of transmissions is based on an *update timer;* the hold timer is normally set to three times the value of the update timer.

As RIP has no true adjacency process, it does not detect whether or not two-way connectivity exists; hence there is no Two-Way Connectivity Check (TWCC). No method to check the MTU between two neighbors is built into RIP, either.

The Enhanced Interior Gateway Routing Protocol

The Enhanced Interior Gateway Routing Protocol (EIGRP) was originally released in 1993 to replace Cisco's Interior Gateway Routing Protocol (IGRP). The primary reason for replacing IGRP was its inability to carry classful routing information; specifically, IGRP could not carry subnet masks. Rather than rebuild the protocol to support prefix lengths, engineers at Cisco (specifically Dino Farinacci and Bob Albrightson) decided to build a new protocol based on Garcia-Luna's Diffusing Update Algorithm (DUAL). Dave Katz rebuilt the transport to resolve some widely encountered problems in the mid-1990s. Based on this initial implementation, a team led by Donnie Savage modified the operation of the protocol heavily in the 2000s, adding a number of scaling features, and rewriting key parts of EIGRP's reaction to topology changes. EIGRP was released, along with virtually all of these enhancements, in the informational RFC7868 in 2013.

While EIGRP is not often considered for active deployment in most service provider networks (most operators prefer a link state protocol instead), DUAL introduces some important concepts into the conversation around loop-free paths. DUAL is also used in other protocols, such as BABEL (specified in RFC6126, and used in lightweight radio and home network environments).

EIGRP Metrics

EIGRP was originally designed to read the bandwidth, delay, error rate, and other factors off links in near real time and carry them as metrics. This would allow EIGRP to react to changing network conditions in real time, and hence allow networks running EIGRP to more efficiently use the

available network resources. At the time EIGRP was originally designed and deployed, however, there were no "guard rails" put in place to prevent feedback loops between, for instance, the reaction of the protocol to changes in available bandwidth and the shifts in traffic based on the available bandwidth. If a pair of links with near real time available bandwidth were placed parallel to one another, traffic will shift to the one with the most available bandwidth, causing the protocol to react by showing more available bandwidth on the other link, improving its metric, and hence traffic to shift to the other link. This process of traffic shifting back and forth between links can be solved in various ways, but it caused enough problems in early EIGRP deployments for this near-real-time capability to be removed from the code. Instead, EIGRP reads the characteristics of an interface at specific times and advertises these metrics for the interface regardless of the network conditions.

EIGRP carries five different route attributes, including bandwidth, delay, load, reliability, and the MTU. Four of the metrics are combined using the formula shown in Figure 15-11.

$$\left[\left[K1 * \text{min throughput} + \left[\frac{K2 * \text{min throughput}}{256 - \text{load}}\right]\right] + \left[K3 * \Sigma\,(\text{delays})\right] + \left[K6 * \text{ext attribute}\right]\right] * \left[\frac{K5}{K4 - \text{reliability}}\right]$$

Figure 15-11 *The EIGRP Metric Calculation Formula*

The default *K* values in this formula cause the entire formula to collapse to $(10^7/throughput) * delay$. Replacing *throughput* with *minimum bandwidth* along the path yields the version most engineers are familiar with, $(10^7/bandwidth) * delay$.

The bandwidth and delay values are, however, scaled in more recent versions of EIGRP to account for links with a bandwidth higher than 10^7 kbps.

Note

Throughout this discussion of EIGRP, the bandwidth of every link is assumed to be set to 1,000, and the K values set to their default values, leaving the delay as the only component impacting the metric. Given this, the delay value alone is used as the metric in these examples to simplify the math.

Figure 15-12 is used to describe the operation of EIGRP.
The operation of EIGRP in this network is very simple on the surface:

1. A discovers 2001:db8:3e8:100::/64 because it is directly attached (this could be through the interface configuration, for instance).
2. A adds the inbound interface cost, here shown as a delay of 100, to the route, and installs it in its local routing table.
3. A advertises 100::/64 to B and C through the two other connected interfaces.
4. B receives this route, adds the inbound interface cost (for a total delay of 200), and examines its local table for any other (or better) routes to this destination; B does not have a route to 100::/64, so it installs the route in its local routing table.
5. B advertises 100::/64 to D.
6. C receives this route, adds the inbound interface cost (for a total delay of 200), and examines its local table for any other (or better) routes to this destination; C does not have a route to 100::/64, so it installs the route in its local routing table.
7. C advertises 100::/64 to D.
8. D receives the route to 100::/64 from B, adds the inbound interface cost (for a total delay of 300), and examines its local table for any other (or better) routes to this destination; D does not have a route to this destination, so it installs the route in its local routing table.

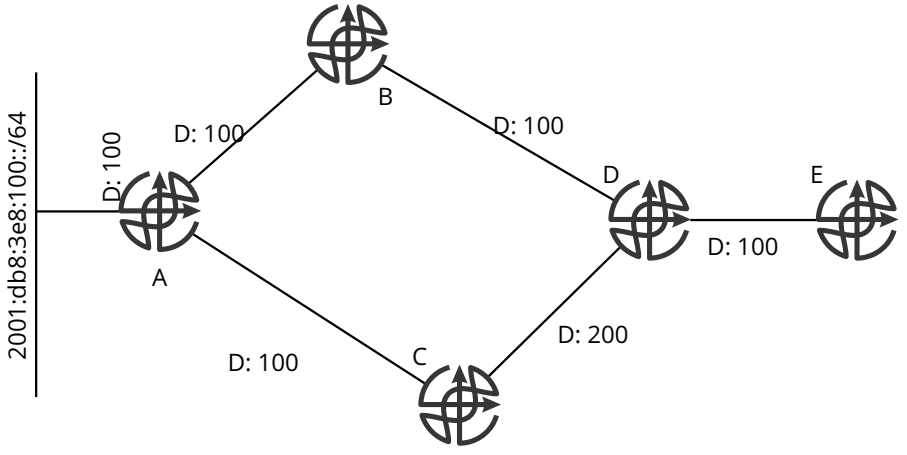


Figure 15-12 *Sample Network for EIGRP Operation*

9. D receives the route to 100::/64 from C, adds the inbound interface cost (for a total delay of 400), and examines its table for any other (or better) routes to this destination; D does have a better route to 100::/64, through B, so it inserts the new route into its local topology table (see below for the additional processing D does on this alternate path).
10. D advertises the route to 100::/64 to E.
11. E receives the route to 100::/64 from D, adds the inbound interface cost (for a total delay of 400), and examines its local table for any other (or better) routes to this destination; E does not have a route to this destination, so it installs the route in its local routing table.

Thus far, this is very similar to the operation of RIP. Step 9, however, needs a good bit more detail. After step 8, D has a path to 100::/64 with a total cost of 300; this is the feasible distance to the destination, and B is the successor, as it is the path with the lowest-cost path. At step 9, D receives a second path to this same destination. In RIP, or other Bellman-Ford implementations, this second path would either be ignored or discarded. EIGRP, being grounded in DUAL, however, will examine this second path to determine if it is loop free or not. Can this path be used if the primary path fails?

To determine whether this alternate path is loop free or not, D must compare the feasible distance with the distance C has reported as its cost to reach 100::/64—the reported distance. This information is available in the advertisement D receives from C (remember that C advertises the route with its cost to the destination; D adds the cost of the [B,D] link to this to find the total cost through C to reach 100::/64). The reported distance through C is 200, which is less than the local feasible distance, which is 300. Hence, the route through C is loop free and is marked as a feasible successor.

Reacting to a Topology Change

How are these feasible successors used? Assume the [B,D] link fails, as illustrated in Figure 15-13.

When this link fails, D will examine its local topology table to discover if it has another loop-free path to the destination. Since the path through C is marked as a feasible successor, D does have an alternate path. In this case, D can simply switch to using the path through C to reach 100::/64. D will *not* recalculate the feasible distance in this case, as it has not received any new information about the network topology.

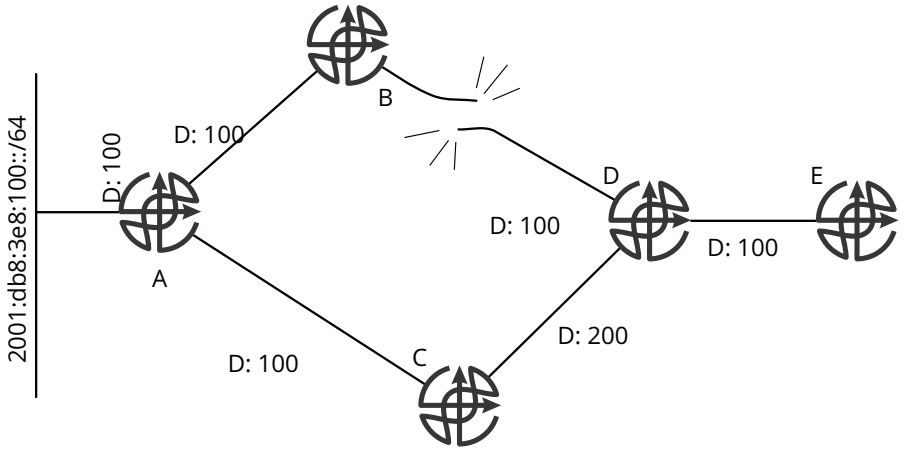


Figure 15-13 *Using the Feasible Successor*

What if the link between C and A fails, instead, as illustrated in Figure 15-14?

In this case, before the failure, C has two paths to 100::/64: one through A with a total delay of 200 and a second through D with a total delay of 500. The feasible distance at C will be set to 200, as this is the cost of the best path available when convergence is complete. The reported distance at D, 300, is greater than the feasible distance at C, so C will not mark the path through D as a feasible successor. Once the [A,C] link fails, since C does not have an alternate path, it will mark the route active

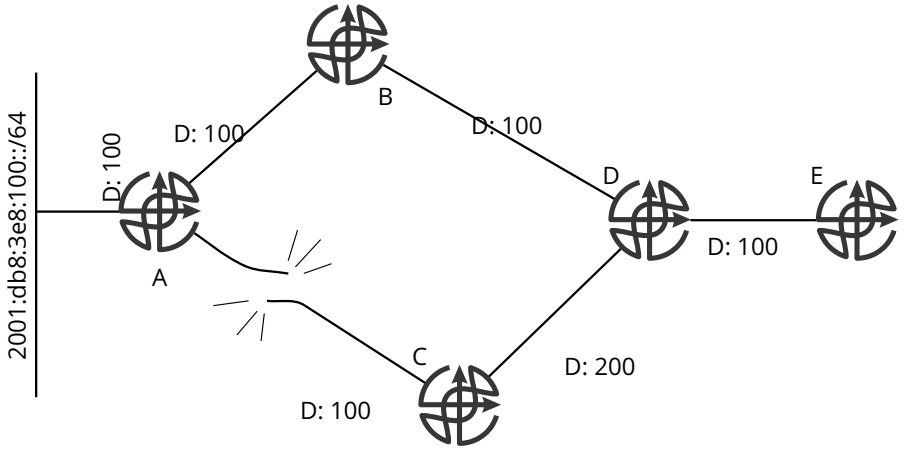


Figure 15-14 *Reacting to Failure Without a Feasible Successor in EIGRP*

and send a query to each of its neighbors requesting updated information about any available path to 100::/64.

When D receives this query, it will examine its local topology table and find that its best path toward 100::/64 is still available. Because this path still exists, the EIGRP process on D can assume that the current best path, through B, has not been impacted by the failure of the [A,C] link. D replies to this query with its current metric, which indicates this path is still available, and is loop free from D's perspective.

On receiving this reply, C will note it is not waiting on any other neighbors to respond (as it has just one neighbor, D). As C has received all the replies it is waiting on, it will recalculate the available loop-free paths, choosing D as the successor, and the cost through D as the feasible distance.

What happens if D never responds to C's query? In older EIGRP implementations, C would set a timer, called the *Stuck in Active Timer;* if D does not respond to C's query within this time, C will declare the route *Stuck in Active (SIA)* and reset its neighbor adjacency with D. In newer implementations of EIGRP, C will set a timer called the *SIA Query* timer. When this timer expires, it will resend the query to D. So long as D responds that it is still working on answering the query, C will continue to wait for a response.

Where do these queries terminate? How far will an EIGRP query propagate in a network? EIGRP queries terminate at one of two points:

- When a router has no other neighbors to send queries to
- When the router receiving the query does not have any information about the destination referenced by the query

This means either at the "end of the EIGRP network" (called an Autonomous System), or one router beyond any sort of policy or configuration that hides information about specific destinations; for instance, one hop beyond the point where a route is aggregated.

EIGRP Query Range and Network Design

EIGRP has always been known as the "protocol that will work on any network," because of its large scaling properties (on the order of BGP in many cases) and apparent capability to run on "any" topology without a lot of configuration. The primary determinant of EIGRP scaling, however, is the query

range; the primary task of network design in an EIGRP network is *bounding* the scope of queries through the network. First, the query range impacts the speed at which EIGRP converges; each additional hop of query range adds some small amount of time to the overall convergence time of the network (around 200ms in most cases). Second, the query range impacts the stability of the network. The farther queries must pass through the network, the more likely it is some router along the way will not be able to answer the query immediately. Hence, the most important point in designing a network around EIGRP as a protocol is bounding queries through aggregation or filtering of some type.

Neighbor Discovery and Reliable Transport

EIGRP checks two-way connectivity between neighbors, the link MTU, and provides for the reliable transport of control plane information through the network by forming neighbor relationships. Figure 15-15 illustrates the EIGRP neighbor formation process.

The steps illustrated in Figure 15-15 are as follows:

1. A sends a multicast hello onto the link shared between A and B.
2. B places A in *pending* state; while A is in pending state, B will not send standard updates or queries to A, nor will it accept anything other than a specially formatted update from A.
3. B transmits an empty update with the *initialization bit* set to A; this packet is sent to A's unicast interface address.
4. On receiving this update, A responds with an empty update with the initialization bit set and containing an *acknowledgment;* this packet is sent to B's unicast interface address.
5. On receiving this unicast update, B places A into the connected state and begins sending updates containing individual topology table entries toward A; piggy-backed onto each packet is an acknowledgment for the previous packet received from the neighbor.

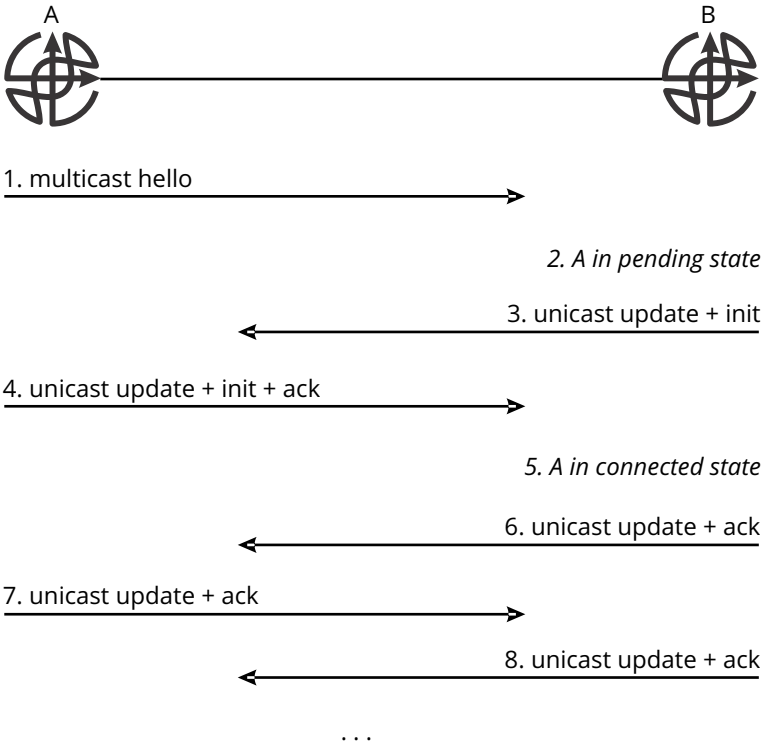


Figure 15-15 *EIGRP Neighbor Formation*

Because EIGRP does not form adjacencies with sets of neighbors, only individual neighbors, this process ensures that both unicast and multicast reachability are available between the two routers forming an adjacency. To ensure that the MTU is not mismatched on either end of the link, EIGRP pads a specific set of packets during the neighbor formation; if these packets are not received by the other router, the MTU is mismatched, and no neighbor relationship should be formed.

Note

EIGRP sends multicast hellos for neighbor discovery by default but will use unicast hellos if neighbors are manually configured.

Concluding Thoughts on EIGRP

EIGRP presents a number of interesting solutions to the problems that routing protocols encounter when sending information across a network, calculating loop-free paths, and reacting to topology changes. EIGRP is classified as a distance vector protocol using DUAL to calculate loop-free paths, and alternate loop-free paths, through the network. EIGRP advertises routes without reference to traffic flows through the network, so it is a proactive protocol.

Distance Vector Protocols and the Routing Table

In most discussions about distance vector protocols (including this one), these protocols are explained in a way that implies they operate completely separate from the routing table and any other routing processes running on the device. This, however, is not the case; distance vector protocols interact with the routing table in a way that link state protocols do not. Specifically, a distance vector protocol will not advertise a route to a destination it has not installed in the local routing table.

For instance, assume EIGRP and RIP are running on the same router. EIGRP learns about some destination and installs a route to the same destination in the local routing table. RIP learns about the same destination and attempts to install the route it has learned into the local routing table, but it fails to do so—the EIGRP route overwrites (or overrides) the RIP learned route. In this case, RIP will *not* advertise this specific route to any of its neighbors.

There are two reasons for this behavior. *First*, the route learned through EIGRP might point to a completely different next hop than the route learned through RIP. If the metrics are not set correctly, the two protocols could form a permanent forwarding loop in the network. *Second*, RIP has no way of knowing just how valid the EIGRP route is. It could be that RIP advertises the route, causing other routers to send the local device traffic destined to the advertised destination, and then the local device actually drops the packet, rather than forwarding it. This is one instance of a *routing black hole*.

To prevent either of these situations from occurring, distance vector protocols will not advertise routes that the protocol itself has not in the local routing table. If a distance vector protocol's route is overwritten for any reason, it will stop advertising reachability to the destination.

Further Reading

Bellman, Richard. "On a Routing Problem." *Quarterly of Applied Mathematics* 16 (1958): 87–90.

Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs." *Numerische Mathematik* 1, no. 1 (1959): 269–71. doi:10.1007/BF01386390.

Envedi, Gabor Sandor, Andras Csaszar, Alia Atlas, Chris Bowers, and Abishek Gopalan. *An Algorithm for Computing IP/LDP Fast Reroute Using Maximally Redundant Trees (MRT-FRR)*. Request for Comments 7811. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7811.txt.

"eRSTP—Enhanced Rapid Spanning Tree Protocol—Industrial Communication—Siemens." WCMS3Article. Accessed September 25, 2017. http://w3.siemens.com/mcms/industrial-communication/en/rugged-communication/technology-highlights/pages/erstp-enhance-rapid-spanning-tree-protocol.aspx.

Ford, L. R. *Network Flow Theory*. Santa Monica, CA: RAND Corporation, 1956.

Garcia-Luna-Aceves, J. J. "Loop-Free Routing Using Diffusing Computations." *IEEE/ACM Transactions on Networking* 1, no. 1 (February 1993): 130–41.

Hendrick, C. *Routing Information Protocol*. Request for Comments 1058. RFC Editor, 1988. https://rfc-editor.org/rfc/rfc1058.txt.

Malkin, Gary S. *RIP Version 2*. Request for Comments 2453. RFC Editor, 1998. https://rfc-editor.org/rfc/rfc2453.txt.

Malkin, Gary S., and Robert E. Minnear. *RIPng for IPv6*. Request for Comments 2080. RFC Editor, 1997. https://rfc-editor.org/rfc/rfc2080.txt.

Moore, Edward F. "The Shortest Path through a Maze." In *Proceedings of the International Symposium on Switching Theory 1957, Part II*. Cambridge, MA: Harvard University Press, 1959.

Perlman, Radia. "An Algorithm for Distributed Computation of a Spanningtree in an Extended LAN." *SIGCOMM Computer Communication Review* 15, no. 4 (September 1985): 44–53, doi:10.1145/318951.319004.

Retana, Alvaro, Russ White, and Don Slice. *EIGRP for IP: Basic Operation and Configuration*. 1st edition. Boston, MA: Addison-Wesley Professional, 2000.

Savage, Donnie, Steven Moore, James Ng, Russ White, Donald Slice, and Peter Paluch. *Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)*. Request for Comments 7868. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7868.txt.

Schrijver, Alexander. "On the History of the Shortest Path Problem." *Documenta Mathematica* Extra (2012): 155–67.

Shimbel, A. "Structure in Communication Nets." In *Proceedings of the Symposium on Information Networks*, 199–203. New York: Polytechnic Press of the Polytechnic Institute of Brooklyn, n.d.

Suurballe, J. W. "Disjoint Paths in a Network." *Networks* 4, no. 2 (1974): 125–45. doi:10.1002/net.3230040204.

"Understanding Rapid Spanning Tree Protocol (802.1w)." *Cisco*. Accessed September25,2017.https://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/24062-146.html.

Review Questions

1. Go through some specific protocols, describing each in terms of Protocols:
   - How the protocol would be broadly classified
   - Which problems they address out of the set described in the book
   - Which solutions they chose for each problem
   - Which problems the protocol does not address
     - What does this tell you about the convergence characteristics of the protocol?
     - Is there a particular use case for each protocol?
     - How does each one overlap with/differ from the ones described in the book?
   - AODV
   - TRILL
   - BABEL
   - OLSR
2. What is the specific method used by STP to prevent loops after a topology change has occurred? Can you relate this to the CAP theorem?
3. The text only describes the handling of unicast packets flowing through an STP domain. How are broadcasts and multicasts handled? What is a broadcast storm, and why is it so dangerous in a network running STP?

4. Examine STP from a complexity perspective. What simplifying assumptions are made, and how do these simplifying assumptions impact the optimization of using network resources?
5. Research the operation of the Rapid Spanning Tree Protocol (RSTP; see the "Further Reading" section for resources). What are the advantages of RSTP over the Spanning Tree Protocol?
6. Consider the VLAN extensions to STP. What are these extensions? Do they make the protocol more complex or less? Do they increase or decrease the optimal use of network resources?
7. Consider the RIP hold-down timer described in the text. Construct a network where RIP could potentially form a loop if the implementation does not support the hold-down timer.
8. Analyze triggered RIP, as described in the text, within the complexity model of state/optimization/surface. Is there an additional interaction surface introduced into RIP by triggered updates? Is there additional state? What is the optimization tradeoff?
9. EIGRP can carry two different kinds of metrics—narrow and wide. Why do these two kinds of metrics exist? What is the relationship between them?
10. EIGRP can carry two different kinds of metrics—narrow and wide. Describe the narrow to wide transition mechanism. Is this effective? Are there any problems inherent in the process? What happens if one router is never upgraded?
11. Consider the EIGRP stuck-in-active process before the SIA query was inserted in the code. Describe the process. Construct a network where EIGRP will reset an adjacency several hops away from a router that is not answering queries without the SIA query.

This page intentionally left blank

Chapter 16

Link State and Path Vector Control Planes

Learning Objectives

After reading this chapter, you should understand:

- OSI Addressing, used with IS-IS
- The basic operation of IS-IS, including topology discovery and flooding
- The basic operation of OSPF, including topology discovery and flooding
- Designated Routers and Designated Intermediate Systems
- The validation of two-way connectivity and link MTU in OSPF and IS-IS
- The basic operation of the Border Gateway Protocol

This chapter continues the discussion on distributed control planes, addressing three more routing protocols. Two of these are link state protocols, and the third is the only widely deployed path vector protocol, the Border Gateway Protocol (BGP) version 4.

Throughout this chapter, it is important to consider *why* each of these protocols is implemented the way it is. While it is always easy to become lost in the finer details of protocol operation, it is far more important to remember the problems that these protocols were designed to address and the range of possible solutions. Each protocol you study will be some combination of a moderately restricted set of available solutions; there are very few *new* solutions available; there are different combinations of solutions implemented in sometimes unique ways to solve specific sets of problems.

When reading through these high-level overviews of protocol operation, you should try to pick out the common solutions they implement and then reflect these common solutions back into the set of problems any distributed control plane must solve in order to succeed in real networks.

A Short History of OSPF and IS-IS

The Intermediate System to Intermediate System (IS-IS, or IS to IS) protocol is unique among the routing protocols in several ways. The work on IS-IS began in 1978, with the acceptance of the seven-layer networking model proposed by Honeywell Labs to the British Standards Institute, which then proposed the idea of forming a working group within the International Organization for Standardization (ISO) to standardize the communications between computers. The idea was so good that the forerunner of the International Telecommunications Union (the ITU) formed a parallel working group to work with the ISO in building these standards. These committees, their subcommittees, and sub-subcommittees, *ad infinitim,* created a suite of standard protocols. Among these protocols was IS-IS.

Open Shortest Path First (OSPF) was originally conceived as an alternative to IS-IS, designed specifically to interact with IPv4 networks. In 1989, the first OSPF specification was published by the Internet Engineering Task Force, and OSPFv2, a much improved specification, was published in 1998 as RFC2328. OSPF was certainly the more widely used protocol, with early implementations of IS-IS being barely exercised in the real world. There were some back-and-forth arguments, and many features were "stolen" from one protocol into the other (in both directions).

In 1993, Novell, a heavyweight in the networking world at the time, used IS-IS as the basis for a replacement to the Netware native routing protocol. Novell's transport layer, Internet Packet Exchange (IPX), ran on a large number of devices at the time, and the ability for a single protocol to route multiple transport protocols was a definitive advantage in the networking market (the Enhanced Interior Gateway Routing Protocol, or EIGRP, can also route IPX). This replacement protocol was based on IS-IS; to implement the Novell's new protocol, many vendors simply rewrote their implementations of IS-IS, greatly improving them in the process. This rewrite made IS-IS attractive to large-scale Internet service providers, so as they moved off the Routing Information Protocol (RIP), they would often move onto IS-IS instead of OSPF.

Note

Parts of this history rely on Dave Katz's presentation at the North American Network Operators' Group (NANOG) in the summer of 2000.[1] Other parts rely on the history given in *IS-IS: Deployment in IP Networks*.[2]

1. Katz, "OSPF and IS-IS: A Comparative Anatomy."
2. White and Retana, *IS-IS: Deployment in IP Networks*.

The Intermediate System to Intermediate System Protocol

In the Intermediate System to Intermediate System (IS-IS) protocol, a *router* is called an Intermediate System (IS), and a host is called an End System (ES). The original design of the suite was for each *device,* rather than interface, to have a single address. Services and interfaces on a device would then have a Network Service Access Point (NSAP), used to direct traffic to a specific service or interface. From an IP perspective, then, IS-IS was originally designed within a host routing paradigm; Intermediate and End Systems communicated directly using the End System to Intermediate System (ES-IS) protocol, allowing IS-IS to discover the services available on any connected End System, as well as to match lower interface addresses with higher layer device addresses.

Another interesting aspect of the design of IS-IS is it runs at the link layer; it did not make a lot of sense to the designers of the protocol to run the control plane to provide reachability for a transport system over the transport system itself. Routers will not forward IS-IS packets, as they are parallel to IP in the protocol stack and transmitted to link local addresses. When IS-IS was developed, most links were very low speed, so the extra encapsulation was also thought to be wasteful. Links also failed quite often, losing and corrupting packets; hence the protocol was designed to withstand errors in transmission and packet loss.

OSI Addressing

As IS-IS was developed for a different transport protocol suite, it does not use Internet Protocol (IP) addresses to identify devices. Instead, it uses an Open Systems Interconnect (OSI) address to identify both Intermediate and End Systems. The OSI

addressing scheme is somewhat complex, including identifiers for the authority allocating the address space, a two-part domain identifier, an area identifier, a system identifier, and a service selector (the N Selector); many of these parts of the OSI address are variable length, making the system even more difficult to understand. Within the IP world, however, only three parts of this address space are used.

- The Authority Format Identifier (AFI), Initial Domain Identifier (IDI), High-Order Domain Specific Part (HO-DSP), and the area are all treated as a single field called the *area*.
- The System Identifier is still treated as the *system identifier.*
- The N Selector, or NSAP, is generally ignored (although there is an interface identifier that is similar to the NSAP used in some specific situations).

Intermediate system addresses, then, normally take the form illustrated in Figure 16-1.

In Figure 16-1:

- The dividing point between the system identifier and the remainder of the address is at the sixth octet, or twelve hexadecimal digits from the right side; everything to the left of the sixth octet is considered part of the area address.

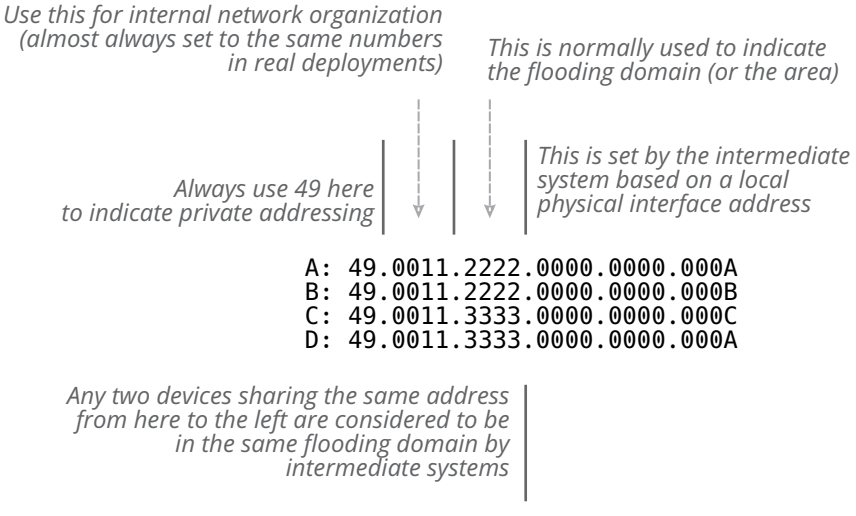


Figure 16-1 *An Overview of the OSI Addressing Scheme Used in IS-IS*

- If the N Selector is included, it is a single octet, or two hexadecimal digits, to the right of the system identifier; for instance, if an N Selector were included for address A, it might be 49.0011.2222.0000.0000.000A.00.
- If an N Selector is included in the address, you need to skip the N Selector when counting over six octets to find the start of the area address.
- A and B are in the same flooding domain because they share the same digits from the seventh octet to the leftmost octet in the address.
- C and D are in the same flooding domain.
- A and D represent different systems, although their system identifier is the same; this sort of addressing, however, can be very confusing, and so is not used in real IS-IS deployments (at least not by thoughtful system administrators).

You may find this addressing scheme more difficult than IP to work with, even if you work with IS-IS as a routing protocol on a regular basis. There is a major advantage to using an addressing scheme that is different from the one being used at the transport level in a network, however; it is much easier to differentiate between the *kinds* of devices on the network, and it is much easier to separate nodes from destinations when thinking through Dijkstra's Shortest Path First (SPF) algorithm.

Marshalling Data in IS-IS

IS-IS uses a fairly interesting mechanism to marshal data for transmission between intermediate systems. Each IS generates three kinds of packets:

- Hello packets
- Sequence Number Packets (Partial, PSNPs; and Complete, CSNPs)
- A single Link State Packet (LSP)

The single LSP contains all the information about the IS itself, any reachable intermediate systems, and any reachable destinations attached to the IS. This single LSP is formatted into Type Length Vectors (TLVs), which contain various bits of information. Some of the more common TLVs include the following:

- **Types 2 and 22:** Reachability to another intermediate system
- **Types 128, 135, and 235:** Reachability to an IPv4 destination
- **Types 236 and 237:** Reachability to an IPv6 destination

There are multiple types because IS-IS originally supported 6-bit metrics (most processors at the time of the protocol's definition could hold only 8 bits at a time, and two bits were "stolen" from this field size to carry information about whether the route was internal or external as well as other information). Over time, as link speeds increased, various other metric lengths were introduced, including 24- and 32-bit metrics, to support *wide metrics*.

The single LSP carrying all IS, IPv4, and IPv6 reachability information—as well as, potentially, MPLS tags and other information—will not fit into a single MTU-sized packet. To actually send information over the network, IS-IS breaks up the LSP into fragments. Each fragment is treated as a separate entity in the flooding process. If one fragment changes, just the changed fragment is flooded through the network, rather than the entire LSP. Because of this scheme, IS-IS is very efficient at flooding new topology and reachability information without using more than the minimum amount of bandwidth required.

Neighbor and Topology Discovery

While IS-IS was originally designed to learn about network reachability through the ES-IS protocol, when IS-IS is used to route IP, it "does as the IP protocols do," and learns about reachable destinations through the local configuration of each device, and through redistribution from other routing protocols. Hence IS-IS is a proactive protocol, learning about and advertising reachability without waiting on packets to be transmitted and forwarded through the network.

Neighbor formation in IS-IS is fairly simple; Figure 16-2 illustrates the process.

In Figure 16-2:

1. IS A transmits a hello toward B. This hello contains a list of neighbors heard from, which will be empty; the hold time setting B should use for A; and it is padded to the local interface Maximum Transmission Unit (MTU) for the link. Hello packets are padded only until the adjacency formation process is complete; not every hello packet is padded to the full MTU of the link.
2. IS B transmits a hello toward A. This hello contains a list of neighbors heard from, which would include A; the hold time setting A should use for B; and it is padded to the local interface MTU.
3. Because A is in B's "heard neighbor" list, A will consider B up and move to the next stage of neighbor formation.
4. Once A has included B in the "heard neighbor" list in at least one hello, B will consider A up and move to the next stage of neighbor formation.

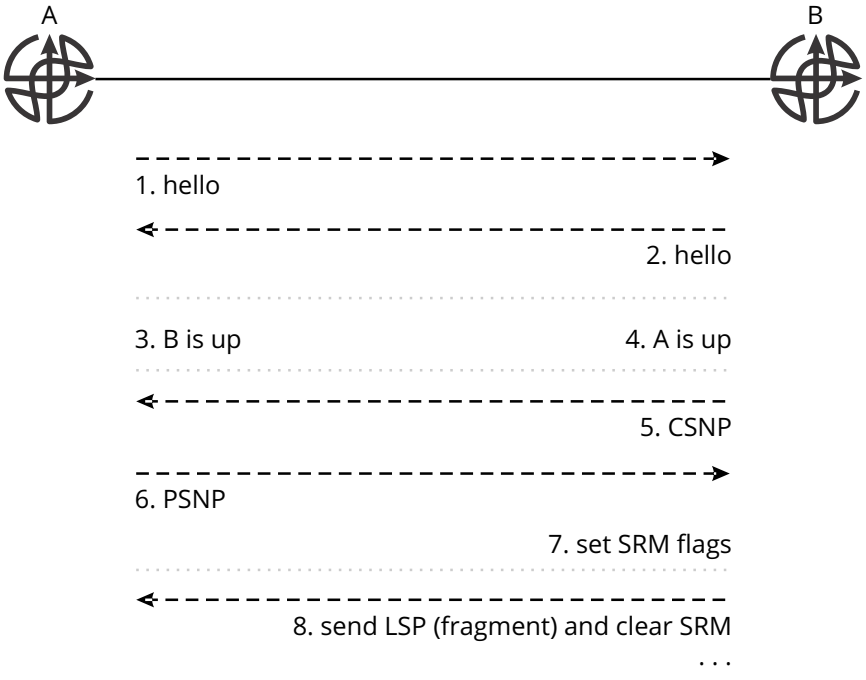


Figure 16-2 *IS-IS Neighbor Formation Process*

5. B will send a complete list of all the entries it has in its local topology table (B describes the LSPs it has in its local database). This list is sent in a Complete Sequence Number Packet (CSNP).
6. A will examine its local topology table, comparing it to the complete list sent by B; any topology table entries (LSPs) it does not have, it will request from B using a Partial Sequence Number Packet (PSNP).
7. When B receives a PSNP, it will set the Send Route Message (SRM) flag on any entry in its local topology table (LSPs) A has requested.
8. The flooding process will later walk the local topology table looking for entries with the SRM flag set; it will flood these entries, synchronizing the databases at A and B.

Note

The process described here includes modifications made by RFC5303, which specifies a three-way handshake, and hello padding, which was added to most implementations around 2005.

Setting the SRM flag marks the information for flooding, but how does flooding actually take place?

Reliable Flooding

For Dijkstra's SPF algorithm (or any other SPF algorithm) to work correctly, every IS in the flooding domain must share a synchronized database. Any inconsistency in the database between two intermediate systems opens the possibility of a permanent routing loop. How does IS-IS ensure connected intermediate systems have synchronized databases? This section describes the process on point-to-point links; the following section will describe the modifications made to the flooding process on multiaccess (such as Ethernet) links.

IS-IS relies on a number of fields in the LSP header to ensure two intermediate systems have synchronized databases; Figure 16-3 illustrates these fields.

In Figure 16-3:

- The packet length contains the total length of the packet in octets. For instance, if this field contains 15, the packet is 15 octets in length. The packet length field is 2 octets, so it can describe a packet up to 65,536 octets long—longer than even the largest link MTUs.
- The remaining lifetime field is also two octets and contains the number of seconds for which this LSP is valid. This forces the information carried in the LSP to be refreshed occasionally, an important consideration on older transmission technologies, where bits can be flipped, packets can be truncated, or information carried through the link can otherwise be corrupted. The advantage of having a timer that counts down, rather than up, is each IS in the network can determine how long its information should remain valid independently of every other IS. The disadvantage is there is no clear way to disable the functionality described. However, 65,536 seconds is a long time—1,092 minutes, or around 18 hours. Reflooding every LSP fragment in the network every 18 hours or so poses very little burden on the operation of the network.

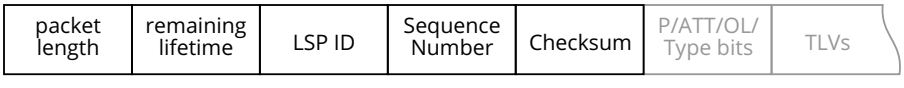


Figure 16-3 *IS-IS LSP Header Fields*

- The LSP ID describes the LSP itself. Actually, this field describes the fragment, as it contains the originating system identifier, the pseudonode identifier (the function of this identifier is described later), and the LSP number, or rather the LSP fragment number. The information contained in a single LSP fragment is treated as "one unit" throughout the entire network; a single LSP fragment is never "refragmented" by some other IS. This field is normally 8 octets.
- The Sequence Number describes the *version* of this LSP. The sequence number ensures every IS in the network has the same information in its local copy of the topology table. It also ensures an attacker (or broken implementation) cannot replay older information to replace new.
- The Checksum ensures the information carried in the LSP fragment has not been modified during transmission.

Note

The term *LSP* is often used for two different things: the complete LSP describing all the connectivity and other information about a particular IS, and each fragment of the LSP as it is transmitted through the network. Hence, an LSP is split into LSPs, each of which is transmitted through the network. This can be confusing; this book will always call the LSP as it is transmitted the LSP fragment or *fragment,* and the LSP as generated by the IS, describing its entire connectivity, simply an LSP.

Flooding is described using Figure 16-4.
In Figure 16-4:

1. A is connected to 2001:db8:3e8:100::/64. A builds a new fragment describing this newly reachable destination.
2. A sets the SRM flag on this fragment toward B.
3. The flooding process, at some point in the future (usually a matter of milliseconds), will examine the topology table and flood any entries with the SRM flag set.
4. Once the new entry is placed in its topology table, B will create a CSNP describing its entire database and send this to A.

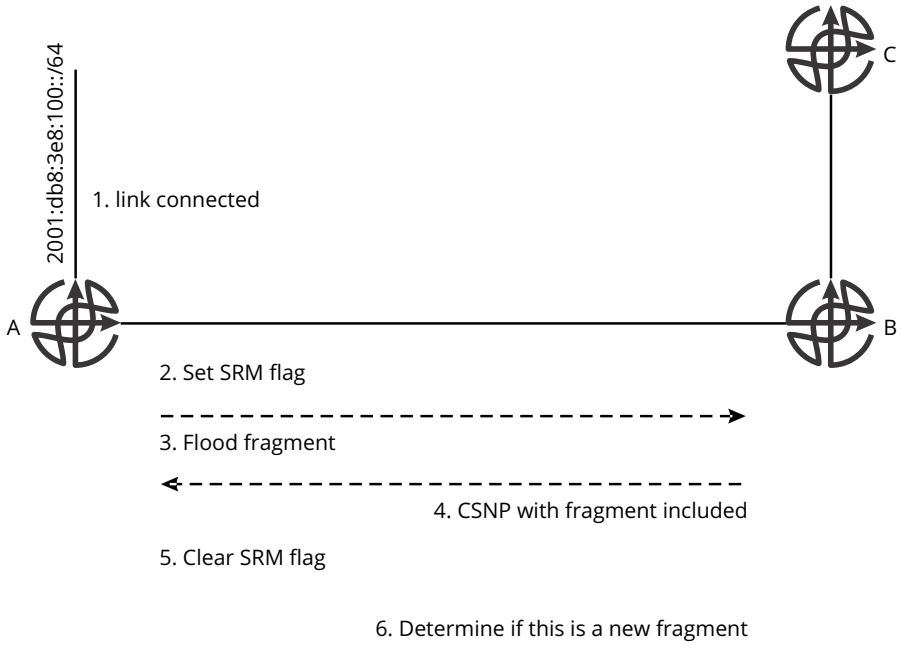


Figure 16-4 *IS-IS Flooding*

5. On receiving this CSNP, A clears its SRM flag toward B.
6. B verifies the checksum and compares the received fragment to existing entries in its topology table. As there is no other entry matching this system and fragment identifier, it will place the new fragment in its local topology table. Given this is a new fragment, B will initiate the flooding process toward C.

What about removing information? There are three ways information can be removed from the IS-IS flooding system:

- The originating IS can originate a new fragment without the relevant information and with a higher sequence number.
- If the entire fragment no longer contains any valid information, the originating IS can flood the fragment with a remaining lifetime of 0 seconds. This causes each IS in the flooding domain to reflood the zero age fragment and remove it from consideration for future SPF calculations.

- If the remaining lifetime timer in a fragment times out at any IS, the fragment is flooded with a zero age remaining lifetime. Each IS receiving this zero-aged fragment will verify it is the most recent copy of the fragment (based on the sequence number), set the remaining lifetime to its local copy of the fragment to zero seconds, and reflood the fragment. This is called *flushing* a fragment from the network.

When an IS sends a CNSP in reply to a fragment it has received, it actually verifies the entire database, rather than just the one fragment it received. Each time a fragment is flooded through the network, the entire database is checked between each pair of intermediate systems.

Concluding Thoughts on IS-IS

IS-IS can be described as

- Using flooding to synchronize the database at every intermediate system in the flooding domain (a link state protocol).
- Calculating loop-free paths using Dijkstra's SPF algorithm.
- Learning about reachable destinations through configuration and local information (a proactive protocol).
- Validating two-way connectivity in neighbor formation by carrying a list of "neighbors seen" in its hello packets.
- Removing information from the flooding domain through a combination of sequence numbers and remaining lifetime fields in each fragment.
- Verifying the MTU of each link by padding the initially exchanged hello packets.
- Validating the correctness of the information in the synchronized database through checksums, periodic reflooding, and database descriptions exchanged between intermediate systems.

IS-IS is a widely deployed routing protocol that has proven capable in a wide range of network topologies and operational requirements.

The Open Shortest Path First Protocol

In 2013, a version of OSPF was published for routing IPv6. Known as OSPFv3, it was originally specified in RFC2740, which was later replaced by RFC5340, and updated by later standards. OSPFv3 is the version assumed for any specific details of OSPF operation in this chapter.

Marshalling Data in OSPF

Like many of the other protocols developed in the early days of network engineering, OSPF was designed to minimize the processing power, memory, and bandwidth required to carry routing information for IPv4 through the network. Two specific choices made early on in the OSPF design process reflect this concern with resource utilization:

- OSPF relies on fixed length fields to marshal data, rather than TLVs. This saves the overhead of carrying the additional metadata in the form of Type Length Value (TLV) headers, reduces processing requirements by allowing fixed sized in memory data structures to be matched with packets as they are received off the wire, and reduces the size of OSPF data on the wire.
- OSPF breaks the topology database up into multiple kinds of data, rather than relying on a single LSP with TLVs. This means each kind of information—reachability, topology, etc.—is carried in a unique packet format.

Note

More recent work in OSPFv3 replaces specific fields in the current fixed field LSAs with TLV-based LSAs. See *OSPFv3 LSA Extendibility* for more information.[3]

3. Mirtorabi et al., "OSPFv3 LSA Extendibility."

Each type of information OSPF can carry is carried in a different *type* of Link State Advertisement (LSA). Some of the more notable types of LSAs are as follows:

- **Type 1:** code 0x2001, Router LSA
- **Type 2:** code 0x2002, Network LSA
- **Type 3:** code 0x2003, Inter-Area Prefix LSA

- **Type 4:** code 0x2004, Inter-Area Router LSA
- **Type 5:** code 0x4005, AS-external LSA
- **Type 7:** code 0x2007, Type-7 (NSSA) LSA

There are a number of other types of LSAs, including opaque data, multicast group membership, and scoped flooding LSAs (such as to a single neighbor, a single link, or a single flooding domain).

Each OSPF router generates precisely one Router LSA (type 1); this LSA describes any neighbors adjacent to the advertising router, as well as any connected reachable destinations. The state of the links to these neighbors and destinations is inferred from the advertisement of the neighbors and destination; in spite of the name "link state," links are not advertised as a separate "thing" (this is often a point of confusion). If the Router LSA becomes too large to fit within a single IP packet (because of the link MTU), it will be split into multiple IP fragments for transmission router to router. Each router reassembles the entire Router LSA before processing it locally and floods the entire Router LSA if it changes.

OSPF uses a few different packet types, as well—these are not the same as the LSA types. Rather, these can be thought of as different "services" within OSPF or, perhaps, as different "port numbers" running on top of User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP).

- The hello is a type 1. These are used for neighbor discovery and liveness.
    - The Database Descriptor (DBD) is a type 2. These are used to describe the local topology table.
    - The Link State Request (LSR) is a type 3. These are used to request specific Link State Advertisements from an adjacent router.
    - The Link State Update (LSU) is a type 4. These are used to carry the Link State Advertisements described in this section.
- The Link State Acknowledgment is a type 5. This is simply a list of LSA headers; any LSA listed in this packet is acknowledged as being received by the transmitting router.

Neighbor and Topology Discovery

As a link state protocol, OSPF must ensure every router within an area (a flooding domain) has the same database to calculate loop-free paths from. Any variation in the shared topology database can result in a routing loop that will last as long as the

variation in the shared topology database exists. One purpose for OSPF neighbor formation, then, is to ensure the reliable flooding of topology information through the network. A second reason for OSPF neighbor formation is to discover the network topology, by determining which routers are adjacent to the local router. Figure 16-5 illustrates the OSPF neighbor formation process.

In Figure 16-5:

1. B sends a hello packet to A.
2. Since B's hello contains an empty *neighbors seen* list, A places B into init state and adds B to its *neighbors seen* list.
3. A sends a hello with B in its *neighbors seen* list.
4. B receives A's hello and sends a hello with A in its *neighbors seen* list.
5. A receives this hello; as A itself is in the *neighbors seen* list, A places B into the two-way state. This means that A has verified two-way connectivity exists between itself and B.
6. If there are a DR and BDR being elected on this link (the function of the DR and BDR is considered in a moment), the election takes place after step 5. Once the election is completed, the DR and BDR are placed in the exstart state. During this state, the master and slave are elected for the exchange of DBDs and LSAs. Essentially, the master controls the flow of DBDs and LSAs between the newly adjacent routers. Adjacent routers on a point-to-point link technically skip directly to full state at this point.
7. B is moved to the exchange state.
8. A sends a set of DBDs describing its database to B; B sends a set of DBDs describing its database to A.
9. A sends a link state request to B for each LSA B describes, and A does not have a copy of it in its local topology table.
10. B sends an LSA for each Link State (LS) request from A.
11. Once the databases are synchronized, B is moved to full state.

The OSPF neighbor formation process verifies the MTUs on both ends of the link match by carrying the MTU of the outbound interface in the hello; if the two hello packets do not match in MTU size, the two OSPF routers will not form an adjacency.

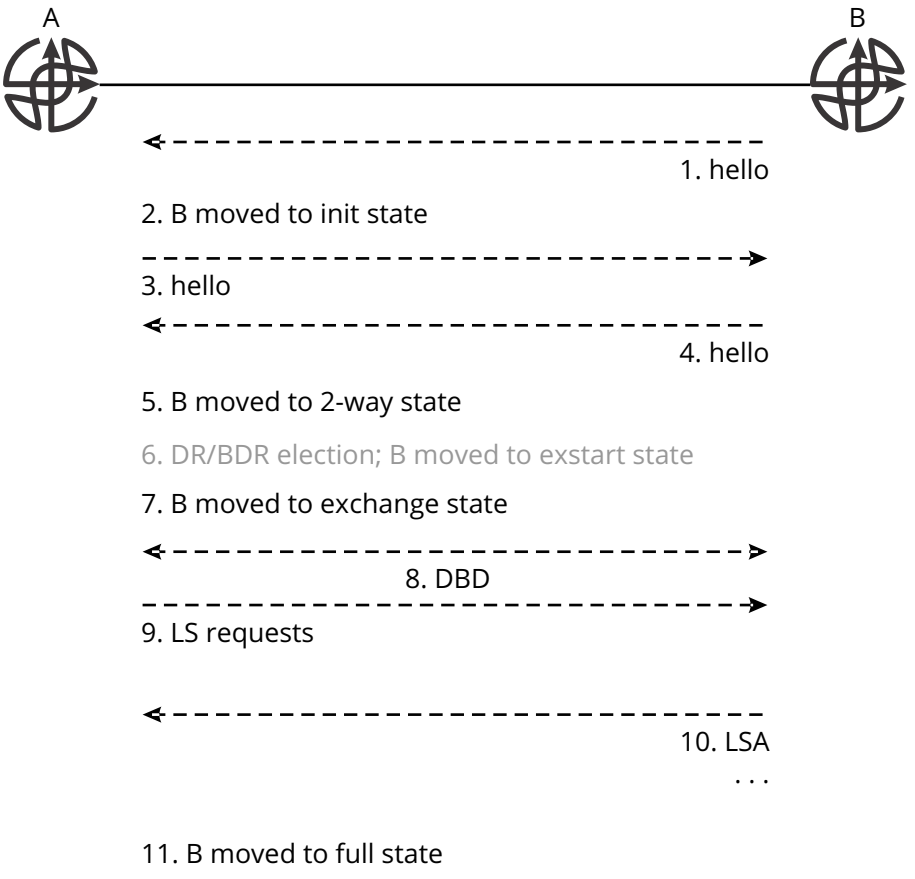


Figure 16-5 *OSPF Neighbor Formation*

Reliable Flooding

OSPF must not only ensure the initial exchange of topology information is completed, but it must also ensure ongoing changes in the network topology are flooded to every router in the flooding domain. Figure 16-6 illustrates the OSPF LSA header; examining this header will yield some important clues about the way OSPF reliably floods topology and reachability information through the network.

In Figure 16-6:

- The LS Age is (roughly) the number of seconds since this Link State Advertisement was generated. This number counts *up,* rather than *down*. When the LS Age reaches the MAXAGE setting (on any router, not just the originating router), the router will increment the sequence number by 1, set the LS Age to the maximum

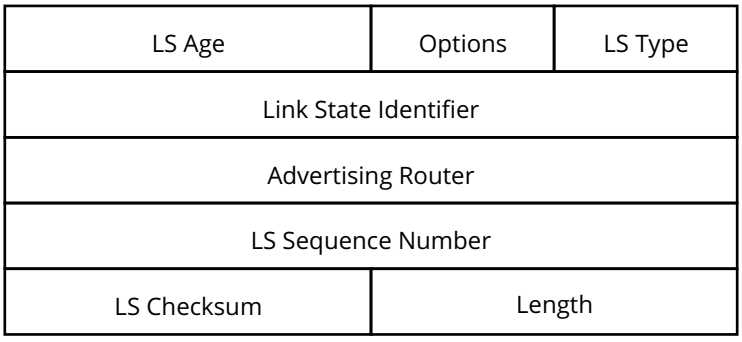


Figure 16-6 *OSPF LSA header*

age, and reflood the LSA throughout the network. This removes older topology and reachability information that has not been refreshed in a while. The router that originates any particular LSA will refresh its LSAs some number of seconds before this LSA Age field reaches the maximum; this is the *LS refresh interval*.

- The Link State Identifier is a unique identifier assigned by the originating router to describe this LSA. It is normally the link address, or some local link layer address (such as an Ethernet Media Access Control, or MAC, address).
- The Advertising Router is the router ID of the originating router. This is often confused with an IP address, as it is often derived from a locally configured IP address—but it is *not* an IP address.
- The Link State Sequence Number indicates the version of the LSA. Generally, higher numbers mean newer versions, although there are earlier versions of OSPF that use a circular number space, rather than an absolutely incrementing one. Implementations that use an absolutely incrementing number space restart the OSPF process if the end of the number space is reached.
- The Link State Checksum is a checksum computed across the LSA used to catch errors in transmission or storage of the information.

Figure 16-7 is used to examine the flooding process.
In Figure 16-7:

1. The link to 2001:db8:3e8:100::/64 is configured, brought up, connected, etc., at A.
2. A rebuilds its Router LSA (type 1) to contain this new reachability information, packages it into an LSU (which may be fragmented while being placed into IP packets), and floods it to B.

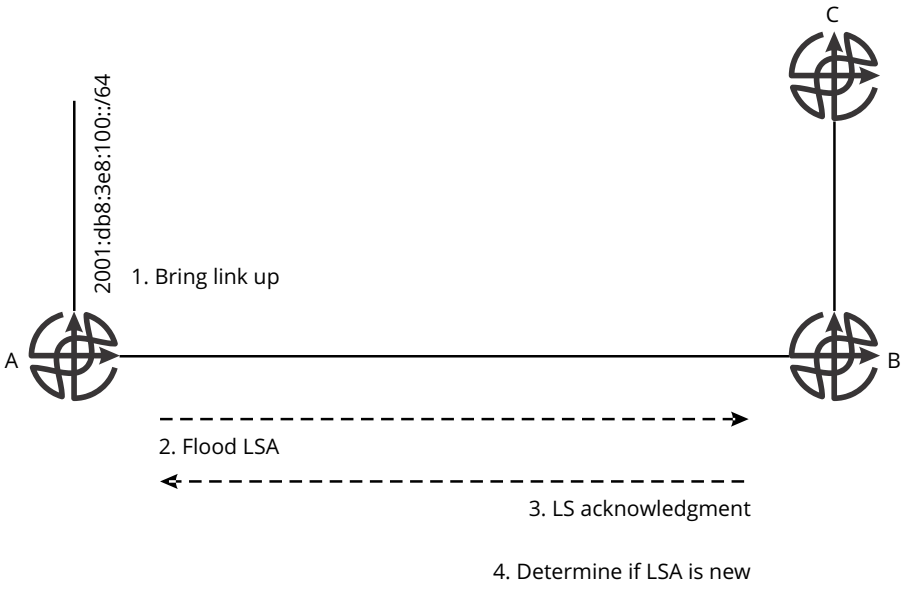


Figure 16-7 *OSPF Flooding*

3. B receives this LSA and acknowledges its receipt with a link state acknowledgment. A will resend the LSA if B does not acknowledge it quickly enough.
4. B will now examine its topology table to determine if this LSA is new or a copy of one it already has. B determines this primarily by examining a sequence number included in the LSA itself. If this is a new (or updated) LSA, B will initiate the same process to flood the changed LSA to C.

Concluding Thoughts on OSPF

OSPF can be described as

- Learning about reachable destinations through configuration and local information (a *proactive protocol)*
- Using flooding to synchronize the database at every intermediate system in the flooding domain (a *link state protocol*)
- Calculating loop-free paths using *Dijkstra's SPF algorithm*
- *Validating two-way connectivity in neighbor formation* by carrying a list of "neighbors seen" in its hello packets

- *Validating the MTU* at adjacency formation by carrying the MTU in the hello packet

OSPF is widely used in small- and large-scale networks, including retail, service provider, financial, and many other businesses.

Common Elements of OSPF and IS-IS

The preceding sections have considered those aspects of OSPF and IS-IS that are different enough to warrant separate explanations. There are, however, a number of things OSPF and IS-IS have implemented in similar enough ways to consider their solutions as simple variants. These include the handling of multiaccess links, the way the Shortest Path Tree is conceptualized, and the way two-way connectivity checks are handled.

Multiaccess Links

Multiaccess links, such as Ethernet, are links where attached devices "share" the available bandwidth, and each device can send packets directly to any other device connected to the same link. Multiaccess links pose special challenges for protocols that synchronize a database across the link; Figure 16-8 is used to explain.

One option a protocol could use when running over a multiaccess link is to simply form adjacencies as it normally would over a point-to-point link. For instance, in Figure 16-8:

- A can form an adjacency with B, C, and D.
- B can form an adjacency with A, C, and D.
- C can form an adjacency with A, B, and D.
- D can form an adjacency with A, B, and C.

If this pattern of adjacency formation is used, when A receives a new LSP fragment (IS-IS) or LSA (OSPF) from some router not connected to the shared link:

- A will transmit the new fragment or LSA to B, C, and D separately.
- When B receives the fragment or LSA, it will transmit the new fragment or LSA to C and D separately.
- When C receives the fragment or LSA, it will transmit the new fragment or LSA to D.

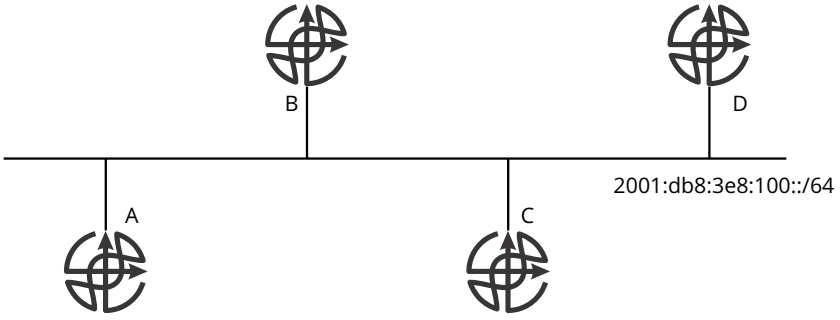


Figure 16-8 *Multiaccess Network to Explain IS-IS Operation*

Given the transmission of each fragment or LSA, and the following CSNP or acknowledgment to ensure the local database is synchronized at each router, a large number of packets must cross the shared link to ensure every device's database is synchronized. To reduce the flooding on multiaccess links, IS-IS and OSPF elect a single device that is responsible for ensuring every device connected to the link has a synchronized database. In Figure 16-8, for IS-IS:

- A single device is elected to manage flooding on the link. In IS-IS, this device is called the Designated Intermediate System (DIS).
- Each device with new link state information sends the fragment to a multicast address so every device on the shared link will receive it. None of the devices connected to the link send acknowledgments of any kind when they receive the updated fragment.
- The DIS sends out a copy of its CSNP on a regular basis to the same multicast address, so every device on the multiaccess link receives a copy of it.
- If any device on the shared link finds it is missing some specific fragment, based on the description of the DIS's database in the CSNP, it will send a PSNP onto the link requesting the missing information.
- If any device on the shared link finds it has information the DIS does not have, based on the description of the DIS's database in the CSNP, it will flood the missing fragment onto the link.

In this way, new link state information is flooded across the link a minimal number of times. In Figure 16-8, for OSPF:

- A single device is elected to manage flooding on the link, called the Designated Router (DR). A backup device is elected, as well, called the Backup Designated Router (BDR—creative, right?).

- Each device with new link state information floods it to a special multicast address monitored by the DR and BDR (all-DR-routers).
- The DR receives this LSA, examines it to determine if it contains new information, and then refloods it to a multicast address that all the OSPF routers on the link listen to (all-SPF-routers).

The election of a DIS or DR does not, however, just impact the flooding of information on the multiaccess link; it also impacts the way SPF is calculated through the link. Figure 16-9 illustrates.

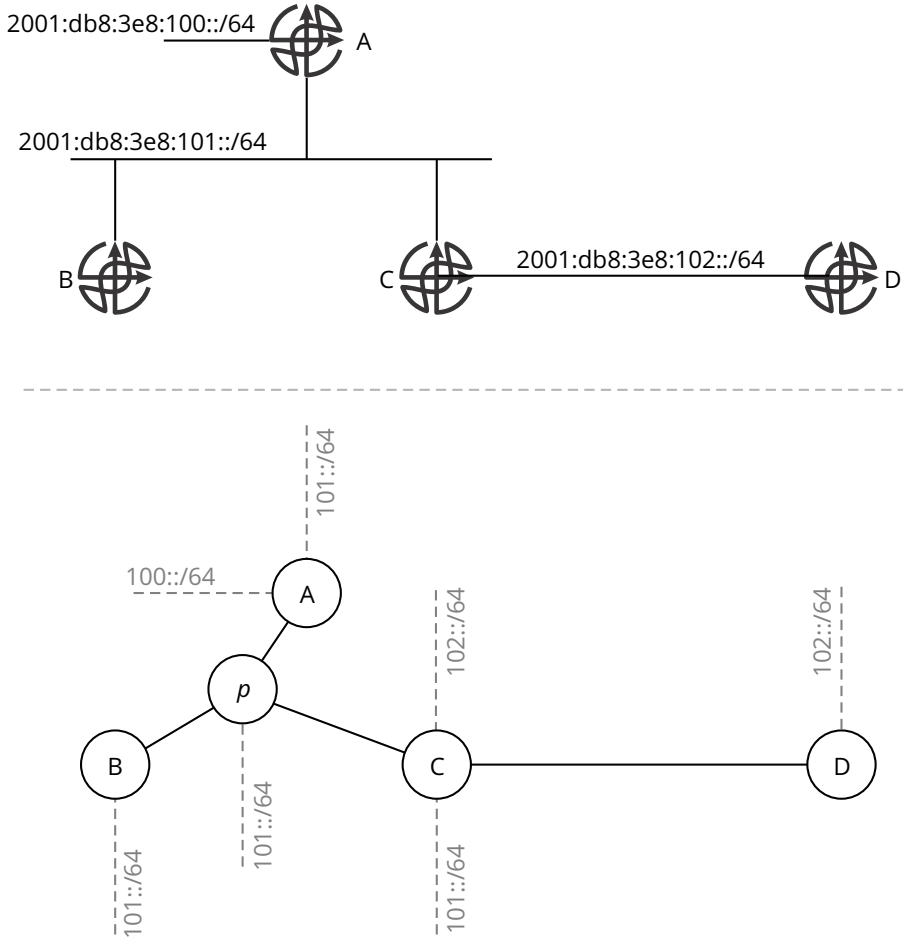


Figure 16-9 *The DIS, DR, and SPF Calculation*

In Figure 16-9, A is elected as the DIS or DR for the multiaccess circuit. A not only ensures every device on the link has a synchronized database, but it also creates a pseudonode, or p-node, and advertises it as if it were a real device attached to the network. Each of the routers connected to the shared link advertises connectivity to the p-node, rather than to each of the other connected systems.

In IS-IS, A creates an LSP for the p-node; this p-node advertises a zero-cost link back to each device attached to the multiaccess link. In OSPF, A creates a Network LSA (type 2).

Without this p-node, the network looks like a full mesh to the other intermediate systems in the flooding domain, as shown on the left side of Figure 16-9. With the p-node, the network appears to be a hub-and-spoke network, with the p-node as the hub. Each device advertises a link toward the p-node, with the link cost being set to the local interface cost onto the shared link. The p-node, in return, advertises a zero-cost link back to each device connected to the shared link. This reduces the complexity of calculating SPF across large-scale multiaccess links.

Conceptualizing Links, Nodes, and Reachability in Link State Protocols

One confusing aspect of link state protocols is how the nodes, links, and reachability interact with one another. Figure 16-10 illustrates.

In both OSPF and IS-IS, the nodes and links are used to be a Shortest Path Tree, as shown in the darker, solid lines. The dashed lines show how reachability information is attached to each node. Every node connected to a particular reachable destination advertises the destination—not just one of the two nodes connected to a point-to-point link, but both of them. Why is this?

The primary reason is this is just the easiest solution to advertising the reachable destinations. If you wanted to build a routing protocol that only advertised each reachable destination as connected to a single device, you would need to find some way to elect which of the connected devices should advertise the reachable destination. Further, if the elected device fails, then some other device must take over advertising the reachable destination, which can take time and impact convergence in a negative way. Finally, by allowing each device to advertise reachability to all connected destinations, you can actually find the shortest path to each destination.

That each device advertises each locally reachable destination is difficult for some engineers to wrap their minds around, however.

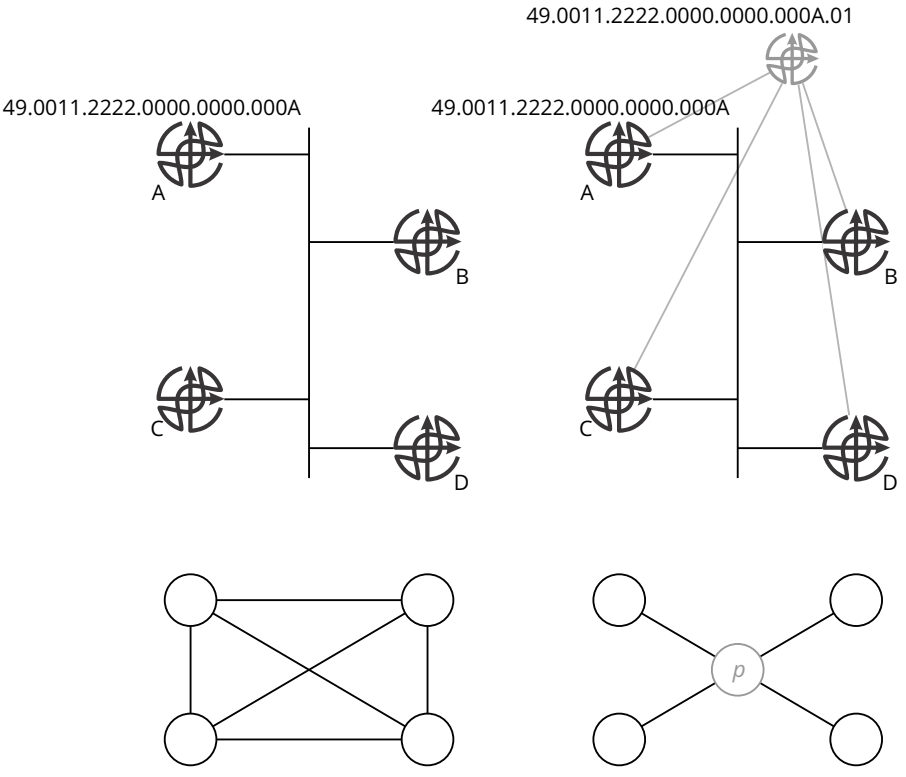


Figure 16-10 *Conceptualizing Nodes, Links, and Reachability in Link State Protocols*

Validating Two-Way Connectivity in SPF

Two-way connectivity is a problem for control planes in two distinct places: between adjacent devices and when calculating loop-free paths through the network. Both IS-IS and OSPF also ensure two-way connectivity is in place when computing loop-free paths.

The essential element is a *backlink check*. Figure 16-11 illustrates.

In Figure 16-11, the direction of each link is labeled with an arrow (or set of arrows). The [A,B] link is unidirectional toward A; the remaining links are two-way connected (bidirectional). When computing SPF, D will do the following:

- When processing C's link state information, note C claims to be connected to B. D will find B's link state information and check to make certain B also claims to be connected to C. In this case, B does claim to be connected to C, so D will use the [B,C] link.

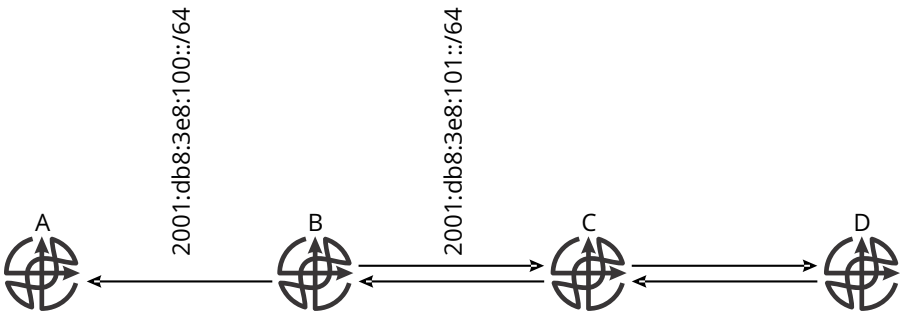


Figure 16-11 *Computing the Back Link to Test for Two-Way Connectivity*

- When processing B's link state information, note B claims to be connected to A. Examining A's link state information, however, the D cannot find any information from A claiming to be connected to B. Because of this, D will not use the [A,B] link.

This check is normally done either before a link is moved to the TENT or before a link is moved from the TENT onto the PATH.

Border Gateway Protocol

In January 1989 at the 12th Internet Engineering Task Force (IETF) meeting in Austin, Texas, Yakov Rekhter and Kirk Lougheed sat down at a table and in a short time a new exterior gateway routing protocol was born, the BGP. The initial BGP design was recorded on a napkin rumored to have been heavily spattered with ketchup. The design on the napkin was expanded to three handwritten sheets of paper from which the first interoperable BGP implementation was quickly developed.

BGP was originally designed to be an Exterior Gateway Protocol (EGP), which means it was intended to connect networks, or Autonomous Systems (ASes), rather than devices. If BGP is an EGP, this must mean that the other routing protocols, like RIP, EIGRP, OSPF, and IS-IS, must be Interior Gateway Protocols (IGPs)—a designation that "stuck." Clearly defining interior and exterior gateways has proven useful in designing and operating large-scale networks. BGP is unique among the widely deployed protocols in its loop-free path calculation. There are three widely used distance vector protocols (Spanning Tree, RIP, and EIGRP). There are two widely used link state protocols (OSPF and IS-IS). And there are many more examples of these two types of protocols developed and deployed in what might be considered niche markets. BGP, however, is the only widely deployed path vector protocol.

What are the most important goals for an EGP? The first is obviously selecting loop-free paths, but this clearly does not mean the *shortest* path. The reason the shortest path is not as important in an EGP as it is in an IGP is that EGPs are used to connect entities, such as service providers, content providers, and corporate networks. Connecting networks at this level means focusing on *policy,* rather than efficiency—in complexity terms, increasing state through policy mechanisms while reducing overall network optimization in pure traffic-carrying terms.

BGP policy mechanisms will not be considered here in any depth; some basic policy concepts are considered in Chapter 17, "Control Plane Policy." This section focuses on transport, peering, advertisement, and the BGP decision process.

BGP Peering

BGP does not provide any sort of reliable transport. Instead, BGP relies on TCP to carry information between BGP peers. Using TCP ensures

- MTU detection is handled, even for connections crossing several hops (or routers).
- Flow control is taken care of by the underlying transport, so BGP does not need flow control directly (although most BGP implementations do interact with the TCP stack on the local host to improve throughput for BGP specifically).
- Two-way connectivity between peers is ensured by the three-way handshake implemented in TCP.

Even though BGP relies on an underlying TCP connection for many of the functions control planes must solve in building adjacencies, there are still a number of functions TCP cannot provide. Therefore, a fuller look at the BGP peering process is still in order; Figure 16-12 illustrates.

In Figure 16-12:

1. The BGP peering session begins in the idle state.
2. A sends a TCP open on port 179; B responds to an ephemeral port on A. After the TCP three-way handshake is completed (the TCP session is successful), BGP moves the peering state to connect. If the peering session is being formed across some type of state-based filtering, such as a firewall, it is important that the TCP open be transmitted from the "inside" of the filtering device.

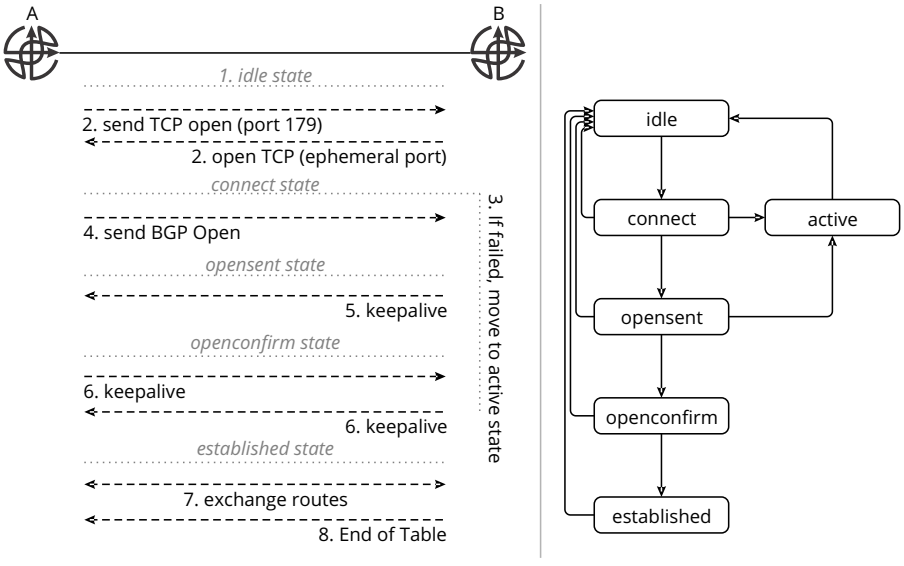


Figure 16-12 *The BGP Peering Process*

3. If the TCP connection fails, the BGP peering state is moved to active.
4. A sends a BGP open to B and moves B to the opensent state. At this point, A is waiting on B to send a keepalive. If B does not send a keepalive within a specific period, A will move the session back to the idle state. The open message contains a number of parameters, such as which address families the two BGP speakers support and the hold timer. This is called capabilities negotiation. The lowest (minimum) hold timer of the two advertised is selected as the hold timer for the peering session.
5. When B sends A a keepalive, A moves B to the openconfirm state.
6. At this point, A will send B a keepalive to verify the connection. When A and B receive one another's keepalives, the peering session will move to the established state.
7. The two BGP speakers exchange routes, so their tables are up to date. A and B only exchange their best paths, unless some form of BGP multipath is supported and configured on the two speakers.
8. To notify A it has finished sending its entire local table, B sends A an End of Table (EOT) or End of RIB (EOR) signal.

There are two kinds of BGP peering relationships: BGP peers within the same Autonomous System (AS, which generally means the set of routers within a single administrative domain, though this is a rather loose definition) are called internal BGP (iBGP) peers, and BGP peers between autonomous systems are called external (or exterior) BGP (eBGP) peers. While the two kinds of BGP peering relationships are built the same way, they have different advertisement rules.

The BGP Best Path Decision Process

As BGP is designed to interconnect autonomous systems, the best path algorithm is focused primarily on policy, rather than loop free-ness. In fact, if you examine any standard explanation of the BGP best path process, whether or not a particular path is loop free is not included in the decision process at all. How, then, does BGP determine a particular peer is advertising a loop-free route? Figure 16-13 illustrates.

In Figure 16-13, each router is in a separate AS, so every pair of BGP speakers will form an eBGP peering session. A, which is connected to 2001:db8:3e8:100::/64, advertises this route toward B and C. BGP route advertisements carry a number of *attributes,* one of which is the AS Path (others will be discussed later in describing the best path selection process). Before A advertises 100::/64 to B, it adds its AS number into the AS Path attribute. B receives the route and advertises it to D; before advertising the route to D, it adds AS65001 to the AS Path. The AS Path then, tracing from A through C, looks something like this at every hop:

- As received by B: [AS65000]
- As received by C: [AS65000, AS65001]
- As received by D: [AS65000, AS65001, AS65003]

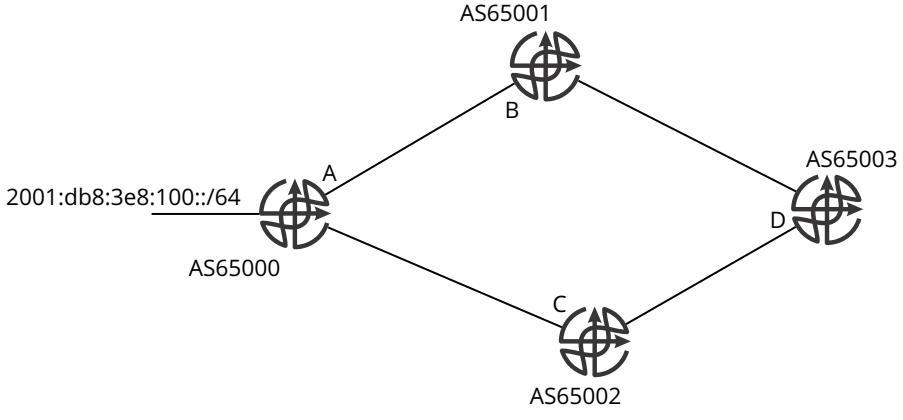


Figure 16-13 *BGP and Loop-Free Paths*

When D received the route from B, it will advertise it back to C (there is no split horizon in BGP). Assume C, in turn, advertises the route back to A for some reason (it would not in this situation, because the path through A would be a better path to the destination, but just to illustrate loop prevention), A will examine the AS Path and discover its local AS is in the AS Path. This is clearly a loop, so A simply ignores the route. Since this route is ignored, it is never placed in the BGP topology table; hence only loop-free routes are compared using the BGP best path process.

The BGP best path process consists of 13 steps in most implementations (the first step is not always implemented, as it is a local decision on the part of the BGP speaker):

1. The route with the highest weight is chosen. Some implementations do not implement a route weight.
2. The route with the highest local preference (LOCAL PREF) is chosen. The local preference represents the exit policy of the local AS—which exit point out of the available exit points would the owner of *this AS* like the BGP speaker prefer.
3. Prefer the locally originated route, which means *on this BGP speaker.* This step is rarely used in the decision process.
4. Prefer the path with the shortest AS Path. This step is intended to prefer the most efficient path through the internetwork, by choosing the path that will pass through the smallest number of autonomous systems to reach the destination. Operators often *prepend* AS Path entries to influence this step in the decision process.
5. Prefer the path with the lowest origin type. Routes that are redistributed from an IGP are preferred over routes with an unknown origin. This step rarely has any impact on the decision process.
6. Prefer the path with the lowest multiexit discriminator (MED). The MED represents the *entrance policy* of the remote AS. As such, the MED is only compared if multiple routes have been received from the same neighboring AS; if the same route is received from two different neighboring autonomous systems, the MED is ignored.
7. Prefer eBGP routes over iBGP routes.
8. Prefer the route with the lowest IGP cost to the next hop. If no local exit policy is set (in the form of the local preference), and the neighboring AS has not set an entrance policy (in the form of the MED), then the path with the closest exit from the local router is chosen as the exit point.
9. Determine if multiple paths should be installed in the routing table (some form of multipath is configured).

10. If comparing two external routes (learned from an eBGP peer), prefer the oldest route, or the route learned first. This rule prevents route churn just because routes are refreshed.
11. Prefer the route learned from the peer with the lowest router ID. This is simply a tiebreaker to prevent churn in the routing table.
12. Prefer the route with the shortest cluster length (see the next section for an explanation of the cluster).
13. Prefer the route learned from the peer with the lowest peering address. This is, again, simply a tie breaker, chosen arbitrarily to prevent ties and cause churn in the routing table, and would normally be used when two BGP peers are connected over two parallel links.

While this seems like a long process, almost every best path decision in BGP comes down to four factors: the local preference, the MED, the AS Path length, and the IGP cost.

Note

If this process isn't complex enough, BGP has been extended to support almost any best path decision scheme an operator can think of. See *BGP Custom Decision Process* for more information.[4] These custom decision capabilities can determine which path is the best path before, or after, any of the decision points described here.

4. Retana and White, "BGP Custom Decision Process."

BGP Advertisement Rules

BGP has two simple rules to determine where to advertise a route:

- Advertise the best path to every destination to every eBGP peer.
- Advertise the best path learned from an eBGP peer to every iBGP peer.

Another way to put these two rules is this: never advertise a route learned from an iBGP to another iBGP peer. Figure 16-14 illustrates.

In Figure 16-14, A and B are eBGP peers, while B and C, and C and D, are iBGP peers. Assume A advertises 2001:db8:3e8:100::/64 to B. Since B received this route

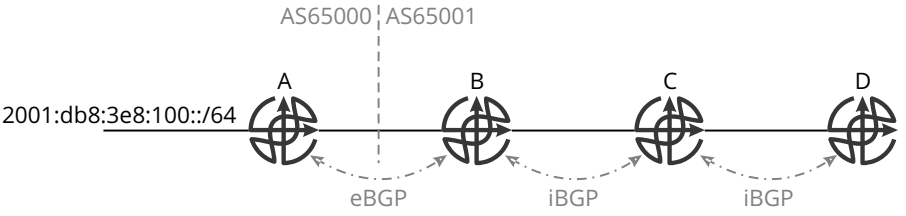


Figure 16-14 *BGP Advertisement Rules*

advertisement from an eBGP peer, it will advertise 100::/64 to C, which is an iBGP peer. C, on learning this route, will not advertise the route to D, however, as C received the route from an iBGP peer, and D is also an iBGP peer. In this illustration, then, D will not learn about 100::/64. This does not seem very useful in the real world; however, the restriction is there for a reason.

Consider how BGP prevents routing loops from forming—by carrying a list of the autonomous systems through which the route has passed in the route advertisement itself. When advertising a route from one iBGP speaker to another, there is *no change in the AS Path*. If iBGP speakers advertised routes learned from iBGP peers to iBGP peers, routing loops can easily be formed. One solution to this problem is simply to build a *multihop* peering relationship between B and D (remember that BGP runs on top of TCP; so long as there is IP connectivity between two BGP speakers, they can build a peering relationship). Assume that B builds a peering relationship with D across C, and neither B nor D builds a peering relationship with C. What will happen when traffic is switched toward 100::/64 by D toward C? What will happen to packets in this flow at C? C will not have a route to 100::/64, so it will drop the traffic. This can be solved in a number of ways—for instance, B and D could tunnel the traffic across C, so C does not need to have reachability to the external destination. BGP could also be configured to redistribute routes into whatever underlying IGP is running (this is a bad idea!—do not do this).

BGP route reflectors were standardized to resolve this problem. Figure 16-15 illustrates the operation of route reflectors.

In Figure 16-15, E is configured as a route reflector; B, C, and D are configured as route reflector clients (specifically, as clients of E). A advertises the 2001:db8:3e8:100::/64 route to B; B advertises this route to E, because it was received from an eBGP peer, and E is an iBGP peer. E adds a new attribute to the route, a *cluster list,* which indicates the path of the update within the AS through the route reflector clusters. E will then advertise the route to each of its clients. Loop prevention, in this case, is handled by the cluster list.

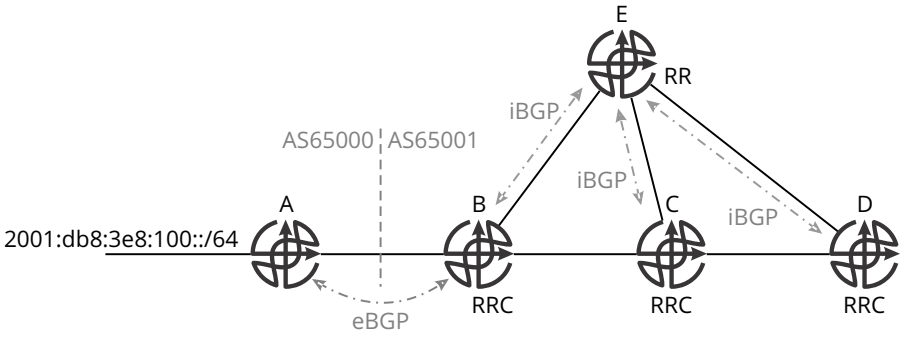


Figure 16-15 *BGP Route Reflectors*

Concluding Thoughts on BGP

While BGP was originally designed to interconnect autonomous systems, its use has spread to data center fabrics, network cores, and carrying information about virtual private networks. The uses to which BGP has been put are, in fact, almost limitless; hence, you will encounter BGP in a number of future chapters. Along the way, BGP has become a very complex protocol; this section barely begins to sketch the operation of the protocol.

BGP can be described as

- A proactive protocol that learns about reachable destinations through configuration, local information, and other protocols
- A path vector protocol that advertises only the best path to each neighbor and does not prevent loops within an autonomous system (unless route reflectors or some additional feature is deployed)
- Selecting loop-free paths by examining the path through which the destination can be reached
- Validating two-way connectivity and MTU through its use of TCP as a transport

Final Thoughts

It is only possible to scratch the surface of distributed control planes in two short chapters. Hopefully, however, these chapters give you a sense of how complex the problem of calculating loop-free paths really is and how many possible solutions to

this problem set there are. So long as you remember the basic classifications, however, you can quickly grasp the basic operation of any routing protocol:

- How does it learn about and advertise information about topology and reachable destinations? Is the protocol reactive or proactive?
- How do devices running the protocol discover other devices running the same protocol? How does it form neighbors?
- How does the protocol detect MTU mismatches?
- How does the protocol distribute routing information reliably through the network?
- How does the protocol marshal data?
- How does the protocol remove topology and reachability information?
- How does the protocol ensure two-way connectivity, both at the neighbor level and when calculating loop-free paths?
- How does the protocol calculate loop-free paths?

You should consider the resources in the "Further Reading" section if you would like to understand each or any of these protocols in greater depth.

Further Reading

Chandra, Ravi, and John Scudder. *Capabilities Advertisement with BGP-4*. Request for Comments 5492. RFC Editor, 2009. https://rfc-editor.org/rfc/rfc5492.txt.

Chen, Enke, Tony J. Bates, and Ravi Chandra. *BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)*. Request for Comments 4456. RFC Editor, 2006. https://rfc-editor.org/rfc/rfc4456.txt.

Chen, Enke, John Scudder, Alvaro Retana, and Daniel Walton. *Advertisement of Multiple Paths in BGP*. Request for Comments 7911. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7911.txt.

Chen, Enke, and Quaizar Vohra. *BGP Support for Four-octet AS Number Space*. Request for Comments 4893. RFC Editor, 2007. https://rfc-editor.org/rfc/rfc4893.txt.

Chunduri, Uma, Wenhu Lu, Albert Tian, and Naiming Shen. *IS-IS Extended Sequence Number TLV*. Request for Comments 7602. RFC Editor, 2015. https://rfc-editor.org/rfc/rfc7602.txt.

Doyle, Jeff, and Jennifer DeHaven Carroll. *Routing TCP/IP, Volume 1*. 2nd edition. Indianapolis, IN: Cisco Press, 2005.

Ferguson, Dennis, Acee Lindem, and John Moy. *OSPF for IPv6*. Request for Comments 5340. RFC Editor, 2008. https://rfc-editor.org/rfc/rfc5340.txt.

Ginsberg, Les, Stephane Litkowski, and Stefano Previdi. *IS-IS Route Preference for Extended IP and IPv6 Reachability*. Request for Comments 7775. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7775.txt.

Heitz, Jakob, Keyur Patel, Job Snijders, Ignas Bagdonas, and Nick Hilliard. "BGP Large Communities." Internet-Draft. Internet Engineering Task Force, January 2017. https://tools.ietf.org/html/draft-ietf-idr-large-community-12.

"Intermediate System to Intermediate System Intra-Domain Routing Information Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service." Standard. Geneva, CH: International Organization for Standardization, 2002. http://standards.iso.org/ittf/PubliclyAvailableStandards/.

Katz, Dave. "OSPF and IS-IS: A Comparative Anatomy." Presented at the NANOG19, Albuquerque, NM, June 12, 2000. https://nanog.org/meetings/abstract?id=1084.

McPherson, Danny R., and Keyur Patel. *Experience with the BGP-4 Protocol*. Request for Comments 4277. RFC Editor, 2006. https://rfc-editor.org/rfc/rfc4277.txt.

Meyer, David, and Keyur Patel. *BGP-4 Protocol Analysis*. Request for Comments 4274. RFC Editor, 2006. https://rfc-editor.org/rfc/rfc4274.txt.

Mirtorabi, Sina, Abhay Roy, Acee Lindem, and Fred Baker. "OSPFv3 LSA Extendibility." Internet-Draft. Internet Engineering Task Force, October 2016. https://tools.ietf.org/html/draft-ietf-ospf-ospfv3-lsa-extend-13.

Moy, John T. *OSPF Version 2*. Request for Comments 2328. RFC Editor, 1998. https://rfc-editor.org/rfc/rfc2328.txt.

Parker, Jeff. *Recommendations for Interoperable Networks Using Intermediate System to Intermediate System (IS-IS)*. Request for Comments 3719. RFC Editor, 2004. https://rfc-editor.org/rfc/rfc3719.txt.

Przygienda, Dr. Antoni B. *Optional Checksums in Intermediate System to Intermediate System (ISIS)*. Request for Comments 3358. RFC Editor, 2002. https://rfc-editor.org/rfc/rfc3358.txt.

Ramachandra, Srihari S., and Yakov Rekhter. *BGP Extended Communities Attribute*. Request for Comments 4360. RFC Editor, 2006. https://rfc-editor.org/rfc/rfc4360.txt.

Raszuk, Robert, Christian Cassar, Bruno Decraene, Stephane Litkowski, Kevin Wang, and Erik Aman. "BGP Optimal Route Reflection (BGP-ORR)." Internet-Draft. Internet Engineering Task Force, January 2017. https://tools.ietf.org/html/draft-ietf-idr-bgp-optimal-route-reflection-13.

Rekhter, Yakov, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. Request for Comments 4271. RFC Editor, 2006. https://rfc-editor.org/rfc/rfc4271.txt.

Retana, Alvaro, and Russ White. "BGP Custom Decision Process." Internet-Draft. Internet Engineering Task Force, February 2017. https://tools.ietf.org/html/draft-ietf-idr-custom-decision-08.

Roy, Abhay, Yi Yang, and Alvaro Retana. *Hiding Transit-Only Networks in OSPF*. Request for Comments 6860. RFC Editor, 2013. https://rfc-editor.org/rfc/rfc6860.txt.

Shand, Mike, Stefano Previdi, Les Ginsberg, and Danny R. McPherson. *Simplified Extension of Link State PDU (LSP) Space for IS-IS*. Request for Comments 5311. RFC Editor, 2009. https://rfc-editor.org/rfc/rfc5311.txt.

Vohra, Quaizar, and Enke Chen. *BGP Support for Four-Octet Autonomous System (AS) Number Space*. Request for Comments 6793. RFC Editor, 2012. https://rfc-editor.org/rfc/rfc6793.txt.

Walton, Daniel, Alvaro Retana, Enke Chen, and John Scudder. *Solutions for BGP Persistent Route Oscillation*. Request for Comments 7964. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7964.txt.

Wang, Lili, Zhaohui (Jeffrey) Zhang, and Nischal Sheth. *OSPF Hybrid Broadcast and Point-to-Multipoint Interface Type*. Request for Comments 6845. RFC Editor, 2013. https://rfc-editor.org/rfc/rfc6845.txt.

White, Russ. *Intermediate System to Intermediate System (IS-IS) Routing Protocol LiveLessons*. Video. LiveLessons. Cisco Press, 2016. http://www.ciscopress.com/store/intermediate-system-to-intermediate-system-is-is-routing-9780134465326?link=text&cmpid=2017_02_02_CP_RussWhiteVideo.

White, Russ, Danny McPherson, and Srihari Sangli. *Practical BGP*. Boston, MA: Addison-Wesley Professional, 2004.

White, Russ, and Alvaro Retana. *IS-IS: Deployment in IP Networks*. 1st edition. Boston, MA: Addison-Wesley, 2003.

Review Questions

1. Why does IS-IS send interface addresses as "neighbors seen" on multiaccess links like Ethernet, and IS identifiers on point-to-point links? What is the reasoning behind the different forms of two-way connectivity checks?
2. IS-IS carries two kinds of metrics—narrow and wide. Describe the mechanism used to transition between these two metric types. Is it effective? How does it compare to the solution adopted by EIGRP? Does it suffer from the same sorts of failure modes as the EIGRP transition mechanism?
3. It is possible that an IS-IS LSP might become longer than the maximum size allowed based on the "size of LSP" field in the LSP header. Describe how RFC5311 solves this problem. Are there any other ways you can think of to solve this same problem?
4. IS-IS and OSPF rely on sequence numbers to indicate which piece of information being flooded through the network is the most recent. Read RFC7602 and RFC5310. Describe the problem caused by this reliance and how IS-IS resolved this problem. Are there problems with the solution standardized in RFC7602?
5. Compare OSPF and IS-IS data marshalling using the complexity model described earlier in the book (state/optimization/surface). Where do you think these two protocols have traded off state for optimization? Do the multiple LSA types and reliance on IP fragmentation represent an interaction surface that increases the complexity of OSPF?
6. Describe the security issue created by the link state age-out and reflood behavior of both OSPF and IS-IS. Find and describe the solution proposed in the IETF.
7. Consider DIS/DR election on a point-to-point link that is considered a broadcast medium (such as a point-to-point Ethernet link). Will electing a DR/DIS and creating a pseudonode reduce overall complexity or increase it? What features have been implemented in commercial implementations of OSPF and IS-IS to mitigate the result?

Chapter 17

Policy in the Control Plane

Learning Objectives

After reading this chapter, you should understand:

- How to define control plane policy
- Hot and cold potato routing as examples of control plane policy
- How to create virtual topologies as a policy implementation mechanism
- Basic traffic engineering concepts, such as flow pinning

The last several chapters have considered the many variations on finding a set of loop-free paths through a network. In the explanation of the Border Gateway Protocol (BGP), however, you might have noticed the emphasis on various policies, rather than strictly finding loop-free paths. This chapter, then, will continue the emphasis on policy begun in the preceding chapter.

The first question to answer is: what is policy? Unfortunately, there is no simple answer. The best way to answer this question is through examples; these will be considered in the following section. The second section of this chapter will draw lessons from these examples, and then consider problems and solutions in the control plane policy space.

> **Note**
>
> Control policy is often difficult to separate conceptually from data plane policy, such as packet filtering and Quality of Service (QoS). In fact, the two overlap in many places, such as the control plane carrying QoS markings that are then applied to packets, or drawing packets into a null interface, effectively dropping them. These sorts of corner cases are avoided here for clarity.

Control Plane Policy Use Cases

Often the best way to understand a concept is through examples. This section examines three examples of policy being used in the control plane to fulfill business requirements: determining where traffic should exit a provider network, optimizing application performance by pinning elephant flows, and increasing or providing security through network segmentation. The next section draws a set of lessons from these examples.

Routing and Potatoes

Service providers normally live within a world of tight budgets, application requirements, and business drivers. The mixture of these three can make for some strange situations when routing between providers of various kinds. Specifically, cold potato routing is designed to keep traffic inside the provider's network for as long as possible, while hot potato routing is designed to push traffic to the closest exit point possible. The result of mixing these two is sometimes called (tongue in cheek) mashed potato routing. Figure 17-1 is used to explain.

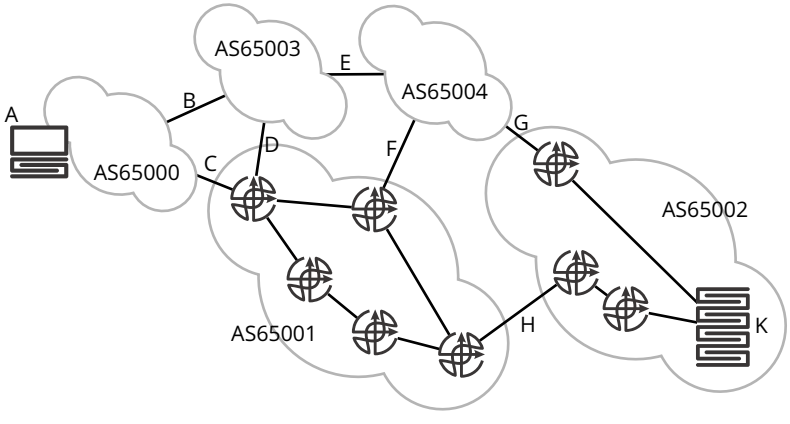


Figure 17-1 *Routing and Potatoes*

Assume AS65000 is an edge provider, or perhaps an "enterprise" network connected to two upstream providers, AS65001 and AS65003. AS65001, AS65003, and AS65004 are transit providers, and AS65002 is a content provider. Some of the policies and business drivers for those policies in this collection of networks might be

- AS65001 wants to draw as much traffic from AS65000, across link C, as possible. The more this link is filled up, the more likely AS65000 is to purchase an upgraded link. There is actually little AS65001 can do to attract traffic, of course, other than perhaps trying to convince the administrators in AS65000 to ship more traffic their direction, or trying to improve the performance of the link from the perspective of some sort of traffic engineering system AS65000 might have configured on their end of the link.
- AS65001 wants to forward any traffic entering at link C to the closest exit with a route to the destination. For instance, if AS65003 is advertising a route to K, on the right side of the diagram, AS65001 will prefer the exit through link D, even though it might not be the shortest overall path to the destination. Normally, AS65001 would implement this sort of policy using BGP's local preference, or by relying on the underlying Interior Gateway Protocol (IGP) metric to draw traffic to the closest exit point out of the Autonomous System (AS). This is called hot potato routing. Why does AS65001 want to push the traffic to the nearest exit point with a route to the destination? Because carrying the traffic along the path to link H, for instance, consumes network resources. AS65001 is being paid based on the usage of link C, rather than for actually carrying the traffic as close as possible to the destination. Hence, AS65001 will draw as much traffic as possible off its paying customers but then push the traffic to the nearest exit point.
- AS65002, on the other hand, generally wants to control its user's experience as tightly as possible, because it is selling a service. If the network between the service and the user has poor quality, then the service itself is perceived to be poor quality, and the content provider's overall business will suffer. The longer the traffic stays in AS65002's network, the more control the content provider has over the Quality of Service delivery. Keeping the customer's traffic inside the network is essentially bringing the customer's eyeballs closer to the service itself. This is a form of cold potato routing. Instead of tossing the traffic out of your network as quickly as possible (as you would with a hot potato), you hold on to the traffic as long as possible (like a cold potato). In this case, AS65002 is going to perceive the closest exit point to the customer as being through AS65001 at link H because the path through H has the shortest AS path. Although the internal path is longer, AS65002 will choose the path through link H to control the traffic as long as possible.

- When traffic is received at link H, AS65001 needs to decide whether to send the traffic to some nearby exit point, say link F, or to carry the traffic along the entire network so it exits at link C. In this case, AS65001 will almost always decide to carry the traffic along its entire network. Again, the primary selling point AS65001 has toward AS65000 is to increase the average utilization along link C. To do this, AS65001 needs traffic to send toward AS65000; the only way to get this traffic is to carry traffic from every entry point into the network to the connection to the customer, if the destination is a customer. This is again cold potato routing.

Forwarding traffic over the fewest number of links (and therefore through the fewest number of network devices) would consume the smallest amount of resources, but cold and hot potato routing both choose some longer-length path in order to satisfy a policy constraint. This trades the efficient use of resources for the efficient operation of a protocol or service in order to increase revenue. Other policies may be applicable to routing systems, as well, such as choosing the path with the highest bandwidth, or the path that takes traffic to the geographic exit point closest to the user. Whatever the policy, it will generally represent a tradeoff between one kind of optimization over some other kind of optimization, and require additional state of some sort to implement.

Resource Segmentation

Many times networks are logically divided to control access to specific resources. The network shown in Figure 17-2 will be used to illustrate.

Figure 17-2 shows three different networks:

- Network A shows the base (routed) topology.
- Network B shows one set of devices and links that must be connected to one another.
- Network C shows a second set of devices and links that must be connected to one another.

In Figure 17-2, host B must only be able to connect to server L, and host A must only be able to connect to H. It is simple enough to provide this kind of segmentation through simple packet filters configured at G and K, of course, but further requirements may rule out using simple packet filters. For instance:

- There may be a requirement for traffic passing between A and H to use the path [C,E,F,K,G]; this mixes a traffic engineering requirement with a service access requirement.

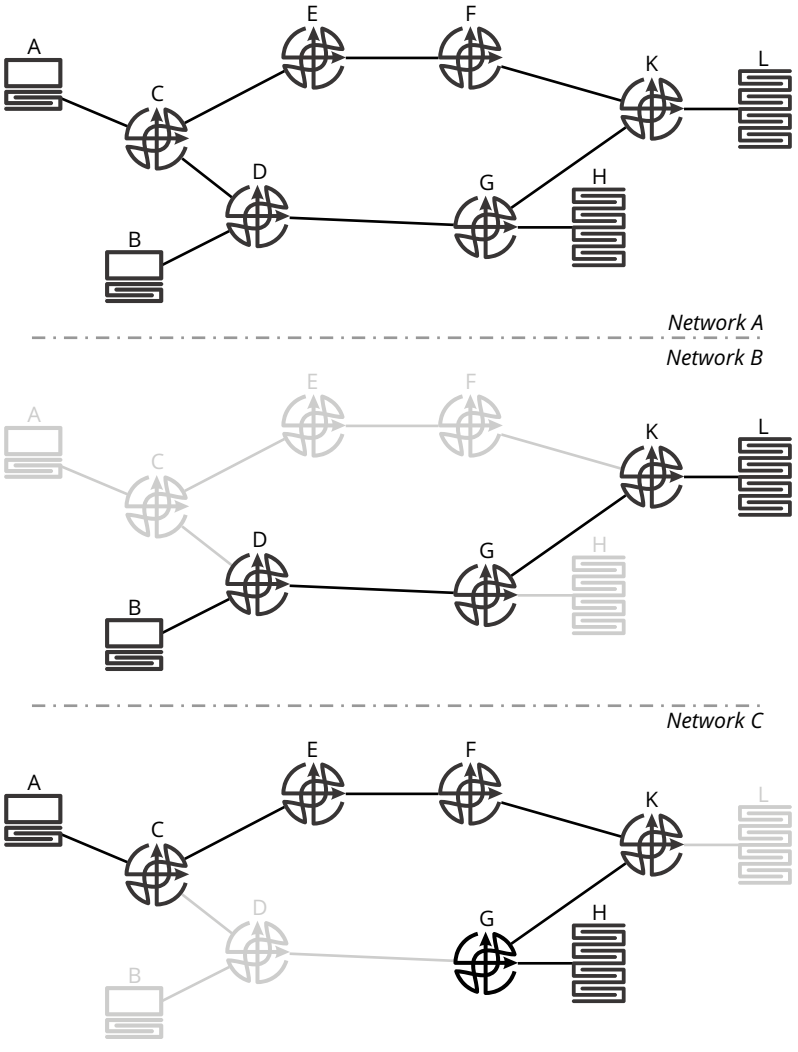


Figure 17-2 *Overlays as a Form of Segmentation*

- There may be a requirement for servers H and L to not even be able to *see* routes and other information from the other topology. This might be very difficult if the two servers are participating in the routed control plane, as might be the case if they are hosting many virtual machines (VMs), each of which needs to advertise its own IP address into the control plane.

When you reach this level of requirement, a common solution is to create an overlay network, often using tunnels to do the heavy lifting of separating the network

into several virtual topologies. In Figure 17-2, these requirements are met with an overlay. In network B, a tunnel would be built starting at the inbound interface of D facing host B (the tunnel headend). This tunnel would be carried across G and K, finally terminating on the interface on K that connects to L (the tunnel tailend). To draw traffic from B to L, there must be some routed control plane to pull traffic into the tunnel headend so it is routed across the tunnel toward L. In network C, a tunnel would be built starting at the inbound interface of C, facing A. The tunnel is carried across C, E, F, K, and G, and terminates at the outbound interface at G facing H. Again, there must be some control plane to draw data across this tunnel, so traffic sourced from A is pulled into the tunnel at C and is presented to G as a "raw IP packet" (without the tunnel headers) so that G can switch the packet to H.

The routing information that draws traffic through these two tunnels may actually be carried in a separate control plane. In this case, the underlay control plane will provide reachability to the tunnel endpoints, while the overlay control plane will draw traffic through the tunnel. This separation of control planes allows the different topologies, the underlay and the overlay, to be completely separated; reachability and topology information is not shared between these two control planes.

The same is true of the traffic being drawn through the network; the two flows are separated by being tunneled. Not only is the traffic being separated by tunneling, but the path of the flow is also being engineered through the network. Tunneling, and the fuller concept of an overlay, is useful in meeting a lot of different policy requirements; this is why overlays are so widely used in network engineering.

Flow Pinning for Application Optimization

Elephant flows and mouse flows are two classes of flows that engineers often encounter. An elephant flow is typically a large, persistent data flow. Any flow taking up more than around 20% of the available bandwidth of a single link and persisting for more than two or three minutes might, for instance, be classified as an elephant flow. Mouse flows, on the other hand, are much lower bandwidth, say less than 1% of the available bandwidth on any link, and tend to last for very short periods of time. Most flows of traffic can be divided into elephant and mouse flows. *What should be done about elephant and mouse flows?*

One solution is to interleave the packets from each flow, which allows each flow fair access to the available bandwidth. While Quality of Service (QoS) is one solution, another solution is to pin particular traffic flows to particular paths, or path pinning. Figure 17-3 is used to explain further.

In Figure 17-3, A begins a flow that will last several hours, and consumes 20% of the available bandwidth on a link (assume all links are the same bandwidth, 100Mbps), and terminates at H. At about the same moment, B sends a series of

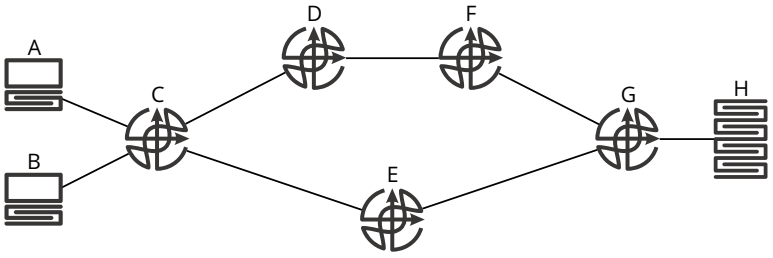


Figure 17-3 *Pinning an Elephant Flow in a Network*

short-term small flows terminating at H. Given just one path will be chosen as the best between C and G, both flows will follow the same path, say along the path [C,E,G]. Mixing the two flows in this way can cause both to suffer from a performance perspective. To understand the problem, it is best to consider the rate at which packets can be serialized onto the wire:

- **64-byte packet onto a 100Mbps link:** .05ms
- **1,500-byte packet onto a 100Mbps link:** .12ms
- **9,000-byte packet onto a 100Mbps link:** .72ms

Assume the entire network is capable of 9,000-byte packet sizes (the Maximum Transmission Unit, or MTU is 9,000 bytes end to end), and the elephant flow is actually shipping 9,000-byte packets. For the mouse flow, assume the packet size is 64-byte packets (at least in one direction). If a single mouse flow packet is trapped behind a single elephant flow packet, the mouse flow packet will be held for .72ms before it can be serialized onto the physical interface. If there is always one packet from each flow alternating, there can be some significant performance reduction, but both applications would likely still work well enough.

But what happens if the interleaving between the two flows is less than optimal? For instance, what if there is a series something like the sequence of packets shown in Figure 17-4?

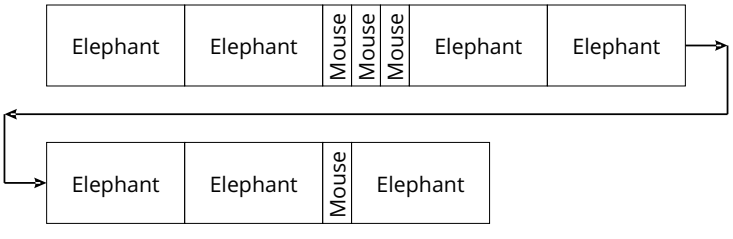


Figure 17-4 *A Problematic Mixture of Packets from Elephant and Mouse Flows*

The difference between the shortest and longest spacing between mouse flow packets is .05ms and .288ms. The difference between the shortest and longest spacing between elephant flow packets is .05ms and .15ms. These variations might seem to be minimal, but even minimal variations show up as jitter end to end. This kind of jitter, particularly on a larger scale, is problematic for flow control and error correction. In this case, even though the elephant flow is overwhelmingly larger than the mouse flow, both are still negatively impacted. This same sort of problem is common in data center fabrics, as well. Figure 17-5 illustrates.

In Figure 17-5, A has two flows: an elephant flow to G and a set of mouse flows to H. While there is plenty of bandwidth to support both flows across the fabric, if both flows happen to be hashed onto the [B,C] link by the equal cost multipath (ECMP) algorithm, the interaction of the two flows can cause jitter for the supported applications, reducing performance.

Pinning the elephant flow to the [B,C] link and keeping other traffic off this link so that the traffic to F follows the [A,B,D,F,H] path can resolve these performance problems. Elephant flows tend to be more common in the data center environment.

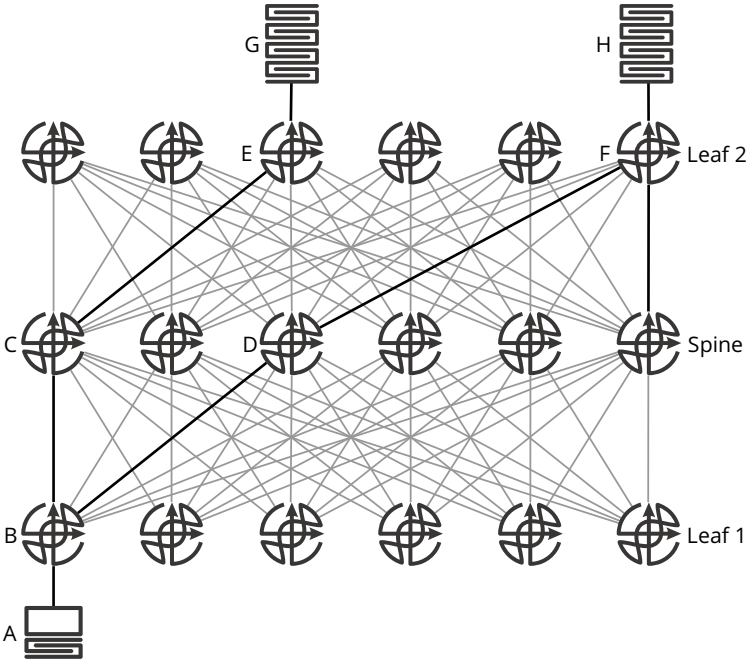


Figure 17-5 *Traffic Engineering in a Data Center Fabric*

How can the problems caused by mixing two different kinds of traffic on a single link be prevented? One obvious way in a two-connected network, such as the ones illustrated in Figure 17-3 and Figure 17-5, is to somehow pin one of the flows onto one path and remove the other flows from the link the elephant flow is pinned to. For instance:

- In Figure 17-3, if one of the two flows is pinned to the longer [C,D,F,G] link, while leaving the other flow on the shorter [C,E,G] link.
- In Figure 17-5, if the elephant flow is pinned to one path, say [B,C,E], and the mouse flows can be somehow directed to avoid [B,C,E] so they use some other path, say [B,D,F].

Not only must the elephant flow be pinned to a particular path, but the mouse flows must be prevented from flowing along the path the elephant flow has been pinned to. Sometimes just allowing one flow to follow the shortest loop-free path while pinning the other flow to some longer (but still loop-free) path will be sufficient. This does not, however, often work in data center fabrics and other networks where the available paths across which the traffic must be engineered are equal cost. Pinning the elephant flow to one path is not useful if the mouse flows can still be placed on the same path as the elephant flow through the operation of a router randomly choosing among a set of equal cost paths.

To separate the two flows in the example in Figure 17-3, there must be some way to differentiate the flows during the switching process. There are a number of ways to differentiate the flows, including

- **The destination address.** In Figure 17-3, both flows are destined to H, so the destination address would not be useful for differentiating between the two flows. This is not always the case.
- **The source address.** In Figure 17-3, the source of the first flow is A, and the source of the second is B. The source address could be used in this situation to differentiate between the flows at C. However, because hosts normally send packets (or open sessions) with a number of servers in a network, the source and destination addresses are normally used together, rather than just the source address.
- **The port number.** Port numbers and protocol numbers are normally associated with a single application on a host or a server. The source and destination port numbers can often be combined to pick out traffic from a specific flow, rather than one or the other.

These differentiators can be combined into a set of markers uniquely identifying every flow in a network running the Internet Protocol (IP) suite, the five tuple. The five tuple consists of

- The source IP address
- The destination IP address
- The protocol number
- The source port number
- The destination port number

Because of the way each protocol operates, either the source or destination port will be an ephemeral port, or a port assigned to this specific host. There are alternative ways for traffic to be identified other than examining the various fields that identify a flow. For instance, host A could be configured to mark all the packets in an elephant flow with a particular number in an IPv6 extension header, or with specific QoS bits. In this case, C could simply check for the specified information in the IP header, determining which link traffic should be switched to based on the contents of the correct field.

Given the traffic flows can be differentiated from one another, what techniques are available to draw (or push) the traffic in each flow along a different link? Several methods are often used.

A statically configured packet filter, sometimes called a policy route, or a filter-based forwarding rule, can be configured at C and G in Figure 17-3. This rule would contain logic that *matches* on the fields differentiating the flows and *sets* the correct next hop. This solution (obviously) requires manual configuration; this configuration must be managed over time, including adjusting where the packet filter is applied, what traffic is matched by the filter, and where the matching traffic is forwarded. This kind of filter can seem simple when first deployed, but can become difficult to maintain over time. For instance, in Figure 17-3, examining the traffic pattern at F would give you no clues about why one of the flows was traveling over the longer path. You would need to trace the traffic back to C to discover why this traffic is passing along this particular path. Because of the additional management and maintenance issues, automated solutions are often preferred.

Metric manipulation can sometimes be used to draw each flow along a different path. In Figure 17-3, if H is sending traffic to A and B, the path through [C,D,F,G] could be manipulated to have a lower cost toward A, while the path through [C,E,G] could be manipulated to have a lower cost toward B. One problem with this solution

is obvious from the example. Consider the situation from the perspective of A and B; these two hosts are, in fact, sending both the elephant and the mouse flows to the same destination, so there is no way to use metrics to draw traffic from these two hosts along the two available paths. A second problem with this solution is similar to the one described previously with a packet filter. If you examine the routing table at F, there would be no obvious reason why the metrics for the two different destinations are different. Again, you would need to trace back the difference in metrics to some configuration on either C or G to discover why two destinations that appear to be on either end of the same set of links have two different metrics.

The packets in one flow may be tunneled through the network, or drawn into a virtual overlay topology. A tunnel would more likely be used to solve an elephant flow problem than a virtual overlay topology in most networks; a tunnel can be directed along a single path, but a virtual overlay topology is likely to have many paths that traffic could take, so the flow isn't pinned to a specific path.

Defining Control Plane Policy

The three use cases (or examples) given in the previous section are examples of traffic engineering, which simply means manipulating the control plane to specify the path that specific flows take through the network. The first point to observe in these examples is for every case, some traffic is removed from the shortest—and hence presumably the most efficient—path through the network and somehow made to follow a longer loop-free path. This common element is helpful in defining policy:

> **Control plane policy is anything that causes traffic to flow over a path longer than the shortest path in order to provide some form of optimization.**

One specific term needs to be considered further in this definition: what, precisely, does optimization mean? While there are many possible optimizations, they can be broken down into four broad categories:

- **Network utilization:** Operators sometimes try to optimize the utilization of a single link, such as a highly utilized path between two data centers. Network utilization can also be optimized in a more global way, such as the average utilization of every link in the network, or perhaps the available switching capacity across devices and links versus the capacity required to support specific business goals and/or applications.

- **Application support:** Applications and data are often the "real" heart of a business. No matter what kind of work a business claims to do, it actually works with information to connect buyers to sellers in some way, a process requiring data and data processing (or data analytics). A network can be optimized to support specific applications representing the primary business drivers by ensuring this set of applications always has reachability or reduced jitter and delay.
- **Business advantage:** The network can be optimized to increase the financial advantage of the business in some way. Specifically, reducing the cost the business pays to other companies to operate or increasing revenue by increasing user engagement or opening up new markets by connecting to new geographical locations might be ways in which the network can create opportunities to improve the business.
- **Cost:** How can the business build and maintain a less expensive network? This is not just a common question; it is often *the only* question the business that relies on the network cares about.

Any particular network will rarely be optimized for a single class among these four. Most networks will be optimized for all four of these in various parts of their topology or even at various times. Optimizations will often cross over these categories; for instance, improving support for a specific application may increase business advantage by allowing information to be applied to a specific area of the business more quickly, while also saving costs by reducing the application's downtime.

Control Plane Policy and Complexity

Control plane policy is not exempt from the "choose two of three"—state, surface, and optimization—complexity tradeoff described in Chapter 1, "Fundamental Concepts." What are the tradeoffs involved in the examples given in the first part of this chapter? Each use case will be covered in the sections that follow.

Routing and Potatoes

In the first use case, various policy mechanisms are used to manage where traffic exits an AS and, to some degree, where traffic enters an AS. It is easy to overlook the complexity impacts of the attributes carried in BGP, as they are actually a *part* of BGP. How can simply using something built into the protocol have an impact from a complexity perspective?

First, the protocol itself must be more complex in order to support the attributes being carried, and implementations build, test, and maintain the code required to

process these attributes. This might seem very minor, but consider the case of BGP update packing. Figure 17-6 illustrates two sets of BGP packets.

The upper pair of packets in Figure 17-6, labeled A, is two different destinations carried in BGP format; there is a set of attributes (in this example only one attribute is shown, the LOCAL_PREF) and a reachable prefix. While the reachable destination is different in the two packets, the LOCAL_PREF, or rather the set of attributes, is the same. Hence, when actually advertising these to destinations, BGP can **pack** the two prefixes into a single update. To do this, the two prefixes are simply combined into a single update with the single set of attributes.

The lower pair of packets in Figure 17-6, labeled B, is two different destinations carried in the BGP format. In this case, the reachable destinations and the attributes are different, so they cannot be combined into a single BGP update.

Packing updates, as shown with packet A, represents a major space saving when transmitting reachability information through a network in BGP. While the savings will vary between networks, it is not surprising for efficient packing to reduce initial convergence time and the number of packets sent by somewhere around 80%.

The goal of adding the policy information was to improve the utilization of the network, or rather to move traffic to maximize revenue and minimize expenses. The decision to add per route essentially trades state for optimization. More state is injected into the control plane, both in terms of terms of the actual amount of state and the efficiency of carrying the state across the network, so the state versus optimization tradeoff holds true.

What about interaction surfaces? There are two places where this solution interacts with other systems in the network. First, the policy marker needs to be set on the correct routes, and associated with some action someplace else in the network. Largely, these settings are going to be made by a pair of human hands adding the right configuration commands to instruct BGP to set and react to these route markers. The interaction surface between people and the network is often the most

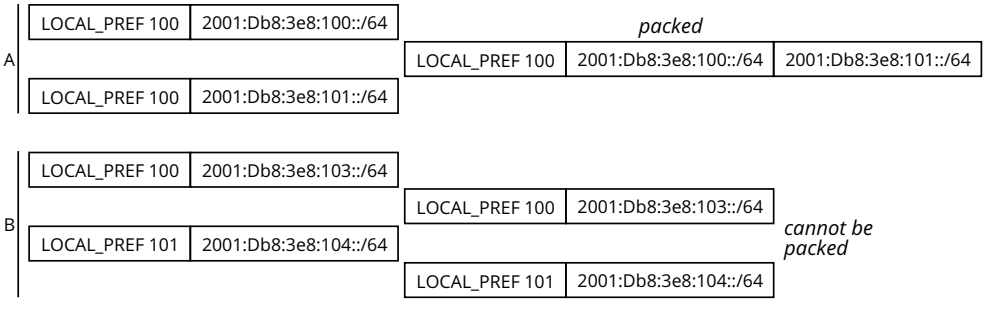


Figure 17-6 *Complexity and BGP Policies*

difficult surface to manage. In the case of the multiple exit, or multiexit, discriminator (MED) and communities sent outside the AS, there is an interaction surface with the neighboring autonomous systems.

Routing policy, in this case, definitely fits within the State/Optimization/Surface (SOS) model; increasing network utilization requires an increase in state and surfaces.

Resource Segmentation

In resource segmentation, it would seem that by splitting the reachability and topology state out of the underlay topology, the amount of state in the underlay topology has been reduced, and hence the overall complexity has been reduced. At the same time, the network appears to be more closely aligned with the business requirements, so it looks like optimization has increased while state has decreased. This seems to go against the complexity model; if the network is becoming more optimized, state should be increasing.

Welcome to the world of abstraction; this is one of those cases where you must consider things more closely to really understand the impact on complexity. Remember: *if you have not found the tradeoffs, you have not looked hard enough*. What is happening here is that there are now three different control planes with less information about the overall topology; the total state in the system has increased, as there are three pieces of state about a subset of the links (one for the underlying physical topology and one for each overlay virtual topology). There is definitely a larger amount of state; it is just more "spread around."

Further, there are now three control planes running in the network; there is definitely an interaction surface to consider between the protocols, even if the protocols do not carry the same reachability information over the same links. Figure 17-7 illustrates.

Figure 17-7 represents the same network topology shown in Figure 17-2, with the virtual overlay topologies collapsed into a single diagram. The physical topology is

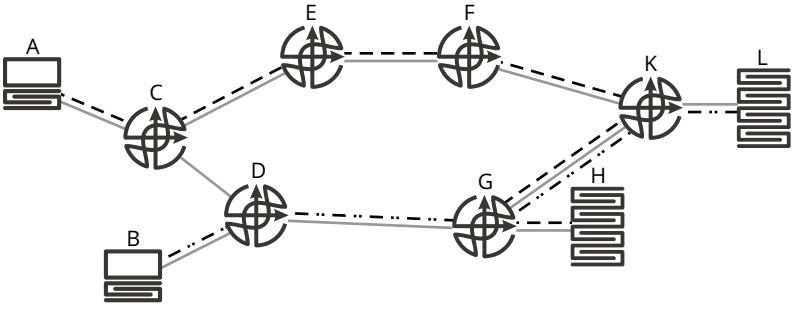


Figure 17-7 *Interaction Surfaces between Multiple Overlay Control Planes*

represented by the solid gray lines; the first overlay is represented by the black dashed line; and the second overlay is represented by the black lines with intermixed dots and dashes. When the physical and overlay topologies are illustrated in this way, it is easy to see the single [G,K] link is shared across all three topologies. If the [G,K] link fails, both of the overlay topologies will also fail; this is called fate sharing. The set of links shared between more than one topology is called the Shared Risk Link Group (SRLG).

These three control planes may not initially appear to interact with one another. They do, however, interact at the [G,K] link. Even if there is no actual link failure, any failure in the underlay control plane will cause both of the virtual topologies to fail to be able to forward traffic between their respective sources and destinations. There is an interaction between the three control planes even though they do not redistribute information between themselves.

What is perhaps worse, however, is that in a two-connected network such as the one shown in Figure 17-7, there should always be two paths between any two points in the network. A single link failure should not cause H to become unreachable from A. Because the virtual topology [C,E,F,K,G] is not two connected, however, the network has been converted to a design where a single link failure at the physical layer can cause both virtual topologies to become disconnected.

This sort of shared fate interaction surface is often easy to miss when designing a network with overlays. The abstraction removes details, making it easier to "see" each topology separately, and reducing the state contained in each control plane, but it also hides failure risks and modes that did not exist before virtualization was deployed.

Often the only way to solve this type of problem is to add state back into the three control planes. For instance, in the network in Figure 17-7, there are a number of ways state could be added back into the network to provide alternate paths in the case of the [G,K] link failure. For instance:

- Some outside process could calculate the topologies and then reach across the layers to find SRLGs, or fate sharing points in the network. In this case, yet another control needs to "ride on top" to at least alert the designer about the SRLGs, so the network design can be modified to work around them. This solution adds (in effect) a fourth control plane that must interact with the other three, including any state carried in the fourth control plane.
- The two virtual topologies could be configured to overlay the entire physical topology, and some form of metric weights be placed on the link costs for each overlay topology so traffic passes along the correct path. This adds the state of carrying the entire control plane back to both topologies and potentially adds more points at which the multiple control planes will interact. Further, the overlay link metrics must be computed, configured, and managed.

- Secondary virtual overlays could be designed and deployed on the network so each topology has a prebuilt backup topology. Multiprotocol Label Switching (MPLS) Traffic Engineering (TE) Fast Reroute (FRR) provides this type of solution. To deploy this kind of solution, additional state for the backup path and switching state at the tunnel headend to quickly switch to the backup path must be added; the additional potential interaction surfaces between the operators and the network, the various control planes now running in the network, and even the various switching paths available at each device, all add complexity back into the network.

There is, in the end, no such thing as a free lunch. Network segmentation is often an effective way to provide separation between customers and workloads—it is often the only way to go, given application and security requirements—but there will always be added complexity someplace in such a design.

Flow Pinning for Applications

What are the gains, from a complexity perspective, in the flow pinning example? The primary point of flow pinning is, of course, to optimize the performance of both the elephant and mouse flow applications. The network may also operate more efficiently, at least from a switching perspective, and QoS settings may well be simpler with the two kinds of flows separated. So there is an increase in optimization, and potentially a decrease in state and interaction surfaces (due to the simpler QoS configurations and processing).

To get these improvements, there must be a corresponding increase in complexity someplace else. In this case, the increase in complexity is in control plane state. The elephant flow must somehow be pinned to a specific link, and the mouse flows must somehow be removed from the link to which the elephant flow has been pinned. There must also be some "backup plan" in case the path to which the elephant flow is pinned fails.

Final Thoughts on Control Plane Policy

Control plane policy is often hiding in plain sight in the form of segmentation, flow pinning, traffic engineering, and other forms. Generally, it will take the form of directing traffic away from the shortest path, and onto a path that might appear less than optimal from a lowest metric or hop count perspective, but is more optimal in some other way, such as application support. In fact, this is as good of a definition of control plane policy as any other:

> **Control plane policy** is any modification to the path that packets take through a network off the shortest path in order to implement some specific business or application requirement.

There is one more form of control plane policy not considered here: the aggregation and summarization of control plane information in order to reduce state and divide (or create) failure domains.

Using the control plane to implement policy presents network engineers with a set of tradeoffs. Distributed control planes, as considered in the previous chapters in this book, often become very complex when they are tasked with discovering topology, providing reachability information, and carrying policy. The next chapter explores some alternative ways to solve these problems by centralizing all or part of the functions of the control plane.

Further Reading

These sources consider a number of other interesting and useful control plane policies not discussed in this chapter and provide more information on the policies that were discussed.

Agarwal, Sharad, A. Nucci, and Supratik Bhattacharyya. "Measuring the Shared Fate of IGP Engineering and Interdomain Traffic." In *13th IEEE International Conference on Network Protocols (ICNP'05)*, 10, 2005. doi:10.1109/ICNP.2005.22.

Casado, Martin, and Justin Pettit. "Of Mice and Elephants." *Network Heresy*, November 1, 2013. https://networkheresy.com/2013/11/01/of-mice-and-elephants/.

Das, V. V. "Honeypot Scheme for Distributed Denial-of-Service," 497–501, 2009. doi:10.1109/ICACC.2009.146.

Hinrichs, Tim, and Scott Lowe. "On Policy in the Data Center: The Policy Problem." *Network Heresy*, April 22, 2014. https://networkheresy.com/2014/04/22/on-policy-in-the-data-center-the-policy-problem/.

Justin Pettit, Kanna Rajagopal, and J. R. Rivers. "Elephant Detection in the vSwitch with Performance Handling in the Underlay." *Network Heresy*, May 16, 2014. https://networkheresy.com/2014/05/16/elephant-detection-in-the-vswitch-with-performance-handling-in-the-underlay/.

Karp, Brad Nelson. "Geographic Routing for Wireless Networks." Harvard University, 2000. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.5738&rep=rep1&type=pdf.

Kaur, Harminder, Harsukhpreet Singh, and Anurag Sharma. "Geographic Routing Protocol: A Review." *International Journal of Grid and Distributed Computing* 9, no. 2 (2016): 254.

McPherson, Danny R., and Keyur Patel. *Experience with the BGP-4 Protocol*. Request for Comments 4277. RFC Editor, 2006. https://rfc-editor.org/rfc/rfc4277.txt.

Psounis, K., A. Ghosh, B. Prabhakar, and G. Wang. "SIFT: A Simple Algorithm for Tracking Elephant Flows, and Taking Advantage of Power Laws." In *Proceedings of the 43rd Allerton Conference on Communication, Control and Computing*, 2005. https://web.stanford.edu/~balaji/papers/05sifta.pdf.

Teixeira, Renata. "Hot Potatoes Heat Up BGP Routing." Presented at the RIPE, Amsterdam, October 2005. https://meetings.ripe.net/ripe-51/presentations/pdf/ripe51-hot-potatoes.pdf.

Weiler, Nathalie. "Honeypots for Distributed Denial of Service Attacks." IEEE, 2002. http://www.csl.mtu.edu/cs6461/www/Reading/Weiler02.pdf.

White, Russ. "Elephant Flows, Fabrics, and I2RS." *Rule 11 Reader*, October 3, 2016. http://rule11.tech/i2rs-elephant-flows/.

Review Questions

1. The chapter mentions four broad classes of optimizations that operators may optimize a network for. Think through the use case examples given in the chapter, and explain which of these four classes of optimization each one could fit into and why. Find three other use cases (need to find sources) and explain which of the four classes of optimization each of them could fit into and why. Are there use cases that do not fit into one of these four broad categories? What category would you add to cover all the use cases?

2. Besides hot and cold potato routing, there is also "mashed potato routing." What is the definition of mashed potato routing, and what is it used for?

3. Elephant and mouse flows are described in the chapter as being large, persistent flows, and short-duration small-volume flows. Name three applications that would generate each kind of flow.

4. In Figure 17-3, packet classification must be configured at both C and G to support differentiating traffic flowing between A and H. Why would these need to be configured in both places? Would the filters be the same? If not, how would they be different?

Chapter 18

Centralized Control Planes

Learning Objectives

After reading this chapter, you should understand:

- An overview of the interfaces between components in a network device used in building the information needed to forward packets
- The four different control plane models
- Using BGP as a southbound interface for a centralized control plane
- Using fibbing to modify paths in a link state protocol
- The Interface to the Routing System as a southbound interface
- The Path Control Element Protocol
- OpenFlow's origins and operation
- ACP Theorem and the subsidiarity principle

As much as it might seem otherwise, the information technology field is strongly driven by egos and fashion. What was "in" last year will be "out" this year, often with very little reason other than "this is new, and that is old." Network engineering is no different in this regard. For instance, network designs have swung between an "ideal" of decentralized control planes to centralized control planes a number of times over their history. Which is truly better?

The best way to cut through these pendulum swings, and their attendant hype factor, is to be able to understand the underlying problems, the underlying solutions, and the tradeoffs between the various solutions (as well as whether the problem at hand even needs to be solved at all—a point far too many designers and architects tend to forget). Toward this end, this chapter will begin by explaining a taxonomy of centralized control planes, developed on the model of a forwarding device.

With this model in hand, several specific examples will be surveyed. Each of these systems will be placed into the framework of the problems a control plane needs to solve, and the tradeoffs against distributed control planes, examples of which were considered in the preceding two chapters, will be examined. These solutions all fit into the roughly defined categories of a Software-Defined Network (SDN) or Programmable Network (PN).

Considering the Definition of Software Defined

SDN is often presented as an either/or proposition: either you build a network using distributed protocols, *or* you build a network with a centralized control plane. The nebulous nature of the term *SDN* contributes to this way of seeing *software defined*. Specifically, how is an implementation of Open Shortest Path First (OSPF), for instance, not *software defined*? The general idea seems to be this: in hardware-based networks the software is *embedded* in appliances; the software and hardware are purchased, configured, and managed as one "thing." In software defined, the software is separate from the hardware. Hence, in the SDN model, the software is seen as separate from the appliance; in the distributed model, the software has traditionally been seen as part of (or embedded in) the hardware. This brings us to a second false either/or divide. In reality, *every* distributed control plane is implemented in software, and hence can *always* be separated from the hardware.

Given software can always be separated from hardware, how can SDN be differentiated from the "traditional" model? The primary idea revolves around separating some part of the functionality of the control plane from the individual forwarding devices, or rather, pulling some part of the functionality of the control plane into a "centralized" control plane. It is important to note the term *centralized control plane*, as it is used here, does not mean a single "god box" that controls the network. Rather, it simply means a control plane that, in some way, does not run entirely on the network devices.

A Taxonomy of Interfaces

The SDN and PN worlds, in many ways, have their own terminology; the most important are the *southbound interface* and the *northbound interface*:

- The *southbound interface* is the interface between the controller and the network devices.
- The *northbound interface* is the interface between the controller and applications (or business logic).

Within the realm of southbound interfaces, there are a number of different interaction points, or ways in which the controller interacts with the forwarding devices.

Figure 18-1 shows four different control plane models:

- In the **distributed** model, the control plane software runs *primarily* on forwarding devices. This does not mean the control plane software is *embedded* in the forwarding device. In an appliance model, the software is treated as an embedded part of the appliance itself. In a disaggregated model, however, the software runs primarily on the forwarding device, but the software is clearly delineated from the forwarding hardware.
- In the **augmented** model, the control plane software runs *primarily* on forwarding devices. Like the distributed model, the control plane is not necessarily embedded in the forwarding device. In the augmented model, the local control plane processes interact with the routing table (Routing Information Base, or RIB). Off-box processes interact with the distributed control plane to influence the set of loop-free paths installed in the RIB.

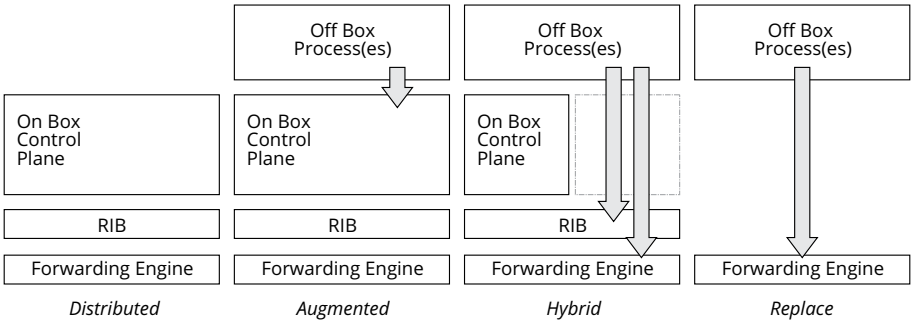


Figure 18-1 *Centralized control plane models*

- In the **hybrid** model, the centralized component of the control plane runs in parallel with the distributed control plane. From the distributed control plane process running on each device, the controller just appears to be another distributed control plane running in parallel (in effect). From the controller's perspective, much the same is true; the distributed control plane is just another control plane running in parallel with the controller.
- In the **replace** model, there is no distributed control plane; the centralized control plane is the only source of loop-free paths for the local switching device. One key marker of implementations using this model is the controller speaks directly to the forwarding table (FIB) rather than the RIB.

Within this framework, it is important to ask which *part* of the control plane—or more specifically, which functionality—is placed in the distribution control plane and which is placed in the centralized control plane.

Considering the Division of Labor

Which *part* of the control plane is centralized is *the* crucial question when considering SDNs and PNs. What are the *parts* of a control plane?

There are three "things" a control plane must provide to support applications and businesses: topology information, reachability information, and policy. Almost every control plane implemented since the beginning of network engineering time has assumed these three functions are part of a single "thing," and hence they must all be done in a single protocol.

Just as data planes are layered by function and location, however, it makes more sense to consider the control plane as a set of functions that can be split into layers. What would these layers look like?

- Discovering topology and advertising reachability are inseparable in some protocols, such as the Routing Information Protocol (RIP) and the Enhanced Interior Gateway Routing Protocol (EIGRP). In other protocols, such as Intermediate System to Intermediate System (IS-IS), the Shortest Path Tree (SPT) is calculated based on the topology, and reachability is "hung off the tree" as leaf nodes. Conceptually, then, it is difficult to see how topology and reachability could be separated into layers; the interplay between the two pieces of information is direct and immediate.
- Policy, on the other hand, relies on the topology and reachability information, but otherwise does not interact with topology and reachability. In fact, control plane policy is generally the process of overriding the basic topology and reachability information calculated by the control plane.

There appears to be a natural "split," with topology and reachability on one side of the divide, and policy on the other side of the divide. It is possible, then, to break the control plane into two "layers," with the bottom layer providing topology and reachability information, and the upper layer providing modifications to the paths calculated by the lower layer in order to implement specific policies.

BGP as an SDN

While the Border Gateway Protocol (BGP) was originally designed to interconnect networks operated by different companies—particularly transit service provider networks—providers with large-scale data centers realized it could be used to scale spine and leaf fabrics. Figure 18-2 is used for illustrating BGP as used in a data center.

Figure 18-2 shows a five-stage spine and leaf fabric using eBGP as a control plane; as there are no "cross links" in a spine and leaf, there is no iBGP between (using the row and column identifiers to label routers) routers 5a and 5b. Rows 1 and 5 are Top of Rack (ToR) devices, connected to servers hosting the applications using the fabric.

To provide the example, assume some flow should be pinned between 5b and 1d. It is always possible to manually configure each router in the network with static routes to pin this one flow to a specific path, but this creates a lot of opportunities for configuration mistakes.

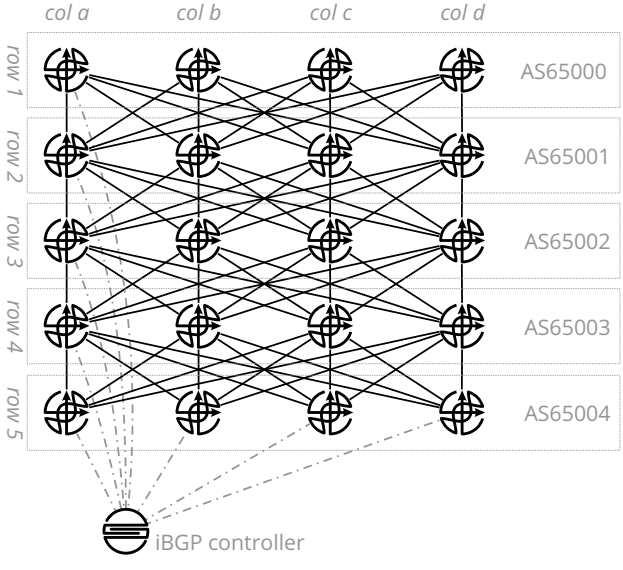


Figure 18-2 *BGP in a data center fabric*

Note

Of course, you could always automate the configuration. But automation does not really reduce the amount of complexity; it just relocates the complexity from the human-to-device interface into a three-layer structure, human-to-automation system, automation system to device. In other words, automating a complex configuration does not make the configuration less complex; it just makes the complexity less apparent. There is no doubt this can sometimes be a good thing, but there is also no doubt automating a bad process does not improve the process. Automation can solve many things, but network engineers need to be careful in thinking automated configurations will "solve all problems."

Another option, particularly since BGP is already running on every router in the network, is to use BGP as an SDN. Toward this end, an iBGP controller, shown at the bottom of the diagram, is connected to every router in the fabric.

Note

Only a small number of the iBGP connections are shown so the illustration remains readable.

Once the iBGP sessions are in place, the controller can "read" the entire topology and use local policies to determine which path the flow should be pinned to, and also which flows need to avoid the path over which the pinned flow is passing. For instance, assume the flow should be pinned to the [5b,4a,3c,2b,1d] path. A lower-cost path toward the destination (behind 1d) through 4a can be injected at 5b, and again through 3c at 4a, and again through 2b at 3c, etc., until the best path at each router along the path is through the selected path. The easiest way to accomplish this in BGP would be to inject a route from the controller with a lower local preference—but there are many ways to express such a policy in BGP.

This is an example of an augmented model; the centralized part of the control plane interacts with the distributed control plane (eBGP) directly. This is a rather interesting version of a hybrid model implementation, however, in that the protocol used to push policy (the southbound interface) is the same as the protocol used to discover and distribute topology and reachability information.

Fibbing

Link state protocols, unlike BGP, are focused on finding the *shortest* path to a given destination; while most implementations do support tags that can be carried in the protocol, these tags are rarely (actually never) used to modify traffic flow. The reason for this is fairly simple: the link state database must be synchronized among all routers. If two routers have a different view of the network topology, it is possible they will compute a looped path through the network.

Fibbing works within this set of constraints to allow traffic-engineered paths to be computed without modifying the link state protocol, such as Open Shortest Path First (OSPF) or Intermediate System to Intermediate System (IS-IS). Essentially, fibbing works by inserting false nodes, similar to pseudonodes, into the link state database, causing OSPF and IS-IS to change the shortest path, and hence engineering traffic flows through the network.

> **Note**
>
> This technique requires the route type used to create these fake nodes be able to carry a third-party next hop; v1, for instance, must be able to set the next hop for h1, which has the same address as H, to D, rather than to the fake node itself. Among link state protocols, as of this writing, only one kind of route can carry a third-party next hop: the OSPF external route. This means destinations for which traffic is engineered using fibbing must be external routes, and the fake nodes and other information the controller injects must also be OSPF external routes.

Figure 18-3 illustrates one possible way in which such fake nodes can be inserted into the network to modify traffic flow.

Figure 18-3 illustrates three stages in the same network: the first stage is the network without fibbing, the second is with fibbing nodes included to alter the best path chosen by OSPF, and the third is after the fibbing nodes have been optimized.

In 1, the top network illustrated in Figure 18-3, OSPF would choose the best path from A to H along [A,B,C,F,H], as this path has a total cost of 40. The next shortest path is through [B,D,E,F] or [B,D,E,G], both of which have a cost of 50. The policy to be applied to this network is to force the A to H traffic along the path [A,B,D,E,G,H]. The first step is adding a controller that can consume the link state database by participating in OSPF, and can also inject new LSAs into the network. This controller is attached to F and is labeled K in the diagram.

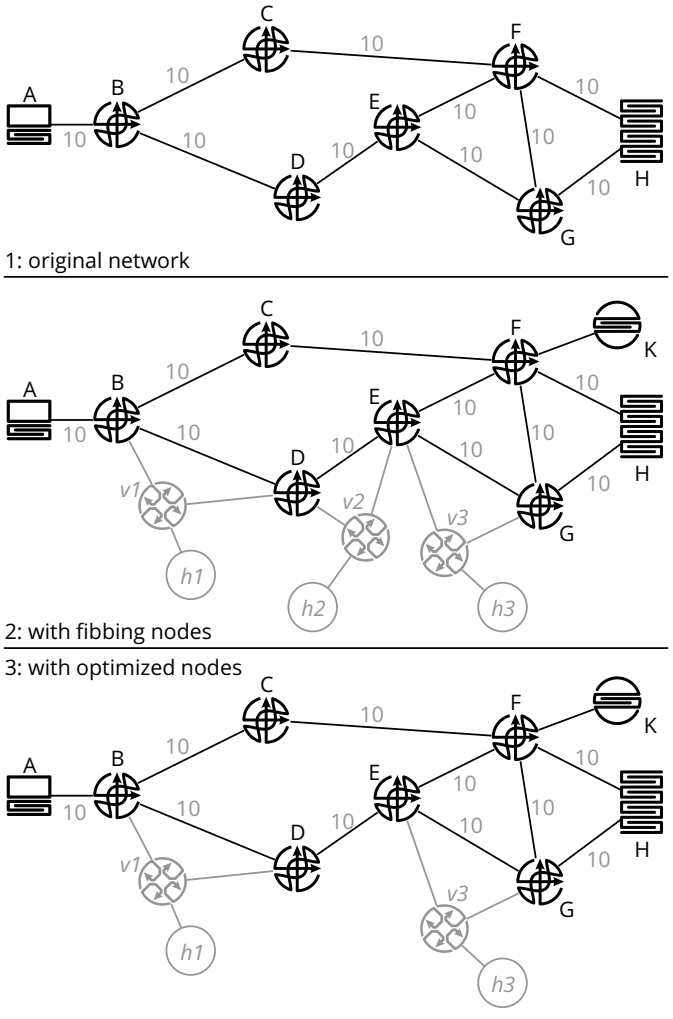


Figure 18-3 *Fibbing traffic engineering process*

To put the policy in place, the controller must convince

- B that the shortest path toward H passes through D.
- D that the shortest path toward H passes through E.
- E that the shortest path toward H passes through G.

To do this, the controller can inject three LSAs for fake nodes into the network, *v1*, *v2*, and *v3*, each of which advertises the destination H as directly connected (shown as *h1*, *h2*, and *h3* on the diagram):

- The advertisement for *h1* from *v1* has D set as the next hop, so that if B chooses this path toward H, the traffic is forwarded to D rather than *v1*.
- The advertisement for *h2* from *v2* has E set as the next hop, so that if B chooses this path toward H, the traffic is forwarded to E rather than *v2*.
- The advertisement for *h3* from *v3* has G set as the next hop, so that if B chooses this path toward H, the traffic is forwarded to G rather than *v3*.

The controller must also advertise some new links—specifically:

- [B,v1] with some cost lower than 40
- [v1,B] with an infinite cost
- [v1,D] with any cost
- [D,v1] with an infinite cost
- [D,v2] with any cost less than 30
- [v2,D] with an infinite cost
- [E,v2] with any cost
- [v2,E] with an infinite cost
- [E,v3] with any cost less than 20
- [v3,E] with an infinite cost
- [v3,G] with any cost
- [G,v3] with an infinite cost

Given this set of nodes and links:

- B will compute the path to H through v1, and forward the traffic toward H to D (because the next hop advertised by v1 to h1 is through D).
- D will compute the path to H through v2, and forward the traffic toward H to E (because the next hop advertised by v2 to h2 is through E).
- E will compute the path to H through v3, and forward the traffic toward H to G (because the next hop advertised by v3 to h3 is through G).

These alternate best paths, then, will carry the traffic along the path [A,B,v1(to D),v2(to E),v3(to G),H]. Adding a node per hop might seem inefficient; hence the fibbing process includes an optimization step. At each hop along the calculated path, the algorithm can compute where each router would forward traffic anyway. In the case of B, the shortest path is normally through C, so the fake node is required to redirect the traffic. In the case of D, however, the shortest path is normally through E, which is the correct path; the fake node does not need to be created to convince D to forward traffic toward H through E. In the case of E, there are two equal cost paths; the fake node would be needed to force E to choose the correct path of the two. The final network, 3, illustrated in Figure 18-3, shows the optimized set of fake nodes inserted in the network.

I2RS

Work on the Interface to the Routing Systems (I2RS) began in the Internet Engineering Task Force (IETF) in 2012. The original charter was to build an interface into the RIB to act as an interface between the RIB and an off-device process or application. To quote the problem statement RFC7920, directly:

> Traditionally, routing systems have implemented routing and signaling (e.g., Multiprotocol Label Switching, or MPLS) to control traffic forwarding in a network. Route computation has been controlled by relatively static policies that define link cost, route cost, or import and export routing policies. Requirements have emerged to more dynamically manage and program routing systems due to the advent of highly dynamic data-center networking, on-demand Wide Area Network (WAN) services, dynamic policy-driven traffic steering and service chaining, the need for real-time security threat responsiveness via traffic control, and a paradigm of separating policy-based decision-making from the router itself. These requirements should allow controlling routing information and traffic paths and extracting network topology information, traffic statistics, and other network analytics from routing systems.[1]

Figure 18-4 illustrates the architecture of I2RS.
In Figure 18-4, there are several critical components:

1. Atlas, Ward, and Nadeau, *Problem Statement for the Interface to the Routing System*.

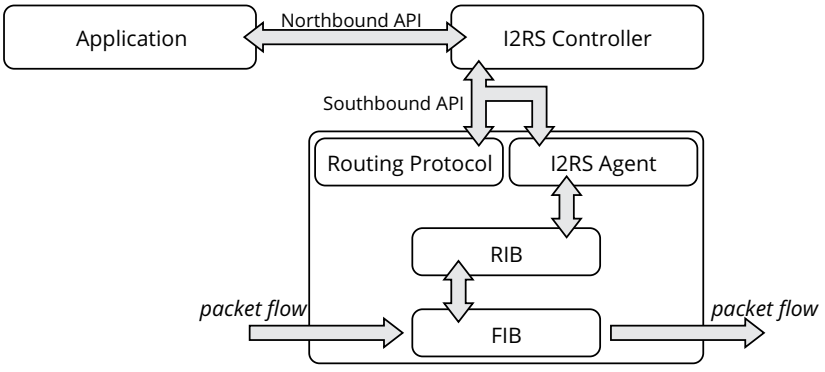


Figure 18-4 *I2RS architecture*

- The *application*, which is normally some sort of network-level orchestration package providing a "business policy" or "intent-focused" interface to the user. This application is responsible for translating intent into some form of input that the I2RS controller can understand, or translating the information that the I2RS controller provides into some form of human-readable information (such as an overlay view of all the topologies currently enabled on the network).
- The *northbound Application Programming Interface (API)*, which is not defined by the I2RS specifications.
- The *I2RS controller*, which is a package executing on a server someplace (virtual or otherwise). This translates the intent and "human readable" requests from the application into the format of the *southbound API*.
- The *southbound API*, which is YANG-modeled data carried over one of several different transport mechanisms.
- The *I2RS agent*, which does two things:
  - Translates the YANG-modeled data into local RIB API calls to install, remove, and modify routes.
  - Translates local RIB and routing information into YANG models describing the topology of the network (including overlay topologies).

It is possible to run only an I2RS agent on each router, replacing the distributed control plane completely. In this case, the controller would take the connected interface and destination information from the RIB, possibly using information from other protocols (such as the Link Local Discovery Protocol, or LLDP), to verify adjacent connected routers, to build a complete view of the network. Based on

this information, the controller could use any one of the loop-free path calculation mechanisms to calculate a set of loop-free routes through the network, modify them based on the policy being fed to the controller by the application(s), and then distribute the resulting routes to the RIB at each router. Deploying I2RS in this way would be an example of the *replace* model discussed in the first section of this chapter.

I2RS is not designed to be deployed in *replace* mode, however. The I2RS agent, which interfaces with the RIB in the same way any distributed routing protocol running on the device does, allows I2RS to act in parallel with other control planes; this would fall under the *hybrid* mode considered in the first part of this chapter. Figure 18-5 illustrates.

In Figure 18-5, A and H are both sending large streams of data to two different services residing on K. The shortest path, calculated by the routing protocol, from A to K is along the path [B,E,G]; the shortest path calculated by the routing protocol from H to K is [E,G]. If both of these flows are placed on the [E,G] link, it could overwhelm the link, so the network operator would like to move A's traffic to an alternate path. This might be expressed as a policy something like "the differential between the utilization of any two paths in the network should not be more than 20%," or something similar.

The controller, C, can then monitor each link in the network; when both A and H send traffic, the controller can note the [E,G] link is out of policy, and hence look for some alternate path over which to send some part of the traffic. The obvious choice will be the traffic originating at A; what is not so obvious is where to send this traffic. There are a number of options available to the controller, depending on the capabilities of each device in the network. For instance:

- If the network supports MPLS label stacks, the controller could impose a label stack on the traffic on the inbound port connecting B to A, causing the traffic to follow the path [B,E,F,G]; this would be implementing *segment routing* using I2RS to push the label stacks to network devices.

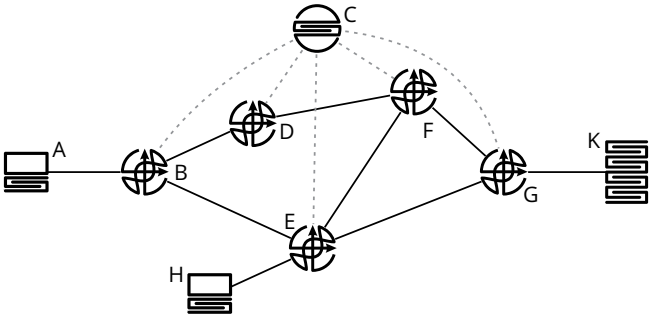


Figure 18-5 *An I2RS use case*

- If E supports forwarding based on the source and destination addresses, the controller could push a forwarding rule stating all traffic sourced from A, and destined to K, should be forwarded toward F instead of toward G; the controller would need to calculate that F will not forward the traffic back to E, of course, which would depend on the local link metrics.
- If F, for some reason, would normally use the path through E to reach K, the controller can set destination-based forwarding rules in B, D, F, and G to cause the traffic sourced from A, and destined to K, to follow the path [B,D,F,G].

All other traffic in the network would continue to follow the routes calculated by the distributed routing protocol running in parallel with I2RS. This means I2RS is being used in a *hybrid* model programmable network mode in this example. This is the operational role I2RS was designed to fill.

I2RS uses the YANG modeling language to describe forwarding and topology information. For instance, a route is modeled as a set of objects, as shown in Figure 18-6.

The three kinds of objects in a route model shown in Figure 18-6 are as follows:

- Route attributes, such as the metric.
- The route match, which is the portion of the route that is matched to the destination address; when being processed, the destination of the packet can be matched on an IPv4 address, an IPv6 address, an MPLS label, a Media Access Control (MAC) address, or an interface.
- When the route and the attributes match, the packet is sent to what is contained in the next hop field.

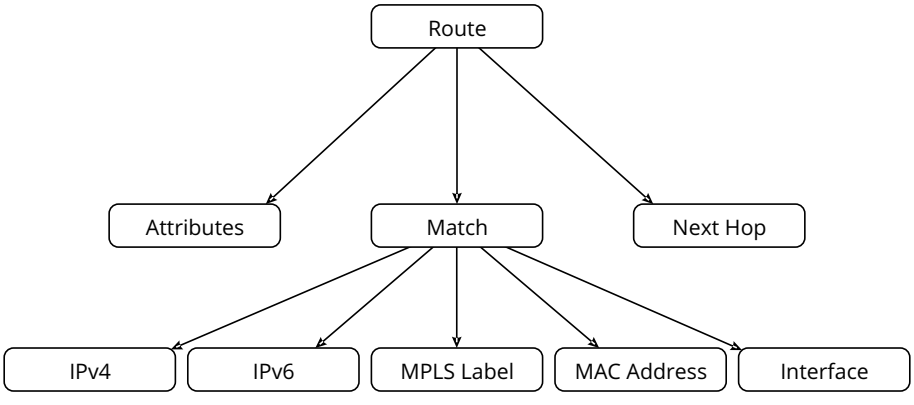


Figure 18-6 *The components of an I2RS route*

Why not define this in a single structure, rather than as a set of related objects? After all, this sort of structure appears to make the model of a single route more complex. The advantage here, however, is the same as the advantages of encoding information into a Type Length Vector (TLV); it is very easy to extend the model if some new kind of match is needed, some new attribute is needed, or some new kind of next hop is needed. One specific example is the idea of an equal cost multipath (ECMP) group. The next hop object can be a single next hop, or a collection of next hops in the form of an ECMP group, or even, perhaps, a next hop and a fast reroute next hop (an alternate next hop).

The model of each route, expressed in YANG, looks like this:

```
+--rw route-list* [route-index]
 |  +--rw route-index     uint64
 |  +--rw match
 |  |  +--rw (route-type)?
 |  |     +--:(ipv4)
 |  |     |  ...
 |  |     +--:(ipv6)
 |  |     |  ...
 |  |     +--:(mpls-route)
 |  |     |  ...
 |  |     +--:(mac-route)
 |  |     |  ...
 |  |     +--:(interface-route)
 |  |        ...
 |  +--rw nexthop
 |  |  +--rw nexthop-id?uint32
 |  |  +--rw sharing-flag?        boolean
 |  |  +--rw (nexthop-type)?
 |  |     +--:(nexthop-base)
 |  |     |  ...
 |  |     +--:(nexthop-chain) {nexthop-chain}?
 |  |     |  ...
 |  |     +--:(nexthop-replicates) {nexthop-replicates}?
 |  |     |  ...
 |  |     +--:(nexthop-protection) {nexthop-protection}?
 |  |     |  ...
 |  |     +--:(nexthop-load-balance) {nexthop-load-balance}?
 |  |        ...
 |  +--rw route-status
 |  |  ...
 |  +--rw route-attributes
```

```
|  |  ...
|  +--rw route-vendor-attributes
+--rw nexthop-list* [nexthop-member-id]
   +--rw nexthop-member-id    uint32
```

You can see each of the elements shown here in the diagram laid out in a human-readable, textual format within the YANG model.

PCEP

The original Path Control Element Protocol (PCEP) work dates from the early 2000s, with the first IETF RFC (4655) being made informational in 2006, which means PCEP predates the time when SDNs were "cool." PCEP was created because of the increasingly complex nature of computing Traffic Engineering (TE) paths through (primarily) Service Provider (SP) networks. Three specific developments drove the design, standardization, and deployment of PCEP:

- The complexity of calculating TE paths across large, dispersed networks with a lot of different available paths
- The complexity of calculating TE paths across multiple organizations and internal network boundaries; for instance multiple flooding domains, multiple interior gateway protocols stitched together with BGP, or multiple BGP autonomous systems
- The complexity of computing TE paths through multiple levels of abstraction, such as computing an MPLS TE path on top of an optical path; this includes the difficulty of computing Shared Risk Link Groups (SRLGs) where a large set of virtual topologies cross a complex set of physical (primarily optical) links

The state necessary to compute TE paths in each of these situations is either very difficult or impossible to assemble in a single distributed control plane. All of these functions require some sort of overlay controller-based network with visibility into the entire network, including the physical through the application layers, and across administrative and failure domain boundaries.

If this set of requirements is starting to sound familiar, it should be; many of the SDN type overlays discussed in this chapter were created to solve some variant of this problem set. Figure 18-7 illustrates the components of the PCEP ecosystem.

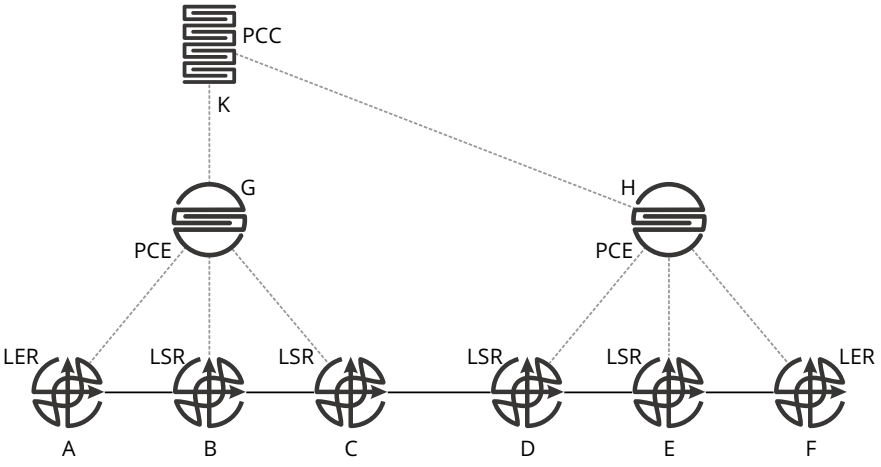


Figure 18-7 *The elements of a PCEP deployment*

There are four crucial components of PCEP shown in Figure 18-7:

- The PCC is the Path Computation Client; this is the application or service requesting a new TE path be configured through the network.
- The PCE is the Path Computation Element; this is the controller with the overall view of the network, and it computes the TE path through the network (normally using some form of Constrained SPF).
- The LER is the Label Edge Router; this is the head- and tailend of the TE Label Switched Path (LSP) through the network.
- The LSR is the Label Switch Router; these simply forward based on the labels as they are configured by the PCE using PCEP.

In a single network (domain or autonomous system), there may be multiple PCEs that may communicate in a number of different ways. For instance, PCEs may share topology information using a link state protocol or BGP (particularly if BGP is carrying topology information through BGP-LS). There may also be one or more PCCs. PCEP is also designed to build paths across domains or autonomous systems; a set of PCCs may communicate with one another to build a TE path across multiple provider networks, instructing local PCEs to set up the correct LSPs through each LSR along the path.

The way a TE path is normally designed in PCEP is each device is configured with a simple set of forwarding rules; any packet received with one label, say *X*, is forwarded out the indicated interface with a new label *Y*. This is exactly the same as any other MPLS technology that swaps the outer label at each hop.

PCEP, as a protocol, is highly tuned to the process of inserting the inbound label, outbound interface, and outbound label into the forwarding table at each LER and LSR. While PCEP does encode information into TLVs, there is no specific capability to insert filtering or traffic classification rules of any kind. The controller must be able to configure the LER to channel the correct traffic into the LSP headend in some way. It is possible, of course, to configure a label to be routed to the NULL0 interface, which effectively filters the packet stream, so it is possible to do some forms of packet filtering using PCEP.

PCEP falls into the *hybrid* model described in the first part of this chapter.

OpenFlow

OpenFlow made SDN technology "cool." The project began in 2006 with two sets of problems. The first was a project at Stanford built around centrally managing policy in a network. The second was a group of projects in other universities where researchers wanted to try new ways of building routing protocols; however, the hardware platforms available at the time were not something end users could modify by installing new routing code on them. These requirements breathed new life into the concept of separating the control and forwarding planes, driven by the idea of a standard protocol to carry information between the control plane and the FIB. Figure 18-8 illustrates the basic concept.

Figure 18-8 illustrates the most basic OpenFlow configuration. The switching device does not have any control plane at all, as the controller interacts directly with the FIB. OpenFlow provides a packet format and a protocol over which these packets can be carried that describes forwarding table entries in the FIB directly. The FIB, in OpenFlow documentation, is referred to as the *flow table*, as it contains information about each individual flow the switch needs to know about.

Note the wording here: *each individual flow*. This is because OpenFlow was originally designed to operate on any and (possibly) every field in a packet header.

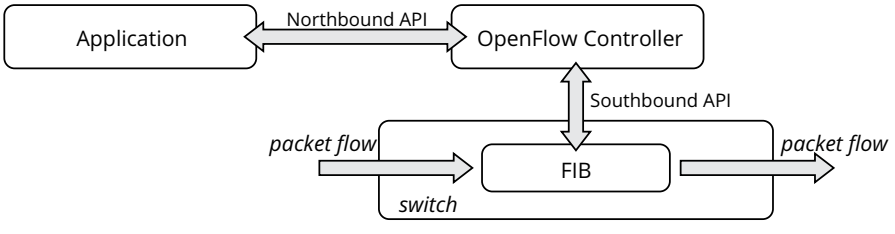


Figure 18-8 *Basic OpenFlow*

The controller specifies a set of bits and an offset the switch is supposed to match, and then a set of actions to take if a packet matches the specified pattern. The switch, then, can just check each packet it processes to see if it matches this pattern. The pattern might contain, for instance, the source and destination Internet Protocol (IP) addresses, the source and destination media access addresses, protocol numbers, port numbers, and just about anything contained in the packet header.

It is impossible to build hardware able to contain information on every flow passing through the device. It is impossible, as well, for the controller to know about every flow being initiated by every host attached to the network. To resolve these problems, OpenFlow is normally implemented as a reactive control plane. This means processing a new stream takes several steps:

1. The host starts sending packets in the new stream.
2. The first hop switch receives these packets and finds it has no flow label matching the new flow.
3. The first hop switch will send the packets to the controller.
4. The controller examines the packet, finds a matching policy (if there is one), and computes a loop-free path through the network.
5. The controller installs flow label information for this new flow in every switch through which packets in this flow will pass.
6. The switches now forward traffic normally.

Flow labels are cached, which means each flow label is held until it has not been used for some time. OpenFlow, then, was originally designed as, and is often deployed as, a reactive control plane, which means the control plane relies on information dynamically (in near real time) supplied by the data plane to build forwarding information.

This kind of processing is generally not scalable in many environments, particularly in the environment OpenFlow is considered ideal for—hyperscale data center fabrics for building private and public clouds. Because of this, many implementations rely on wildcard flow labels, which work much like IP routes; if a subset of the information is matched, the packet is processed based on the rules given for the partial match. Much like more traditional IP routing, the partial match is often the destination subnet.

While OpenFlow is often shown with an off-device controller, this is not the only deployment pattern where OpenFlow has been used. Figure 18-9 illustrates.

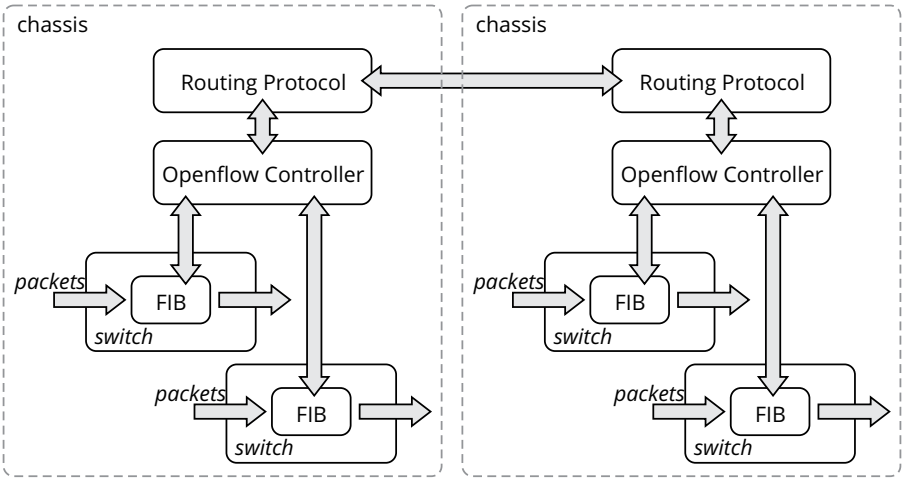


Figure 18-9 *OpenFlow in a chassis system*

In Figure 18-9, two chassis devices are represented. In each one, there is a processor (or compute engine) running a standard distributed routing protocol. This routing engine communicates with an OpenFlow controller *within* the device, perhaps running on the same processor. This controller then uses OpenFlow to send routes to individual line cards, each of which acts as a sort of independent switch. The entire unit might appear to be a fairly standard chassis switch, with OpenFlow being used as a sort of Interprocess Communication (IPC) system between the components. The advantage in such a design is that line cards with different sorts of processors can be used; so long as each kind of processor has an OpenFlow interface, the hardware under the controller (and within the stack or chassis) can be replaced fairly easily.

CAP Theorem and Subsidiarity

Centralization can often bring many benefits in terms of policy implementation; some engineers and researchers think centralized control planes are much simpler than distributed control planes. Why not completely centralize all control planes, then? The answer lies in another three-way tradeoff problem, much like the State/Optimization/Surface (SOS) three-way problem discussed in Chapter 1, "Fundamental Concepts." To understand this problem, consider Figure 18-10.

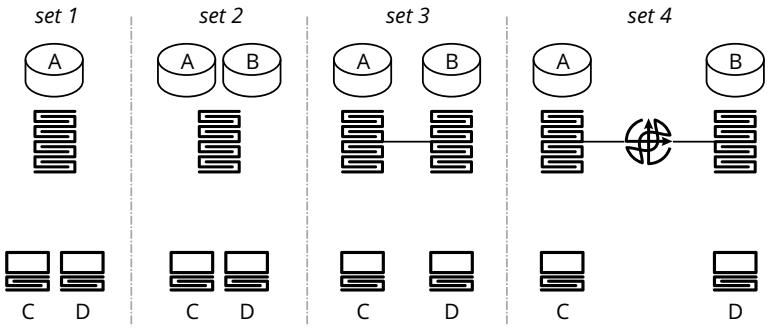


Figure 18-10 *Understanding the CAP theorem*

There are four *sets* illustrated in Figure 18-10:

1. A single database on a single server, accessed by two processes running on two hosts, C and D
2. A pair of databases containing the same information (which must be synchronized) running on a single server, accessed by two processes running on two hosts, C and D
3. A pair of databases containing the same information (which must be synchronized) running on a pair of servers connected by a single wire, accessed by two processes running on two hosts, C and D
4. A pair of databases containing the same information (which must be synchronized) running on a pair of servers connected through a router, accessed by two processes running on two hosts, C and D

Now consider what happens in each case if C writes some piece of information and D immediately reads it:

1. The information is written to the database; when D reads the information, it will be identical to what C has written.
2. The information is written to one database, and it takes a few moments to be synchronized to the other database because it must be transferred across some sort of internal bus so it can be copied from one database to the other. If D reads the information immediately from B, it will receive the old information; D must wait for the synchronization process to complete to see an accurate copy of the information (or the database).

3. Once C has written the information to copy A, the information must cross an internal bus to a network interface, be marshaled into some form of data packet, serialized onto the wire (potentially after being queued for a few moments), copied off the wire at the second server, passed over the second server's internal bus, and then synchronized to the second copy of the database.
4. Once C has written the information to copy A, the information must cross an internal bus to a network interface, be marshaled into some form of data packet, serialized onto the wire (potentially after being queued for a few moments), copied off the wire by the router, processed and switched in memory, queued in the router, serialized back onto the wire, copied off the wire at the second server, passed over the second server's internal bus, and then synchronized to the second copy of the database.

It should be obvious that the farther apart the two copies of the database are logically, the longer it will take for the information in copy B to match the information in copy A after C has finished writing. This is the first third of the CAP theorem: partitionability. The database in set 1 is not partitioned; as you move left to right, the database becomes more "strongly" partitioned by adding more processes that the information must pass through before the two copies of the database can be synchronized.

Assume you must ensure that the information D retrieves is exactly what C writes. The simplest way to ensure this is to simply block D from reading the copy at B until you know the two copies are synchronized. To put this in other terms, you can block D's access to the database. This is the second third of the CAP theorem, the "A"—accessibility. You can solve the synchronization problem solved by partitioning the database by making the database unreadable part of the time, or less accessible.

An alternate assumption might be that you do not need B to read precisely the same information as C has written; hence the read and write do not need to be consistent. This is the third third of the CAP theorem, the "C"—consistency.

Putting this all together, the CAP theorem states there are three design parameters in building a database: *consistency*, *accessibility*, and *partitionability*. You can choose, in some measure, two of the three.

How does this apply to control planes? The answer is quite simple: if you want to have a consistent view of the network, you must somehow block access to the database containing the description of the network topology during some periods of time (specifically while the network is converging). What distributed routing protocols do is to allow access all the time, and simply "live with" the inconsistencies resulting from this "always available" distributed database of topology and reachability information.

Centralized control planes, however, face a double problem. First, the database is now distributed between the actual forwarding devices and the device with the database describing the network. Second, there cannot be "only one controller"—this would create an unacceptable single point of failure. To prevent having a single point of failure, there must be at least two controllers. Those controllers must be synchronized in some way.

So centralized control planes face a number of challenges, such as

- Ensuring the actual state of the network is reflected from the devices that are connected to links and destinations into the controller
- Ensuring controllers have a consistent view of the network, or that an inconsistent view of the network does not cause systemic, large-scale failures in some way
- Ensuring the information needed to forward packets is available at individual forwarding devices fast enough that forwarding does not suffer because of the distributed nature of the control and forwarding planes

There are solutions in each of these spaces, but they often introduce as much complexity as a distributed control plane in the first place. Quite often, hybrid models are chosen to balance between the complexity of distributed control planes and the complexities of centralized control planes.

One interesting way to think about centralization and decentralization is through the *subsidiarity principle*. Applying subsidiarity, which arises out of the social teaching of Thomas Aquinas, might seem to go far afield of engineering, but consider the principle itself:

> This tenet holds that nothing should be done by a larger and more complex organization which can be done as well by a smaller and simpler organization.[2]

The "root" of the subsidiarity principle is this: decisions should be made as close as possible to the information the decisions themselves depend on. Applying this principle to network engineering means thinking about where information is *produced* and placing any *decision maker* (generally a protocol, process, etc.) as close to the source of information as possible. Looking at this from a CAP theorem perspective, putting the decision maker close to the source of the information on which the decision maker depends reduces the amount of time between the information being available and the decision being made.

What does this suggest in the network engineering world? Policy comes primarily out of business decisions, and business decisions should be close to the business, not

2. Bosnich, "The Principle of Subsidiarity."

the topology. Hence, policy, or least some element of policy, is often best done when centralized. Topology and reachability, however, are grounded in what should be the only source of truth about the state of the network, the network itself. Therefore, it makes sense that decisions related to the topology and reachability, from detection to reaction, should be kept close to the network itself; hence, topology and reachability decisions should trend toward being decentralized.

Final Thoughts on Centralized Control Planes

There are no absolutes in the world of network engineering; if you have not found the tradeoff, you have not looked hard enough. This is true of choosing when and where to centralize, and when and where to decentralize. These choices are presented as either/or absolute choices far too often. The reality is that different problems often require different solutions.

Centralization and distribution, in terms of solving the many different problems control planes must resolve in the real world, provide network and protocol designers with a number of tradeoffs. A range of possible solutions seems obvious when considering these two different ways of providing reachability and topology information; for instance:

- Centralize reachability and topology discovery in a set of distributed controllers, *replacing* the distributed control plane with a centralized one. This model does not truly assume a "centralized" control plane, so much as it does removing the processing of information discovered about the network out of the individual switching devices. Distribution of the information across a number of devices is still key in creating a resilient design.
- Centralize some part of the function of the control plane, normally the policy, and distribute the remaining parts. PCEP, I2RS, and many other "overlay" control planes take this route.
- Decentralize all of the control plane components, including reachability, topology, and pushing policy. There are actually very few large-scale networks fully decentralized in this way; at least some policy is normally.

There is, in the end, no simple answer to the problems control planes pose in the real world. Between the three-way tradeoff implied by complexity theory (state, optimization, and surface) and the CAP theorem (consistency, accessibility, and partitioning), designers can make a wide range of choices when building networks and protocols.

Choose wisely.

Further Reading

Atlas, Alia, David Ward, and Thomas Nadeau. *Problem Statement for the Interface to the Routing System*. Request for Comments 7920. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7920.txt.

Bjorklund, Martin. *The YANG 1.1 Data Modeling Language*. Request for Comments 7950. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7950.txt.

Bosnich, David A. "The Principle of Subsidiarity." *Religion & Liberty* 4, no. 4 (July 2010). https://acton.org/pub/religion-liberty/volume-6-number-4/principle-subsidiarity.

Clarke, Joe, Gonzalo Salgueiro, and Carlos Pignataro. *Interface to the Routing System (I2RS) Traceability: Framework and Information Model*. Request for Comments 7922. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7922.txt.

Doria, Avri, Ligang Dong, Weiming Wang, Hormuzd M. Khosravi, Jamal Hadi Salim, and Ram Gopal. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. Request for Comments 5810. RFC Editor, 2010. https://rfc-editor.org/rfc/rfc5810.txt.

Hares, Susan, and Mach Chen. "Summary of I2RS Use Case Requirements." Internet-Draft. Internet Engineering Task Force, November 2016. https://tools.ietf.org/html/draft-ietf-i2rs-usecase-reqs-summary-03.

Hares, Susan, Qin Wu, and Russ White. "Filter-Based Packet Forwarding ECA Policy." Internet-Draft. Internet Engineering Task Force, October 2016. https://tools.ietf.org/html/draft-ietf-i2rs-pkt-eca-data-model-02.

Medved, Jan, Nitin Bahadur, Hariharan Ananthakrishnan, Xufeng Liu, Robert Varga, and Alexander Clemm. "A Data Model for Network Topologies." Internet-Draft. Internet Engineering Task Force, March 2017. https://tools.ietf.org/html/draft-ietf-i2rs-yang-network-topo-12.

Medved, Jan, Nitin Bahadur, and Sriganesh Kini. "Routing Information Base Info Model." Internet-Draft. Internet Engineering Task Force, December 2016. https://tools.ietf.org/html/draft-ietf-i2rs-rib-info-model-10.

Medved, Jan, Robert Varga, Hariharan Ananthakrishnan, Nitin Bahadur, Xufeng Liu, and Alexander Clemm. "A YANG Data Model for Layer 3 Topologies." Internet-Draft. Internet Engineering Task Force, January 2017. https://tools.ietf.org/html/draft-ietf-i2rs-yang-l3-topology-08.

Nadeau, Thomas, Alia Atlas, Joel M. Halpern, Susan Hares, and David Ward. *An Architecture for the Interface to the Routing System*. Request for Comments 7921. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7921.txt.

Prieto, Alberto Gonzalez, Eric Voit, Ambika Tripathy, Einar Nilsen-Nygaard, Balazs Lengyel, Andy Bierman, and Alexander Clemm. "Subscribing to YANG Datastore Push Updates." Internet-Draft. Internet Engineering Task Force, October 2016. https://tools.ietf.org/html/draft-ietf-netconf-yang-push-04.

Vissicchio, Stefano, Laurent Vanbever, and Jennifer Rexford. "Sweet Little Lies: Fake Topologies for Flexible Routing." In *ACM HotNets*. Los Angeles, California, 2014.

Wang, Lixing, Hariharan Ananthakrishnan, Mach Chen, Sriganesh Kini, and Nitin Bahadur. "A YANG Data Model for Routing Information Base (RIB)." Internet-Draft. Internet Engineering Task Force, January 2017. https://tools.ietf.org/html/draft-ietf-i2rs-rib-data-model-07.

Review Questions

1. Research Microsoft's SWAN architecture. How would you classify this architecture? What parts of the control plane are centralized, what parts are distributed, what interface is used, and what is the southbound protocol?
2. Research Google's FirePath architecture. How would you classify this architecture? What parts of the control plane are centralized, what parts are distributed, what interface is used, and what is the southbound protocol?
3. Research OpenFabric. How would you classify this architecture? What parts of the control plane are centralized, what parts are distributed, what interface is used, and what is the southbound protocol?
4. Research RESTful interfaces. What is the difference between a RESTful and non-RESTful interface?
5. Among the four possible interfaces, what interface did the forCES protocol interact with?
6. Research OpenFlow hybrid mode. Why did the developers of the OpenFlow protocol abandon this idea? How would this mode have changed the classification of the OpenFlow protocol?
7. One of the problems facing operators who use BGP as a southbound interface is the lack of a full view of the network topology. How does BGP-LS (Link State) solve this problem?
8. What are the state, surface, and optimization tradeoffs in fibbing?

This page intentionally left blank

Chapter 19

Failure Domains and Information Hiding

Learning Objectives

When you finish reading this chapter, you should understand:

- The reason for hiding information in the control plane
- The concepts of control plane state scope and speed
- Feedback loops and the dangers of positive feedback loops
- The difference between summarizing topology and aggregating reachability
- Filtering reachability information
- Layering control planes
- Caching control plane information

The intentional modification or shaping of traffic flows across a network is not the only kind of policy that network engineers must interact with. Information hiding, while not often considered a form of policy, relates to the larger goals, or policies, of building scalable, repeatable networks. These policies have consequences in terms of traffic flow, although these consequences are often unintentional rather than intentional—which means they are often ignored. This chapter and the next, Chapter 20, "Examples of Information Hiding," are dedicated to considering this one problem, the solution space, and some widely used solution implementations. The first section in this chapter will examine the problem space, the second various kinds of

solutions that can be used to counter the problem, and the third section will consider information hiding in the context of network complexity.

The Problem Space

Control planes are designed to learn about and carry as much information about the network topology and reachability as possible. Why would network engineers want to limit the scope of this state, once the processing and memory have been spent to learn it? There are several answers, including

- To reduce resource utilization in devices participating in the control plane, generally just to save costs
- To prevent a failure in one part of a network from impacting some other part of the network; in other words, to break up the network into failure domains
- To prevent leaking information about the topology of the network, and reachability to destinations attached to the network, to attackers; in other words, to reduce the network's attack surface
- To prevent positive feedback loops that can cause a complete network failure

The problems in the preceding list can be divided into two categories: reducing the scope of control plane information and reducing the speed at which control plane information is allowed to change. These will be considered in the two following sections.

Defining Control Plane State Scope

Figure 19-1 illustrates the scope of control plane state.

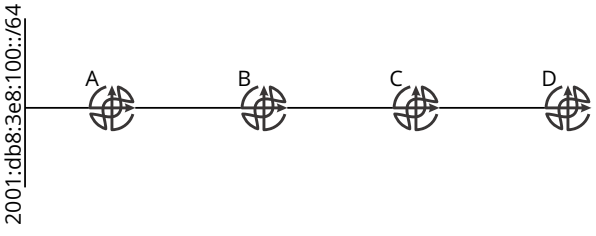


Figure 19-1 *The scope of control plane state*

There are two kinds of state carried by the control plane: topology and reachability. These two kinds of control plane state can have different scopes in a network. For instance:

- If D has knowledge of 2001:db8:3e8:100::/64, then the scope of this reachability information is A, B, C, and D—the entire network.
- If C has knowledge of 2001:db8:3e8:100::/64, and D does not, then the scope of this reachability information is A, B, and C.
- If D knows about the link connecting A and B, or that A and B are adjacent, the scope of this topology information is A, B, C, and D—the entire network.
- If D does not know about the link connecting A and B, or that A and B are adjacent, the scope of this topology information is A, B, and C.

Another way to look at this is to ask: if a link or reachability to a specific destination fails, which devices must participate in convergence? Any device that does not participate in convergence, perhaps by sending an update, recalculating the set of loop-free paths through the network, or switching to an alternate path, is not part of the failure domain. Any device that does need to send an update, recalculate the set of loop-free paths, or switch to an alternate path is part of the failure domain. The scope of a failure, then, determines the scope of the failure domain. In Figure 19-1:

- If D has knowledge of 2001:db8:3e8:100::/64, then D must recalculate its set of reachable destinations if 100::/64 is disconnected from A; hence D is part of the failure domain for this destination.
- If D does not have knowledge of 2001:db8:3e8:100:/64, then D does not change its local forwarding information when 100::/64 is disconnected from A; hence D is not part of the failure domain for this destination.
- If D has knowledge of the link between A and B, then D needs to recalculate the set of loop-free paths through the network if the link fails (along with any reachability information passing through the link); hence D is part of the failure domain for this specific link.
- If D does not have knowledge of the link between A and B, then D does not need to recalculate anything when the link fails; hence D is not part of the failure domain.

This definition means failure domains must be determined for each piece of reachability and topology information. While protocols and network designs will

block reachability and/or topology at common points in a network, there are cases in which

- Topology information is blocked, but not reachability information.
- Some reachability information is blocked, but not all.
- Some reachability or topology information leaks, causing a leaky abstraction.

The scope of control plane information within a network is important because it has a very large impact on the speed at which the control plane converges. Each additional device required to recalculate because of a change in topology or reachability represents some amount of time the network will remain unconverged, and hence either some destinations will be unnecessarily unreachable, or packets will be looped across some set of links in the network because some routers have a different view of the network topology than others. Looping, in particular, is a problem, because loops quite often have the potential to become positive feedback loops, which can cause the control plane to fail to converge permanently.

Positive Feedback Loops

Positive feedback loops are a bit harder to imagine than the scope of control plane information; Figure 19-2 illustrates.

In Figure 19-2, there are four devices:

- Device A, which adds whatever it receives from the signal input and what it receives from B
- Device B, which can either increase or decrease the size or frequency of the signal it receives from C
- Device C, which passes the signal along unchanged to D, and also samples the signal, sending the sample to B
- Device D, which measures the signal

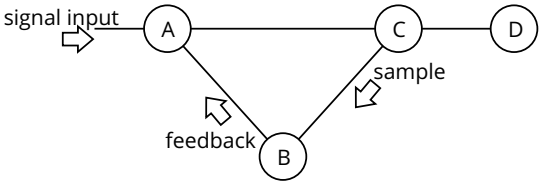


Figure 19-2 *A sample circuit to illustrate positive feedback loops*

To create a simple feedback loop, assume C samples some fraction of the signal passing through it, passing this sample to B. Device B, in turn, amplifies the sample by some factor, and passes this amplified signal back to A. Figure 19-3 shows the result.

The case shown in Figure 19-3 is a positive feedback loop; C amplifies the sample it receives, making the signal just a bit larger. The result, at D, is a signal with constantly increasing amplitude. When will this feedback loop stop? When some limiting factor is hit. For instance, A may reach some limit where it cannot continue to add the two signals, or perhaps C reaches some input signal limit and fails, releasing its magic smoke (as all electronics will do if driven with too much input power). It is also possible to set up a negative feedback loop, where C removes a slight bit of power each cycle; in the case of a sine wave (as shown here), this would require C to invert the sample it receives from A. Finally, it is possible to configure each component in this circuit to neither increase nor decrease the final output at D. In this case, C would be somehow tuned to compensate for any inefficiency in the wiring, or A or C's operation, by injecting just enough feedback to A to keep the signal at the same power at D.

Figure 19-4 changes the amplitude of the output signal to the frequency of an event to illustrate why.

In Figure 19-4, B (as shown previously in Figure 19-2) is programmed to send a single event for every pair of events it receives. In the original signal input, there are six event signals, so B adds three more into the feedback path toward A. In the second round, shown in the center column, the original six events from the input signal are added to the three from B, resulting in nine event signals. Based on these nine event

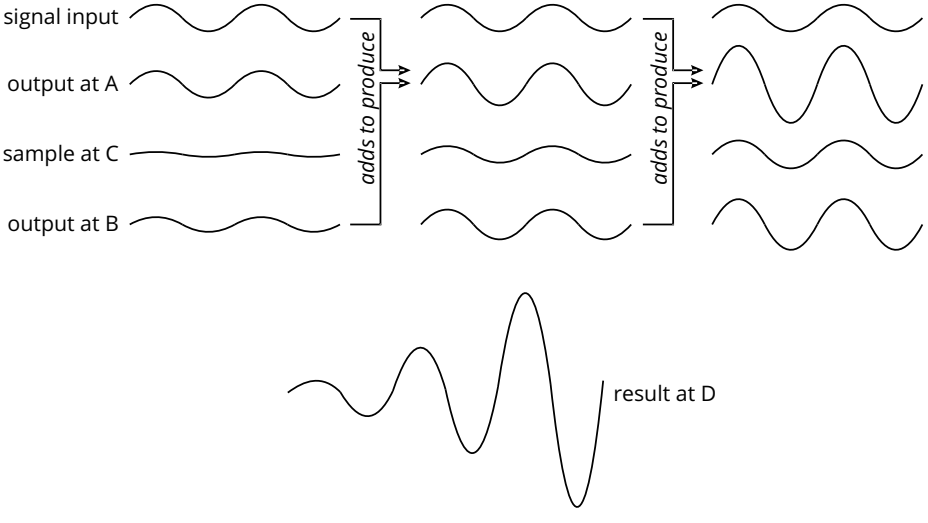


Figure 19-3 *Result of a positive feedback loop*

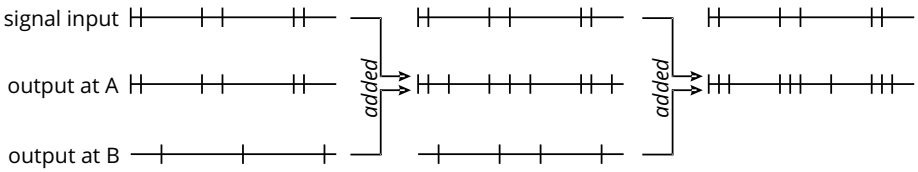


Figure 19-4 *Positive feedback loop using events*

signals at the output of C, B will generate four event signals and feed them back to A. The result is that the output of A now has ten event signals. This increase in the number of signals will continue until the entire time space is saturated with event signals.

Physical and logical loops can cause links to become saturated, devices to run out of processing power or memory, or a number of other conditions that will eventually cause a network failure. Figure 19-5 is used to provide an example.

Assume that each router in Figure 19-5 is capable of processing ten changes to the network per second—either a route or topology change, for instance—and there are five routes total in the routing table. Because of the speeds of the interfaces (or for some other reason), the order in which updates are transmitted through the network is always [D,A,C,B]; updates from D through [A,C] always arrive at B before updates through [D,A] directly.

The 2001:db8:3e8:100::/64 link begins to flap three times per second. It seems like the network should converge on this flap rate fine; it is 50% of the rate at which any device can support, after all. To understand the impact of the feedback loop, however, it is important to trace the entire process of convergence:

- Each time the 100:/64 link fails or comes up, D sends an update to A; this is three failures and three recoveries, for a total of six events per second.
- For each of these events, D will send an update to A.
- For each of these events, A will send an update to B and C.
- B will also send an update toward C for each update it receives; this effectively doubles the rate of events at C to 12 per second.

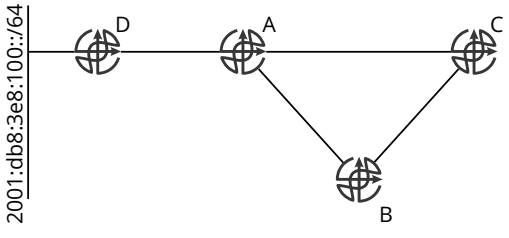


Figure 19-5 *A permanent control plane failure due to a positive feedback loop*

During the first second C receives 12 events per second, it will fail, in turn taking down its relationships with A and B. When it comes back up, it will attempt to establish new adjacencies with each of the connected routers, which means it will send its entire database, containing five routes, to A and B. Given the 100::/64 link is still flapping at the same rate, this will drive B above its threshold, causing B to crash. It is possible, as well (depending on the timing), that A could crash.

Once A crashes, the chain of crashes through resource exhaustion will continue—if the timing is correct, or the crashes form their own self-supporting feedback loop, even if the original flapping link is repaired. Although feedback loops of this kind are not tagged as the root cause of the failure (the flapping link would be considered the root cause of the failure in this example), they are often what turns a single event into a complete failure of the control plane to converge.

The Solution Space

A number of solutions have been developed over the years to limit control plane state, including summarization, aggregation, filtering, layering, caching, and back-off timers. All of these solutions fall into one of two different ways to limit control plane state—reducing the *scope* or the *speed* of control plane information. Each of these, in turn, solves a specific problem, such as

- Reducing the scope of control plane information improves security by controlling the set of devices through which a view of the network can be obtained.
- Reducing the scope of control plane information improves convergence by controlling the set of devices that must recalculate loop-free paths through the network because of any individual change.
- Reducing the scope of control plane information reduces the chance of positive feedback loops by preventing state from "looping back" through the control plane.
- Reducing the scope of control plane information reduces the chance of resource exhaustion in any particular device (and potentially lowers the cost of any particular device) by reducing the size of any tables held in memory and across which the set of loop-free paths must be calculated.
- Reducing the speed of control plane information traveling through the network, or the velocity of state, reduces the chance of positive feedback loops forming and reduces the chance of resource exhaustion in any individual device.

The following sections consider several widely implemented and deployed techniques used to control the scope and velocity of state.

Summarizing Topology Information

Topological information can be summarized by making destinations that are physically (or virtually) connected several hops away appear to be directly attached to a local node, and then removing the information about the links and nodes in any routing information carried in the control plane from the point of summarization. Figure 19-6 illustrates this concept from the perspective of F, with E summarizing.

Before the topology is summarized (the upper network), F might (depending on the protocol) know A is connected to B, B is connected to C and D, and C and D are connected to E. If E begins to summarize the topology information (shown in the lower network), each of these other nodes appears, from F's perspective, to be directly connected to E. The physical topology does not change, of course, but F's view of the topology does change.

Summarization is a form of abstraction over the network topology; the set of reachable destinations is abstracted from the network and connected so that loop-free paths are preserved, but not detailed topology information. The way this is normally done is to remove actual link information while preserving the metric information associated with each destination, as the metric information alone can be used to calculate loop-free paths.

Distance vector protocols essentially summarize topology information at every hop, as they transmit each destination with a metric between devices. In Bellman-Ford, the local device examines its local view of the network to calculate the set of loop-free paths through the network. In Garcia-Luna's Diffusing Update Algorithm

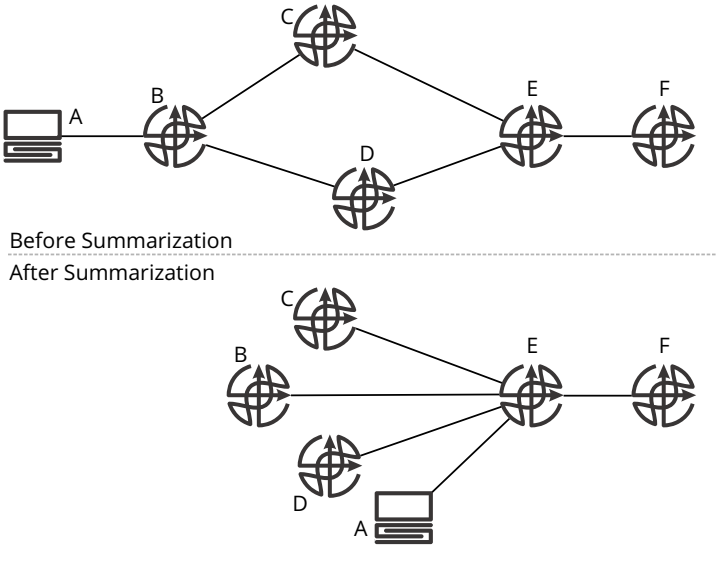


Figure 19-6 *Summarization of topology information in the control plane*

(DUAL), the device keeps (in effect) one hop of topology information, the cost to each destination as seen from each of its neighbors, and uses this information to calculate alternate loop-free paths to each destination. Link state protocols carry full topology information, including links and metrics, within a single flooding domain.

Aggregating Reachability Information

If you take a trip to a distant city through a series of flights, you will need

- Directions from your home to the local airport
- Directions within the local airport to the correct gate to board the aircraft
- Directions from gate to gate within the airport where each flight connection is made
- Directions from the gate to the place where you pick up a rental car, or to a taxi, or to some form of public transportation
- Directions to the hotel where you will be staying
- Directions from the hotel to the site of the meeting or conference you will be attending

What would happen if you called your destination hotel and asked for full directions to its location from yours? Assuming the hotel staff even know how you are traveling, the directions would easily overwhelm you. Maybe they would look something like this:

1. Walk out your front door and get into your car.
2. Turn left out of your driveway, go to the first stop sign, turn left.
3. Proceed three blocks and turn right onto the entrance ramp onto the highway.
4. Merge into traffic and stay on this road for 4.1 miles.
5. …
6. When you disembark from the plane, turn left on exiting the gate.
7. Travel 400 yards to the internal airport transportation station.
8. Ascend the steps or escalator to the second level, turn left, and board the first train arriving there.
9. On the third stop, exit the train, turn left, and proceed down the steps or escalator to the first floor.
10. …

You can see how such a set of directions might be overwhelming in their scope. In fact, they would be so overwhelming as to be confusing.

> **Note**
>
> Why are down escalators called escalators? Since they go down, shouldn't they be called descalators?

The way travelers really navigate is in stages, or segments. A broad set of directions is given (board flight 123, which will take you to Chicago; then flight 456, which will take you to San Jose; rent a car; and drive to the hotel). At each of these steps, you assume there will be directions available locally to take you between any two points. For instance, you assume there will be signs on the local highway, or some software or map you can consult to provide you with directions from your home to the local airport, and then there will be signs within the airport where you are connecting between flights to guide you between the gates, etc.

This process of taking a trip in stages is, in reality, a form of abstraction. You know, when you travel, that information will become available as you proceed through the trip, and hence you do not need it *right now*. What you need is enough information to get you into a general area and then access to more detailed information when you get there.

This is precisely how aggregation in network protocols works. Aggregation removes more specific information about a particular destination as topological distance is covered in the network. Figure 19-7 illustrates.

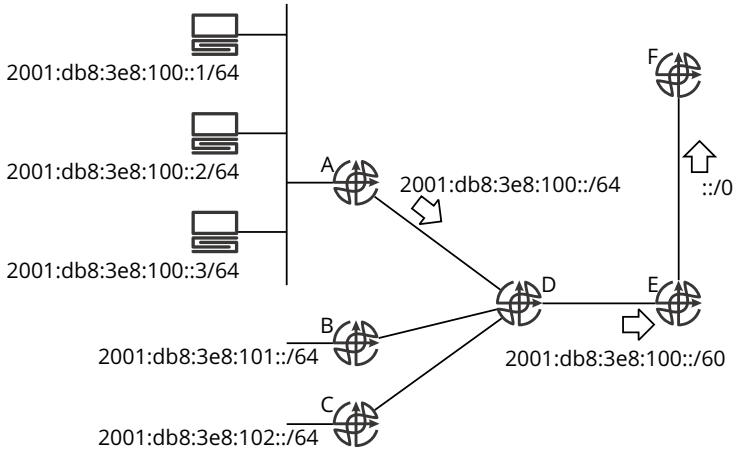


Figure 19-7 *Aggregation of reachability information*

In Figure 19-7, there are three hosts connected to a single shared link (broadcast domain) attached to an interface on A. Each of these hosts has its own physical Media Access Control (MAC) address, which is related to an Internet Protocol (IP) address, which has been assigned either manually or through the Dynamic Host Configuration Protocol (DHCP). These addresses all fall within a single /64 range of addresses. A aggregates these host addresses into a single advertisement, traditionally considered the address of the "wire" in IP networks: 2001:db8:3e8:100::/64.

Two other routers, B and C, are advertising two other /64s; the three /64s advertised by A, B, and C fall within the same /60 address range. Router D is configured to aggregate these three /64s to the /60. E, in turn, advertises a default route (::/0) to F, which means "any IP address you do not know about, you can reach through me." This is an aggregate sitting "above" 2001:db8:3e8:100::/60. Some useful terminology:

- **Supernet or aggregate:** An address that covers, or represents, a set of longer prefix, or more specific, destinations
- **Subnet:** An address that is covered, or represented by, a longer prefix, or less specific, destination in the routing table

Subnets and aggregates look identical in the routing table of any individual device. The only way you can see if a particular route is either a supernet or subnet is if the longer and shorter routes both exist in the routing table of the aggregating device at the same time. Without the subnet, you cannot tell whether a route is an aggregate or not.

A, in advertising 2001:db8:3e8:100::/64, does not *remove* any reachability from the network; rather it *adds* unreachable destinations that appear to be reachable to the control plane. Router A is advertising reachability to a large number of hosts, such as 2001:db8:3e8:100:4//64, even though this host doesn't exist. In the same way, D is advertising unreachable address space into the network by advertising 2001:db8:3e8:100::/60, and E is advertising unreachable address space into the network by advertising ::/0.

Packets transmitted to a nonexistent host are normally just dropped by the first device with specific enough routing information to know the host doesn't exist. For instance:

- If a packet is forwarded by F toward E with a destination address of 2001:db8:3e8:110::1, E can drop this packet, as this destination does not fall within any of the available destinations in E's routing table.

- If a packet is forwarded by F toward E with a destination address of 2001:db8:3e8:103::1, D can drop the packet, as this destination does not fall within any of the available destinations in D's routing table.
- If a packet is forwarded by F toward E with a destination address of 2001:db8:3e8:100::100, A would need to drop the packet, as this destination is not in the local Address Resolution Protocol (ARP) cache at A's connection to 2001:db8:3e8:100::/64.

There is another place where aggregation can be configured in a network: between the routing table (Routing Information Base, or RIB) and the forwarding table (Forwarding Information Base, or FIB), within an individual network device. This type of aggregation is fairly unusual; it is primarily used in situations where a device's forwarding table is restricted to a particular size because of memory limitations.

Filtering Reachability Information

Filtering reachability information, unlike aggregation, does remove reachability information from the control plane; hence filtering is normally used as an aid or part of a layered defense for network security. Figure 19-8 is used to illustrate.

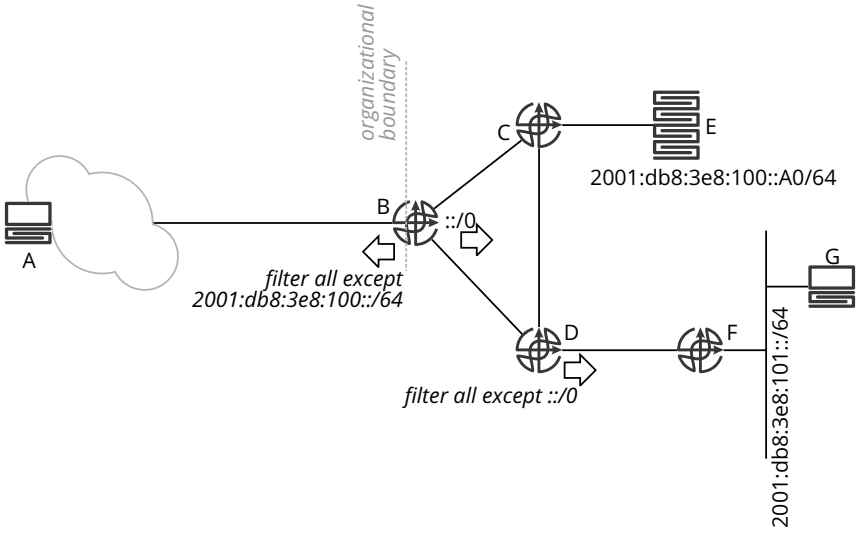


Figure 19-8 *Route filtering*

In Figure 19-8, A should be able to reach E within the organization (to the right of the organizational boundary line) and no destinations outside the organization. Host A definitely should not be able to reach G, for instance, or any of the transit links or routers within the organization's network. There are several ways to accomplish this, of course. The network administrator could place a stateful packet filter at the edge of the network to block traffic that is not part of a session originating from inside the network, or the network administrator could configure a packet filter to block A from accessing any destination other than E. While these are, of course, good ideas, it is often best to combine such filters with some control plane filter to prevent any routers in the network that A is attached to (within the cloud) from learning about these destinations. To accomplish this, the network administrator can place a filter at B blocking the advertisement of any reachable destination within the network other than the subnet that E is attached to.

At D, all routes are also filtered toward F—except the default route. While this is configured as a route filter on D, it acts like route aggregation; the default still allows G to reach E, even though F does not have a specific route, by following the default route. It is important to differentiate between the two cases: a route filter being used like aggregation and a route filter being used to prevent or block reachability to or from a particular device (or set of devices).

Layering Control Planes

In Chapter 9, "Network Virtualization," the case for building virtual topologies was laid out from the perspective of the data plane: primarily to provide traffic separation, reachability separation, and to provide "over the top" network services, particularly encryption and tunneled protocol support. There is an entirely separate case to be made for layering control planes, either with virtualized topologies, or without. Consider the security example set out previously in Figure 19-8; another way to solve the same problem might be to provision an overlay network, as shown in Figure 19-9.

In Figure 19-9, A needs to access H and K, but not M; N needs to access all three. Router B is a smaller device, perhaps a small home office router, which can support just a handful of routes. It is possible, of course, to filter routing information at C such that B has just the one or two routes it needs, but this may not be scalable from a network management perspective. Nor does this provide traffic separation, which is a requirement in many places where overlay networks are used. Meeting any traffic separation requirements would necessitate building packet filters at every device along the path, adding further to the network management load.

A better option, in many cases, is to create a virtual overlay network including just the devices that need to communicate. In this case, the dashed gray lines represent

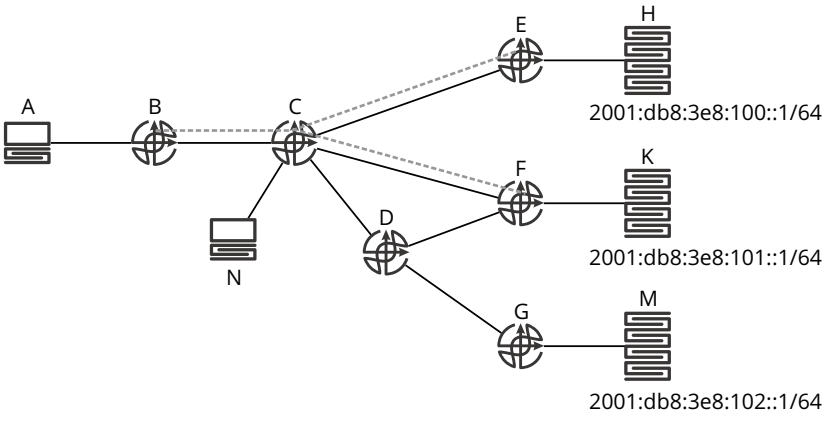


Figure 19-9 *An overlay as control plane information hiding*

the virtual overlay network created to fulfill the requirements given. From an information hiding perspective, what is important to note is the following:

- B does not need to know about D or G, the links connecting them, nor the 2001:db8:3e8:102::/64 subnet; information about these topology elements and reachable destinations are hidden from the control plane at B by building a tunnel, or virtual topology, with one end at B and the other ends at E and F.
- The second control plane can run as a different process on C, E, and F; this second control plane also does not need to know about these topology elements or reachable destinations.

Some information about topology and reachability, then, is hidden from B entirely, and some processes on C, E, and F, without reducing the required reachability. To connect this back to the concept of failure domains, routers that do not know about specific topology elements and/or reachable destinations do not need to recalculate the set of loop-free paths through the network when those (hidden) elements change. Because of this, B can be said to be in a different failure domain than D and G. Virtualization, then, can often be treated as another form of information hiding.

Caching

Caching begins with a simple observation: not all forwarding information is used all the time. Rather, particular flows pass along particular paths in a network, and particular pairs of devices (typically) only communicate for short periods of time. Storing forwarding information for short-lived flows, and in devices far off the path any particular flow might use, is a waste of resources. Figure 19-10 is used to illustrate.

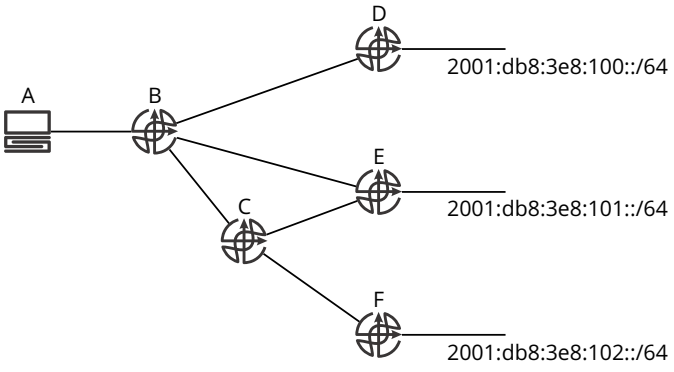


Figure 19-10 *Considering why caching works*

In Figure 19-10, the path from A to 2001:db8:3e8:100::/64 does not pass through C, E, or F; if A is the only device that ever originates paths toward this destination, it is a waste of memory and processing power for C, E, and F to calculate shortest paths to the 100::/64 destination. But how would E know no host attached to 101::/64 is going to send traffic to some device connected to 100::/64? There is no way, from a control plane perspective, to know this.

Instead, E must rely on traffic as it passes through the network. For instance, E could calculate a route toward 100::/64 when some packet is transmitted from a locally attached host toward some destination on the 100::/64 subnet. This is a reactive control plane. Caching is not restricted to reactive control planes, however. It is possible for E to calculate a loop-free route to 100::/64, but to not install this information into its local FIB. This is another form of FIB compression, which can be used when the size of the RIB is not limited, but the size of the FIB is (for instance, when there is a limited hardware forwarding table). FIB compression was once quite common in network devices but has generally fallen out of favor as the cost of memory has decreased and other techniques to store more forwarding information in smaller amounts of memory have been developed and deployed.

Note

There are also bad memories in the culture of network engineering around RIB to FIB caching schemes; in the late 1990s, many provider networks failed due to these schemes, so many network engineers avoid such schemes—and often rightly so. There are many interesting and unpredictable failure modes in RIB to FIB caching schemes, beyond those found in "normal" caching schemes.

The key question in any caching scheme is: how long should the cached information be held? There are at least two answers to this question:

- Remove a cache entry some specific time after it has been installed, or some specific time after its last use to forward a packet; this is timer based.
- Remove the oldest or most specific cache entries when the cache reaches some percentage of its capacity; this is capacity based.

Normally these are combined, with the first being the "normal" process for removing stale cache information, and the second used as a "safety valve" to prevent the cache from overflowing. Caches normally rely on the number of forwarding table entries *in use* being some small percentage of the reachable destinations. Generally, the rule of thumb is somewhere around 80/20—80% of the traffic will be directed at 20% of the destinations, or, in other situations, about 20% of the total reachable destinations will need to be stored at any given time.

There are a number of problems designers face when caching forwarding information in this way. Figure 19-11 is used to illustrate one interesting failure mode.

In Figure 19-11, E has 100 hosts attached; at the same time, C and D can support 70 entries in their forwarding table and will start removing items from cache when their forwarding table is 80% full (so when the cache reaches 56 entries, the caching algorithm begins removing the oldest entries to bring the cache under some number of total entries, say 50 for the purposes of this example). Assume caching is taking place at the individual destination IP address level, rather than at the subnet level (the reason for this will be explained in a following example). The situation that caching solutions normally assume is that A will communicate with a limited number of the 100 possible destinations at once. If A builds sessions with 20 of these destination devices for one minute, then another 20 the next minute, and so on, the cache can be "tuned" to carry information about any particular reachable destination for just a few seconds after its last use.

The worst possible case, from a caching perspective, is that A attempts to communicate with all 100 reachable hosts at once, or the cache timers are set long enough to

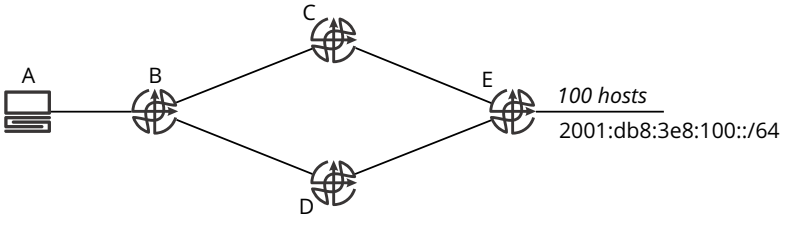


Figure 19-11 *An interesting cache failure mode*

cause every one of these destinations to remain in the cache at all times. Two problems are going to develop in this case. *First*, the cache at B is going to overflow. When B receives a packet that triggers caching of the 57th destination, it will begin removing older cache entries in order to protect the cache from failing entirely. The flow dependent on the removed cache entries will, of course, continue sending packets (or perhaps reset, and begin sending packets again), again causing the cache to reach the 57th entry, and hence the oldest entries to be removed again. This is a straightforward problem, easily detected, even if it is not easily mitigated.

Second, the caches at C and D are likely to develop problems. It is possible to build a stable system if B splits the load perfectly between C and D. However, this is rarely going to happen in real life. Instead, what is likely to happen at A is, at best, a 60/40 split; so traffic sent by B toward 40 of the destinations is sent to C, while traffic sent by A toward the other 60 destinations is sent toward D. The result is the cache on D overflows (there would need to be 60 cache entries, which is more than the 56 allowed by the caching algorithm), causing D to start removing cache entries. The removal of this caching information will cause the session to reset, as well.

The cache churn at B, C, and D can easily develop into a positive feedback loop, where dropped packets and sessions cause a refactoring of where traffic flows in the network, in turn causing different caches to overflow, in turn (again) causing dropped packets and session resets. There are few ways to resolve this sort of problem other than the obvious ones: increase the cache size, or reduce the number of concurrent flows through the network.

One apparently obvious answer—caching to the subnet level, rather than individual hosts—will not work. Figure 19-12 is used to explain why this will not work.

Figure 19-12 shows two networks: one (the upper) labeled *before* and the other (the lower) labeled *after*. Assume B, C, D, and E cache to the subnet of the destination, rather than the individual host information. What happens in this network is

- A sends a packet to 2001:db8:3e8:101::1.
- B receives this packet and discovers (through some mechanism—it does not matter what this mechanism is) that the destination is reachable through C and D.
- B determines (perhaps based on load sharing) that the traffic should travel through C; it builds a cache entry toward 2001:db8:3e8:100::/60 through C in its local forwarding table.
- A now sends a packet to 2001:db8:3e8:100::1.
- B forwards this traffic along the path toward 100::/60, so the traffic is sent to C, then forwarded to E, where it is dropped.

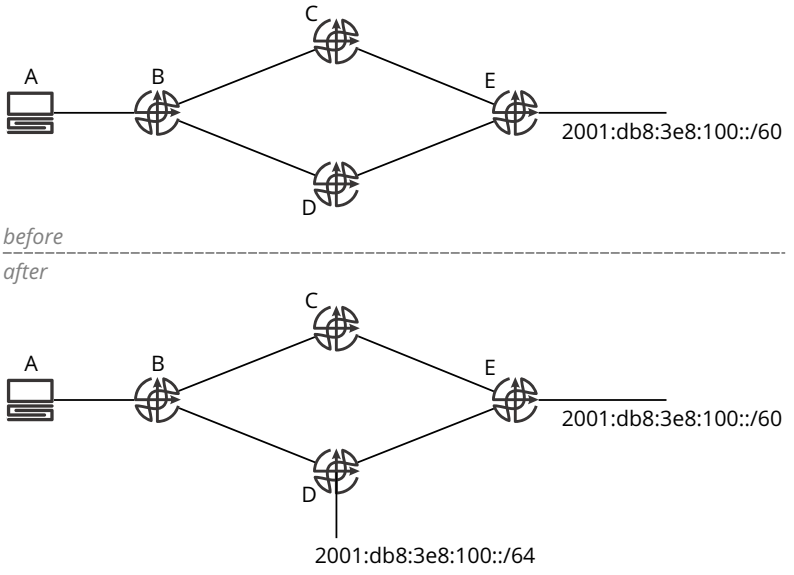


Figure 19-12 *Caching to the subnet level*

Why does E drop this traffic? The packet destined to 100::1 "lives" in two different network address spaces: the 100::/60 and the 100::/64. E knows about the 100::/60 address space, so it should know about every reachable destination in this space. Because E believes it knows about every destination in this address space, there is no reason for E to ask any of its neighbors about 100:1; it should already know about this specific destination. This destination, however, is connected to D, so there is no way for E to have 100::1 in its local forwarding table. In effect, E believes it *knows* 100::1, as an individual host, *does not exist*, so it will drop any traffic destined to this address.

Because of this, A has no effective way to reach any device attached to 100::/64 network; it might be that when (or if) the cache entry times out at B, the next packet will happen to be for a destination within the 100::/64 network, causing the correct set of cache entries to be built at B. Whether or not this is likely to happen, it is never a good thing for control planes to have possible states, such as this one, where reachability is variable or unpredictable.

There are a number of ways this problem could be fixed, none of which appear to be deployable in the real world. For instance, you could dictate that *every prefix in the network must have the same prefix length*, but this would rule out aggregation, which is problematic.

Note

There is another reason to build cache entries at the per device (or per address) level—to improve load sharing. Consider the example shown in Figure 19-11; if B built its cache at the per subnet level, then B would choose one path, either through C or through D, to send all the traffic in the network. The other path would remain unused (at least until B's cache entry timed out, at which point the used and unused paths might switch). Caching at the subnet level can cause a large set of network resources to go unused; generally this is considered a result to be avoided.

Slowing Down

Everyone in the modern world should know the value of slowing down sometimes—it can reduce information overload. It is no different for a control plane; slowing down the pace at which information is presented to a device does not really reduce the processing and memory requirements so much as spread them out over time. Another point in favor of slowing down state velocity is that it can allow multiple state changes to be "gathered," or "bunched," into a single processing cycle. Figure 19-13 illustrates these concepts.

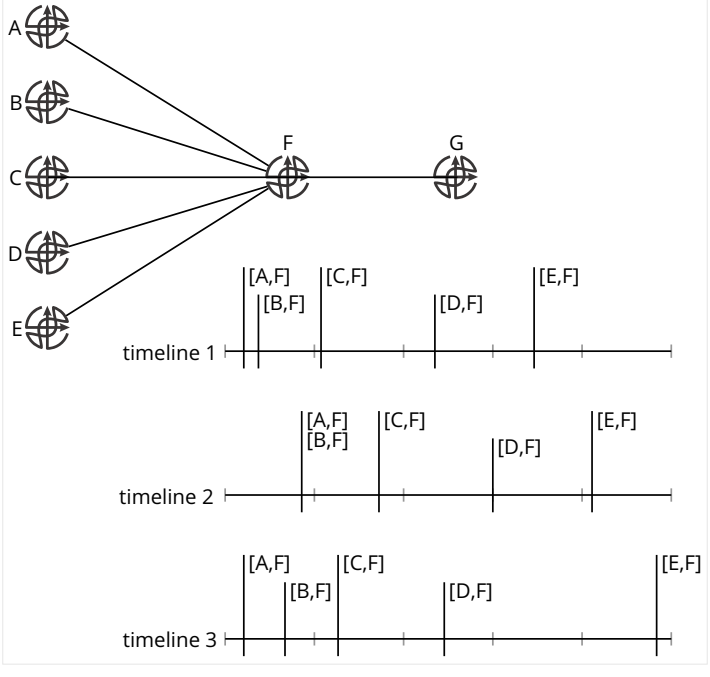


Figure 19-13 *Examples of slowing down state velocity*

In Figure 19-13, timeline 1 illustrates the actual order in which the links between F and another router fail; [A,F] and [B,F] fail relatively close to one another, and the remaining links fail a bit farther apart (or spread out in time). In timeline 2, F waits to advertise the control plane state change for a fixed amount of time. Because of this delay between the event occurring and reporting the event, the failures of the [A,F] and [B,F] links are reported at the same time, or in the same update. This allows G to process both events at the same time, which (should) require less processor and memory resources.

Finally, in timeline 3, an exponential backoff timer is shown. Essentially, the first time an event occurs, a timer is set, and the event is reported after the timer has expired. In timeline 3, this timer is set to 0 seconds, so the event is reported immediately (a common configuration for exponential backoffs). Once the event has been reported, a separate timer is set that must expire (or wake up) before the next event can be reported. Each event occurring after this increases this timer exponentially, causing the reporting of events to be spread out over ever-increasing amounts of time.

Final Thoughts on Hiding Information

Hiding information has several positive effects:

- It breaks a network into failure domains by limiting the scope of devices that must react to any particular change in topology or reachability.
- It reduces the velocity and scope of control plane state, allowing network to scale to larger sizes while retaining network stability.
- It is a "hook" through which to implement policy, specifically in relation to network security.

It might seem hiding more state is always better, based on these advantages. However, as with all things in network engineering, the truth is closer to a tradeoff. *If you have not found the tradeoff, you have not looked hard enough*. In the case of information hiding, refer back to Chapter 1, "Fundamental Concepts," specifically the section on complexity, and the example given concerning stretch and route aggregation. A second instance of hiding state can be found in relation to microloops, which are explained in Chapter 13, "Unicast Loop-Free Paths (2)." The more you slow down the velocity of state, the longer such microloops will exist in the network.

Hiding state is, then, a useful tool in the hands of good designers, but it can also cause many problems by negatively impacting network performance.

Further Reading

Bollapragada, Vijay, Russ White, and Curtis Murphy. *Inside Cisco IOS Software Architecture*. Indianapolis, IN: Cisco Press, 2000.

Doyle, Jeff, and Jennifer DeHaven Carroll. *Routing TCP/IP, Volume 1*. 2nd edition. New Delhi, India: Cisco Press, 2005.

Stringfield, Nakia, Russ White, and Stacia McKee. *Cisco Express Forwarding*. 1st edition. Indianapolis, IN: Cisco Press, 2007.

Teixeira, Renata, Aman Shaikh, Timothy G. Griffin, and Jennifer Rexford. "Impact of Hot-Potato Routing Changes in IP Networks." *IEEE/ACM Transactions on Networking,* 16, no. 6 (December 2008): 1295–307. doi:10.1109/TNET.2008.919333.

White, Russ, and Denise Donohue. *The Art of Network Architecture: Business-Driven Design*. 1st edition. Indianapolis, IN: Cisco Press, 2014.

White, Russ, and Alvaro Retana. *IS-IS: Deployment in IP Networks*. 1st edition. Boston, MA: Addison-Wesley, 2003.

White, Russ, Alvaro Retana, and Don Slice. *Optimal Routing Design*. 1st edition. Indianapolis, IN: Cisco Press, 2005.

White, Russ, and Jeff Tantsura. *Navigating Network Complexity: Next-Generation Routing with SDN, Service Virtualization, and Service Chaining*. Indianapolis, IN: Addison-Wesley Professional, 2015.

White, Russell I., Steven Edward Moore, James L. Ng, and Alvaro Enrique Retana. United States Patent: 8121130—Determining an optimal route advertisement in a reactive routing environment. 8121130, issued February 21, 2012. http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=8,121,130.PN.&OS=PN/8,121,130&RS=PN/8,121,130.

———. United States Patent: 9191227—Determining a route advertisement in a reactive routing environment. 9191227, issued November 17, 2015. http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=9191227.PN.&OS=PN/9191227&RS=PN/9191227.

Review Questions

1. Describe how you can determine the scope and speed of control plane state.
2. Is it possible to cause a network failure through a negative feedback loop? If so, how?

3. Describe the difference between summarizing and aggregating control plane information to control state.
4. Consider the State/Optimization/Surface (SOS) model. How would you describe summarization and aggregation within this model?
5. Consider the State/Optimization/Surface model (SOS). How would you describe hiding information through layered control planes within this model?
6. What might be some limiting factors in the formation of a positive feedback loop in a network control plane?
7. Research classful Internet Protocol addressing. How might the supernet/subnet concepts fit more "neatly" into this kind of scheme than they do with classes addressing schemes?
8. Research the two patents listed in the "Further Reading" section around reactive control planes and the caching of reachability information. Describe the solution presented in these patents to the problem described in the text.
9. Describe the importance of breaking up a network into failure domains.
10. Describe the relationship between information hiding and failure domains.

Chapter 20

Examples of Information Hiding

Learning Objectives

After reading this chapter, you should understand:

- The mechanisms used to summarize and aggregate control plane information in Intermediate System to Intermediate System (IS-IS)
- The mechanisms used to summarize and aggregate control plane information in Open Shortest Path First (OSPF)
- The interaction between summarization, aggregation, and external routing information in Open Shortest Path First
- Aggregation in the Border Gateway Protocol (BGP)
- The Border Gateway Protocol as a reachability overlay
- Controller-driven segment routing
- Exponential backoff
- Link state flooding reduction

The preceding chapter considered the problems that information hiding is designed to prevent or resolve, including positive feedback loops, as part of an overall pattern of security, and to reduce the amount and velocity of state in a control plane. This chapter will provide several examples of information hiding deployed in networks and protocols. The first section here will describe summarization in

Intermediate System to Intermediate System (IS-IS) and Open Shortest Path First (OSPF); as described in earlier chapters, summarization here means the removal of topology information, rather than reachability information. The second section will describe aggregation in the Border Gateway Protocol (BGP), which originally included the interesting feature of preserving topology information within aggregated routes. An example of layering will be considered next, specifically external routes carried in BGP layered over internal routes carried in IS-IS. The final section will describe exponential backoff in some detail, as applied to BGP neighbor dampening and the distribution of control plane state and calculation of loop-free paths in IS-IS.

Summarizing Topology Information

When reading this section, remember that summarization is removing topology information to manage control plane state, and aggregation is removing reachability information to manage control plane state. This section will consider summarization in the context of link state protocols, as distance vector protocols remove topology information at every hop in the network—even the Enhanced Interior Gateway Routing Protocol (EIGRP), which keeps a small radius of topology information.

Intermediate System to Intermediate System

IS-IS is a link state protocol used in many large-scale networks, including transit providers and data center (cloud) fabrics. The amount and velocity of state carried in a link state control plane can overwhelm slower processors with smaller amounts of memory, such as might be used in lower-cost routers and switches, or devices that must fit into limited physical spaces. Consider the network illustrated in Figure 20-1.

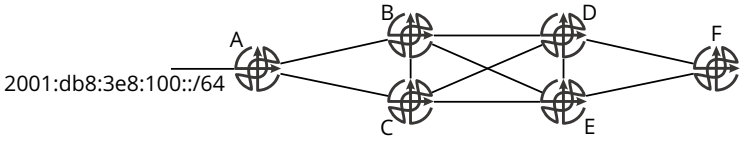


Figure 20-1 *Flooding copies in a link state protocol*

Note

In the early years of networking, routers were often shipped with much less computing power and storage than are available at the time of this writing. Even the routers deployed today for small office and home use can have more available resources than midrange routers used in early networks, and "low-end" routers deployed today often have more available resources than the most capable routers just a few years ago. It is always important to consider this when looking at protocol design and deployment, particularly in the area of summarization and aggregation.

In Figure 20-1, if the status of 100::/64 changes, D and E will receive three or four different copies of the Link State Packet (LSP) generated by A (depending on which LSP arrives at which intermediate system, or IS, when). Further, if the [A,B] link fails, F will receive an update about this topology change, even though it has no impact on E's ability to reach 100::/64. There are a number of ways to reduce the control plane state in this network; this section considers one method included in every link state protocol: flooding domains.

A flooding domain is a set of routers (intermediate systems in the case of IS-IS) with completely synchronized databases. When the flooding domain boundary is crossed, topology information will be summarized, and reachability information may be aggregated.

To understand flooding domains, it is best to start with a quick review of Open Systems Interconnect (OSI) addressing, which is used in IS-IS. Figure 20-2 will help illustrate this addressing scheme.

There are two primary sections of the OSI address to consider in Figure 20-2. The right three parts of the address between the dots are unique to each IS, and are generally calculated based on a local physical (Media Access Control, or MAC) address (as these are almost always designed to be unique for each physical interface or piece of hardware). The left sections, which are variable in length (although almost always used as shown in Internet Protocol, or IP, networks), are considered the "area ID" by the IS. Any two intermediate systems with the same information to the left of the right three dotted sections of their OSI address (called the area identifier, or area ID), are considered part of the same level 1 flooding domain, and will form a level 1 adjacency. Likewise, any two intermediate systems with different information in the left sections of their OSI addresses will form a *level 2* adjacency, and hence will be considered part of the level 2 flooding domain. There may be many different level 1 flooding domains in an IS-IS network, but there can be only one level 2 flooding domain.

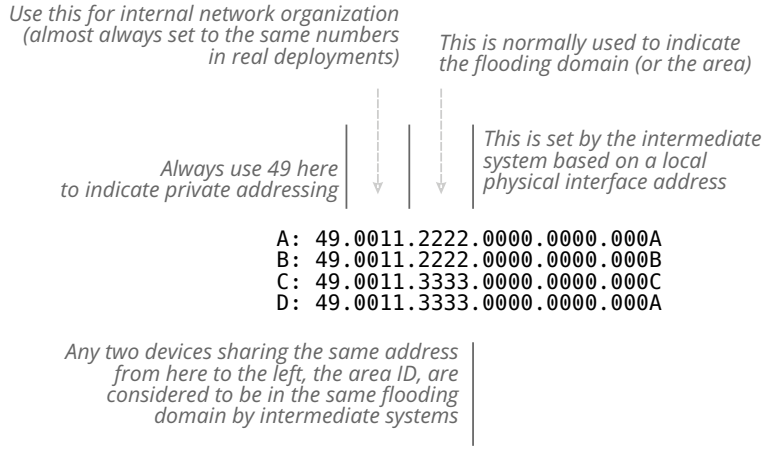


Figure 20-2 *OSI addressing*

Note

IS-IS forms level 2 adjacencies between each pair of intermediate systems, regardless of the area identifier. To simplify the explanation of flooding domains, however, this text will assume that neighbors capable of forming level 1 adjacencies will form *only* level 1 adjacencies. This will be revisited later in this chapter to provide more detail on this point.

Figure 20-3 illustrates.

In Figure 20-3, A has a different area ID than B and C; A has 49.0011.2222, while B and C have 49.0011.5555. Hence, [A,B] and [A,C] will be level 2 adjacencies. [B,C], [B,D], [C,E], [B,E], [C,D], [D,F], and [E,F] will all be level 1 adjacencies. Each adjacency type will exchange only the database associated with the correct adjacency level; Figure 20-4 illustrates.

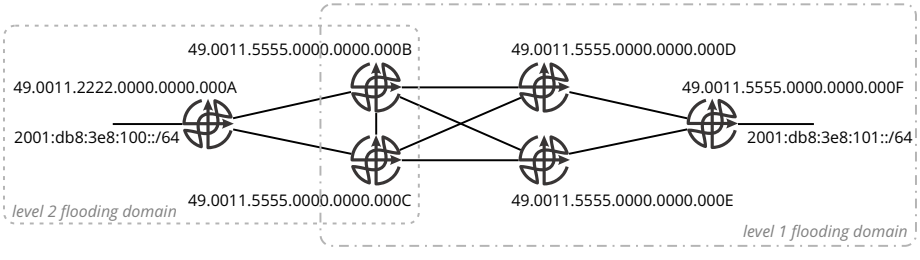


Figure 20-3 *Flooding domains in IS-IS*

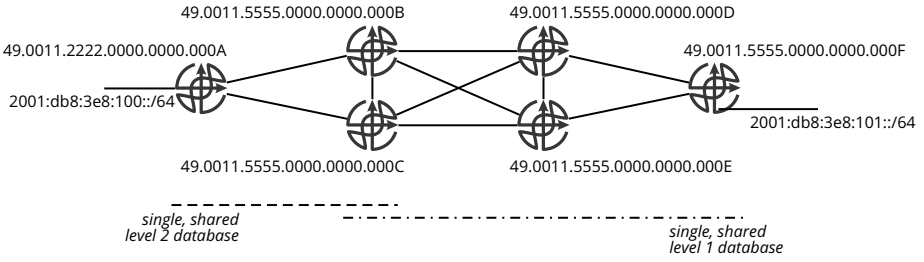


Figure 20-4 *Shared databases in IS-IS flooding domains*

In Figure 20-4, A, B, and C all have a synchronized (shared) level 2 database; B, C, D, E, and F all have a synchronized (shared) level 1 database.

Note

To be precise, every IS builds and maintains both a level 1 and a level 2 database. If you were to examine F, for instance, you would find it has a level 1 database containing information about every link, node (IS), and reachable destination in the 49.0011.5555 flooding domain. The level 2 flooding domain at F, however, would contain a single entry, for F itself. Why does the level 2 database at F have a single entry? F only builds level 1 adjacencies with D and E; it will not synchronize anything in its level 2 database across a level 1 adjacency. Hence, it builds a level 2 database but will not share the contents (synchronize) the contents of this database with any adjacent neighbors.

This arrangement certainly seems to reduce the scope of the reachability and topology information in the network; as the information about 100::/64 is in the level 2 flooding database at A, it will be shared just across level 2 adjacencies; hence only B and C will receive this information. But how can a host connected to F (at 101::/64, for instance) reach this destination? F does not receive a copy of the level 2 database, and therefore cannot know about 100::/64.

IS-IS solves this through the *attached bit*. B and C, because they are *attached* to the level 2 flooding domain (remember there can be only one level 2 flooding domain in an IS-IS network), will set the *attached bit* in their advertisements. This causes D, E, and F to create a default route in their local routing tables to point toward B and C. Traffic originating someplace on 101::/64, then, will be switched based on this default route at F toward either D or E, and then toward B or C from D or E. When the traffic reaches B or C, it will follow the specific route installed in the local routing table based on the information contained in the level 2 database toward the destination.

What about the return traffic? If 101::/64 is shared to B, C, D, E, and F through a level 1 flooding database, A cannot know about this destination. Hence, hosts attached to the 100::/64 network will not be able to send traffic toward a host on the 101::/64 network. IS-IS solves this problem through redistribution between the two databases. Any destinations in the level 1 flooding database are automatically redistributed into the level 2 flooding database as if they are attached to the redistributing IS. The cost to reach the destination from the redistribution point is preserved in the new route injected into level 2 to provide (some level of) optimal routing through the network.

What if you want to carry a more specific route toward 100::/64 into the level 1 flooding domain in the network shown in Figure 20-4? Most IS-IS implementations allow this through redistribution from the level 2 database into the level 1 database through route leaking. To prevent routing loops (consider what would happen if 100::/64 were redistributed from the level 2 database into the level 1 database and back again), routes redistributed from the level 2 flooding domain into a level 1 flooding domain have the down bit set; this means the route has been redistributed *down* the flooding domain hierarchy and should not be redistributed back up the hierarchical level structure.

IS-IS, then, both aggregates routing information and summarized topology information at a flooding domain boundary. It is possible to leak more specific reachability information through the aggregate (the attached bit, which causes a ::/0 route to be installed in the local routing table of each IS in the level 1 flooding domain), so route aggregation can be "undone," if the network designer decides it is important to do so.

An interesting point to note about IS-IS flooding domains is this: there are no "hard boundaries" of any kind between the level 1 and level 2 flooding domains. It is perfectly valid for every IS in a network to be a part of the level 2 flooding domain, as well as some level 1 flooding domain. Figure 20-5 illustrates a network in which some intermediate systems are in both level 1 and level 2 flooding domains.

Four flooding domains are illustrated in Figure 20-5. The first, 49.0011.1111, contains A, B, and C. The second, 49.0011.3333, contains D and F. The third, 49.0011.4444, contains G, H, M, and N. The fourth flooding domain is the overlaying level 2 flooding domain, which contains C, E, D, G, and F. The first interesting point here is all the intermediate systems in the level 2 flooding domain are also in a level 1 flooding domain with the exception of E. Each of the intermediate systems in both a level 1 and level 2 flooding domain have formed a level 2 adjacency with each of its connected neighbors in the level 2 flooding domain, and are

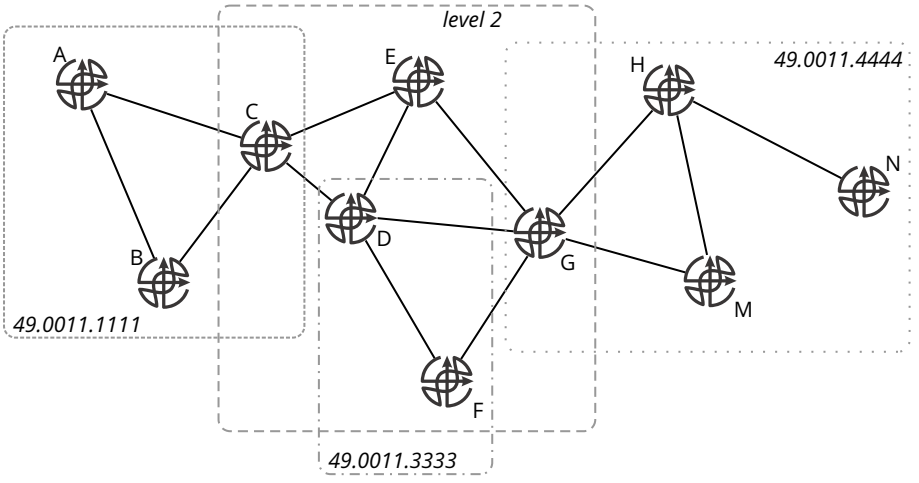


Figure 20-5 *Overlapping flooding domains in IS-IS*

synchronizing their level 2 database with their level 2 neighbors. For instance, C has four adjacencies:

- A, with which it is synchronizing only the level 1 database
- B, with which it is synchronizing only the level 1 database
- D, with which it is synchronizing only the level 2 database
- E, with which it is synchronizing only the level 2 database

The second interesting point is that D and F are in a level 1 flooding domain, 49.0011.3333, completely overlapped by the level 2 flooding domain. D and F have formed both a level 1 and level 2 adjacency across the link between them, and are synchronizing both the level 2 database and the 49.0011.3333 database. It is possible for two adjacent intermediate systems with the same area ID to form a level 2 adjacency; it is not possible for two adjacent intermediate systems with different area IDs to form a level 1 adjacency.

Open Shortest Path First

OSPF is also a link state protocol, and hence also subject to the same sorts of limitations as IS-IS; a rapidly changing topology can sometimes overwhelm slower

processors with smaller amounts of memory. To prevent this, network designers can break up the flooding domains in an OSPF network into *areas*. OSPF areas are implemented differently from the flooding domains in IS-IS; Figure 20-6 illustrates.

While some of the mechanisms are similar, there are some important differences between the two.

First, OSPF areas cannot overlap; Area Border Routers (ABRs) connect two flooding domains (or areas) together and have two databases (one per area). Every other router in the network has one link state database (LSDB), which contains reachability and topology information. The outlying area IDs can identify which area a particular router is in; *area 0* acts as a centralized area connecting all of the outlying areas together.

Second, OSPF summarizes by default, but it does not aggregate by default. OSPF carries information in a series of Link State Advertisement (LSA) types, each type carrying a different kind of information. The most common types are

- **Router:** Information about the originating router, connected neighbors, and connected destinations
- **Network LSA:** A pseudonode
- **Inter-Area Prefix (or *summary*) LSA:** Summarized reachability information
- **Inter-Area Router LSA:** Information about the originating router
- **AS-External LSA:** External reachability information
- **AS-External Not-So-Stubby Area (NSSA) LSA:** External reachability information

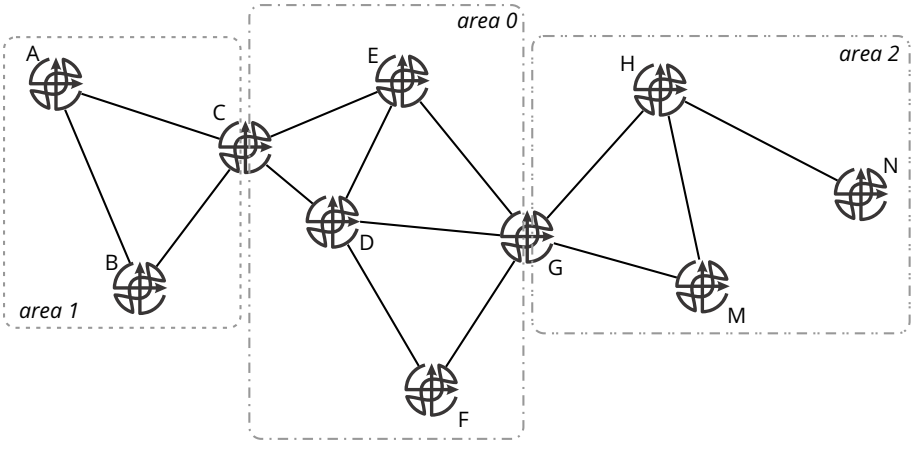


Figure 20-6 *Flooding domains in OSPF*

Many of these LSA types are used to carry information between flooding domains; Figure 20-7 illustrates.

The set of LSAs carried between two areas depends on the *type* of the outlying (nonarea 0) area. Several of these are described in the following sections.

Normal Area

An Area Border Router (ABR) sitting between a normal area and area 0 will summarize topology information by default, but not aggregate routing information. It is best to begin with the information B knows about area 1, and then examine what information B would send toward A, in area 0. In this network, B would have in its area 1 LSDB:

- An AS-external LSA for 100::/64 originated by D
- A network LSA (pseudonode) for the [C,D] broadcast link, which would include a connection to C, D, and 101::/64
- A router LSA from D with a connection to the [C,D] network LSA (pseudonode), a connection to C, and a connection to 101::/64
- A router LSA from C with a connection to the [C,D] network LSA (pseudonode), a connection to C, a connection to 101::/64, a connection to B, and a connection to 102::/64
- A router LSA from B with a connection to C and a connection to 102::/64

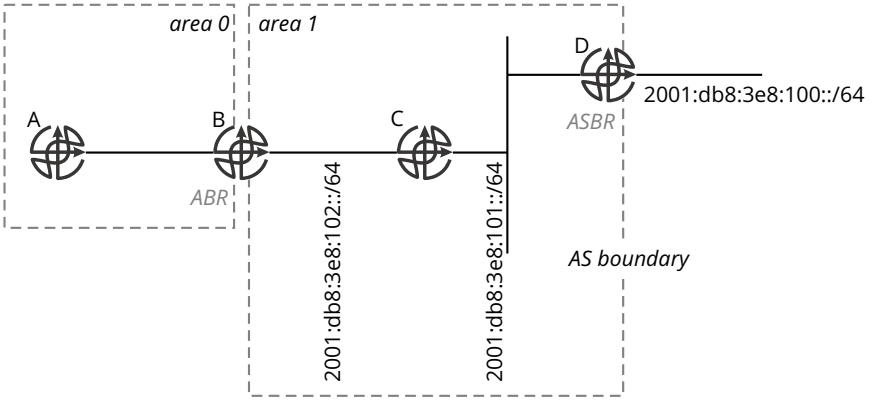


Figure 20-7 *OSPF Link State Advertisements between flooding domains*

If no aggregation is manually configured, A would receive the following information about area 1:

- An AS-external LSA for 100::/64 originated by D. External LSAs are not modified in any way by the ABR in a normal OSPF area.
- A summary LSA including 101::64 and 102::/64 originated at B. From the perspective of the Shortest Path Tree, these two routes will appear to be connected to B itself, with B's cost to reach each destination preserved in the summary LSA.

Route aggregation can be manually configured in OSPF implementations; for instance, the 100::/64, 101::/64, and 102::/64 reachable destinations in OSPF area 1 in Figure 20-7 could be aggregated into a 100::/60. In this case, B would advertise a single route in its summary LSA toward A, 100::/60.

Stub Area

OSPF stub areas are designed to support outlying areas with no external reachability; all topology and reachability information is internal to OSPF. Because of this, routers in OSPF stub areas are not allowed to carry redistributed routing information into the outlying area; likewise, external routing information from the rest of the network is not carried into the stub area. Because of this, D would not be able to redistribute the 100::/64 route into area 1. In this network, B would have in its area 1 LSDB:

- A network LSA (pseudonode) for the [C,D] broadcast link, which would include a connection to C, D, and 101::/64
- A router LSA from D with a connection to the [C,D] network LSA (pseudonode), a connection to C, and a connection to 101::/64
- A router LSA from C with a connection to the [C,D] network LSA (pseudonode), a connection to C, a connection to 101::/64, a connection to B, and a connection to 102::/64
- A router LSA from B with a connection to C and a connection to 102::/64

If no aggregation is manually configured, A would receive the following information about area 1:

A summary LSA including 101::/64 and 102::/64 originated at B. From the perspective of the Shortest Path Tree, these two routes will appear to be connected to B itself, with B's cost to reach each destination preserved in the summary LSA.

The ABR, B in Figure 20-7, will also transmit a default route (::/0) into the outlying flooding domain (area 1, in this case), so C and D can reach any external destinations connected to other areas in the network. B, C, and D would still know about the [A,B] link, as this is internal routing information.

Totally Stubby Area

If an area is configured as a totally stubby area, routers within the area cannot originate (redistribute) external routing information into OSPF, and external routes are not carried into the area. In this case, the LSDB at B would be the same as in the stub area case, and the LSAs advertised by B toward A would also be the same as the stub area case. The primary difference between a stub area and a totally stubby area is the handling of reachability information *into* area 1 from area 0. In a totally stubby area, the ABR (B, in the network in Figure 20-7) will generate a summary LSA into area 1 with just a default route (::/0). C and D would have no knowledge of the topology or the reachable destinations beyond area 0, such as the existence of A or the [A,B] link.

Not-so-Stubby Area

Routers in an NSSA can redistribute routing information into the network from other sources (such as another routing protocol, or statically configured routes), but external routes from other OSPF routers (in area 0) are blocked at the ABR. Because AS-external LSAs are not allowed within the flooding domain, a special kind of LSA is used instead: the AS-external NSSA LSA (a type 7 LSA in OSPFv4). In Figure 20-7, B would have the following LSDB entries if area 1 is configured as an NSSA:

- A network LSA (pseudonode) for the [C,D] broadcast link, which would include a connection to C, D, and 101::/64
- A router LSA from D with a connection to the [C,D] network LSA (pseudonode), a connection to C, and a connection to 101::/64
- A router LSA from C with a connection to the [C,D] network LSA (pseudonode), a connection to C, a connection to 101::/64, a connection to B, and a connection to 102::/64
- A router LSA from B with a connection to C and a connection to 102::/64
- An AS-external NSSA LSA from D carrying 100::/64

The special NSSA AS-external cannot be leaked outside the area, so the ABR must translate it into a standard AS-external LSA before sending it into area 0. Given this

translation, if no aggregation is manually configured, A would receive the following information about area 1:

- An AS-external LSA for 100::/64 originated by D. External LSAs are not modified in any way by the ABR in a normal OSPF area.
- A summary LSA including 101::64 and 102::/64 originated at B. From the perspective of the Shortest Path Tree, these two routes will appear to be connected to B itself, with B's cost to reach each destination preserved in the summary LSA.

Totally Not-so-Stubby Area

The totally not-so-stubby area (totally NSSA) is

- Similar to the totally stubby area because the ABR just sends a single summary LSA containing a default route (::/0) into the outlying area
- Similar to the not-so-stubby area (NSSA) because routers within the area can originate external routes using the AS-external NSSA LSA, which the ABR translates into an AS-external LSA, which is then transmitted into area 0

The Inter-area Router LSA

It is possible, in the right situation, for the summarization of topology information to cause a router in area 0 to choose a less than optimal path to external destination. Figure 20-8 illustrates.

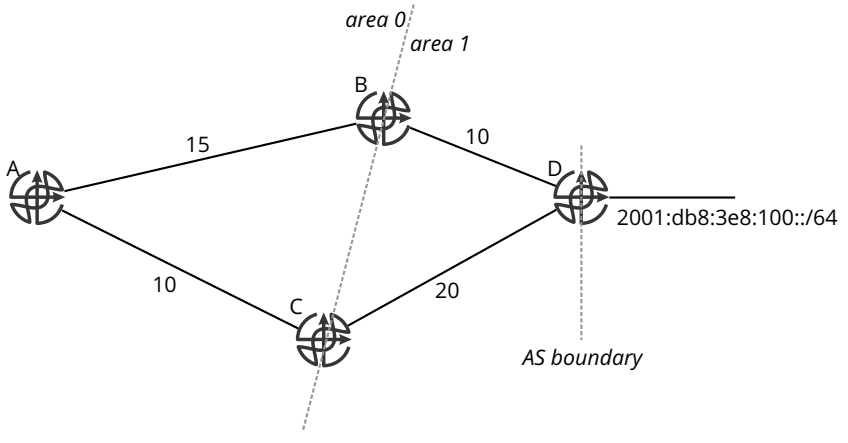


Figure 20-8 *Suboptimal routing with OSPF external routes*

In Figure 20-8, if A just has a single AS-external route toward 100::/64, it will choose the closest ABR connected to area 1 (the outlying area that the route is being redistributed into). In this case, A would choose C, sending all traffic toward the 100::/64 destination along a path with a total cost of 30. There is a path with a total cost of 25 available, but A does not "know" about this path, as the internal topology of area 1 is hidden through the OSPF summarization process at the ABRs.

To resolve this problem, OSPF ABRs will generate an inter-area router LSA for each ASBR (or each router which is redistributing reachable destinations into OSPF). The inter-area router LSA contains the ABR's cost to reach a particular ASBR. In this network, then, assuming area 1 is some sort of area that supports redistribution, A will have at least the following entries in its LSDB:

- An AS-external LSA for 2001:db8:3e8:100::/64 originated by D (this could potentially be a translated AS-external NSSA LSA, but A will not know the difference between these two possibilities)
- An inter-area router LSA generated by B with a cost of 10 to reach D
- An inter-area router LSA generated by C with a cost of 20 to reach D

Using this information, A can compare

- The cost through B toward 100::/64, by adding the cost to B to the cost from B to D, for a total cost of 25
- The cost through C toward 100::/64, by adding the cost to C to the cost from C to D, for a total cost of 30

The additional LSA provides the information A needs to choose the optimal path to the 100::/64 external destination, through B.

Note

This discussion around optimal routing and inter-area router LSAs should bring to mind the discussion in Chapter 1 around state, optimization, and surface. This is a specific instance where removing state from the control plane can result in suboptimal traffic flows, and where adding information back in is used as a technique to make traffic flows more optimal again.

Final Thoughts on OSPF Stub Areas

If you find the various area types confusing, you are not alone; network engineers struggle with remembering which area type permits what kind of information. If you

can remember three simple rules, however, you can easily figure out what sort of information should be where in any OSPF implementation:

- Stub means no externals are allowed in the area at all.
- Not-so means externals are allowed out of the area.
- Totally means no internals are allowed into the area.

Area types are designed to decrease the amount of information carried between flooding domains, reducing the amount of information any particular router in the network needs to store and process. Many OSPF implementations can also *filter* the information inserted into a summary LSA; this is often called type 3 filtering, even though the summary LSA may not be a type 3 in every version of OSPF.

Aggregation

Aggregation reduces the state in the network by combining multiple reachable destinations into a single destination. Most of the time, aggregation entails summarization, as seen in the example of routing information transiting flooding domains in IS-IS. This is true of OSPF, as well—in almost all cases, aggregating routing information involves summarizing topology information as well.

Is there any case where aggregation is used without summarization? There is an older feature in the Border Gateway Protocol (BGP), now generally deprecated and almost never really deployed, that does aggregate routes without discarding all of available topology information. Figure 20-9 is used to illustrate.

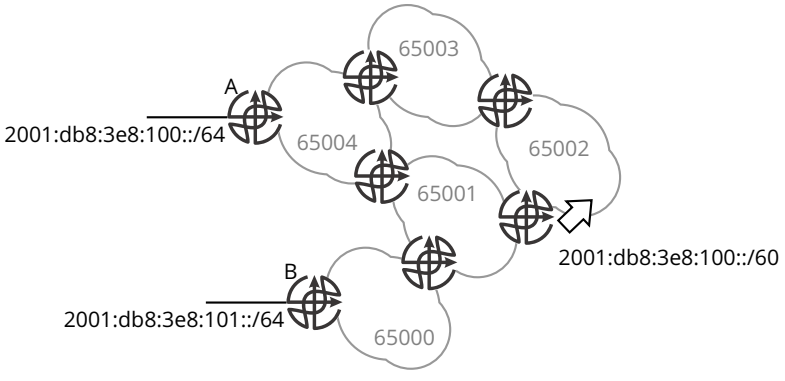


Figure 20-9 *The BGP atomic aggregate*

In Figure 20-9, a series of BGP Autonomous Systems (AS) have been connected in a ring. Assume the following:

- 100::/64 is advertised through the [65004,65001] boundary, toward AS65000 and AS65002.
- 100::/64 is filtered at the [65004,65003] boundary toward AS65003.
- The 100::/64 and 101::/64 routes are aggregated at the [65001,65002] boundary toward AS65002.

If some router in AS65004 prefers the aggregate over the longer prefix route within AS65004 (such as a local filter), it is possible a routing loop can form in this network. This type of situation can occur because BGP relies on the AS path to prevent loops across an internetwork. How can this problem be resolved? The most obvious solution would be to somehow include enough information about the AS path to prevent the 100::/60 route from being leaked back into any AS where a component of the aggregate is connected.

To prevent such loops from forming, BGP required any speaker aggregating routing information to include an atomic aggregate in the aggregate update. The atomic aggregate included the full list of every AS in the path of any of the component routes making up the aggregate. In this case, the 100::/60 aggregate advertised into AS60552 would have an AS path with one entry, 65001, but it would also contain an atomic aggregate containing [65004, 65000]. When the eBGP speaker between Autonomous Systems 65003 and 65004 receives the 100::/60 aggregate, it can examine the atomic aggregate attribute and determine at least some component of the aggregated route originated in AS65004. Hence, the eBGP speaker at the edge of AS65003 can reject the aggregate route, preventing the loop.

Note

There have been many proposals to remove the atomic aggregate from BGP; the most recent is Deprecate Atomic Aggregate.[1]

1. Hares, "Deprecate Atomic Aggregate."

Layering

While layering is not normally considered a form of information hiding by network engineers, it definitely does hide *full* information about the topology and reachability from some set of forwarding devices. Two examples—using BGP as an overlay to

carry external routing information and Segment Routing (SR) combined with a controller to produce a traffic engineering (TE) overlay—will be used to illustrate layering.

> **Note**
>
> It is impossible for an introductory-level book, such as this one, to give an in-depth overview of the many possible layering protocols and systems invented and deployed in networks. To give a small sample: Layer 3 virtual private networks (L3VPNs) based in MPLS and IP and IP tunnels, Layer 2 virtual private networks (L2VPNs), Ethernet VPNS (eVPNs), traffic-engineered overlays using Path Computation Element Protocol (PCEP), VXLAN (which has a native control plane, although the tunneling encapsulation is often used with a different control plane), 802.1q virtual local area networks (VLANs), Transparent Connection of Lots of Links (TRILL) VLANs, and SR. Covering these topics would require another entire book, and this one is quite large enough. Readers who would like to read more on these topics should look at the "Further Reading" section at the end of this chapter for more information.

The Border Gateway Protocol as a Reachability Overlay

The Border Gateway Protocol (BGP) was originally designed to carry inter-Autonomous System (inter-AS) information; it was explicitly *not* designed to carry reachability information within an AS. The basic design was to separate internal reachability (within the AS) from external reachability (outside the AS, or in the default free zone, or DFZ), in order to

- Prevent changes outside the network from impacting the operation of the network itself
- Allow different policies to be applied to internal and external routes

The first reason, to prevent changes external to the network from impacting the operation of the network itself, should be a familiar reason to hide information—to break up a network into multiple failure domains. Figure 20-10 illustrates.

In Figure 20-10:

- IS-IS is running on B, C, D, and E to provide reachability and topology information within the AS.

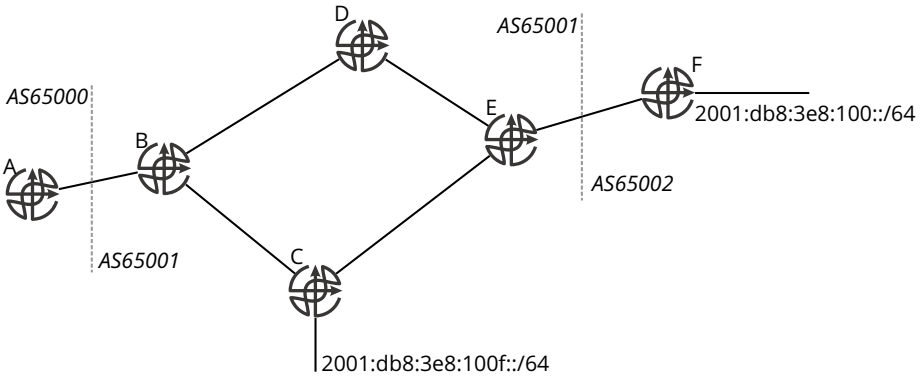


Figure 20-10 *BGP as an overlay*

- E and F are configured with an external BGP (eBGP) session.
- A and B are configured with an eBGP session.
- Each pair of [B,C], [B,D], [D,E], and [C,E] is configured with an interior (iBGP) session.
- C and D are acting as route reflectors for B and E.

Tracing the path of the route advertisement to 100::/64 through the network:

- F advertises 100::/64 to E over the eBGP session; the AS path is set to [65002].
- E advertises the 100::/64 route to D and C, which then reflect the route to B.
- B advertises the route to A over the eBGP session; the AS path is set to [65002,65001].

The Interior Gateway Protocol (IGP), which runs within the AS, does not need to carry the 100::/64 route at all. IS-IS carries just internal destinations, such as 100f::/64. Another way to put this is, IS-IS provides the internal reachability to allow BGP to form sessions through the AS, while BGP carries the reachability information allowing other Autonomous Systems to transit the local AS (to forward traffic from A, through AS65001, and on to F).

This separation of duties is a form of layering; BGP overlays IS-IS (the IGP), using the IGP to form adjacencies and discover paths within the AS along which it can forward traffic. IS-IS, on the other hand, does not need to know about any externally reachable destinations. How does this divide the network into two failure domains?

First, IS-IS (or any other IGP) is not impacted by changes to topology and reachability information external to the AS. If the link between F and 100::/64 changes, the IS-IS processes running on B, C, D, and E do not need to recalculate anything, as nothing in the network has changed from their perspective.

Second, peering Autonomous Systems are shielded from changes within AS65001. For instance, if the [C,E] link fails, the path has not changed from the perspective of A; the AS path remains the same, so BGP does not need to reconverge.

The fate of internal and external topology and reachability information is (at least to a large degree) separated from one another; hence the internal routing and the external routing are two different failure domains.

Note

There has been some research into how often, and under what circumstances, AS level policies, combined with route changes within an AS, will "leak" out to the rest of the DFZ.[2] These kinds of information leaks cross what should be a failure domain boundary, merging the failure domains at least in some small part. This is an example of leaky abstractions, which are discussed in Chapter 1.

2. Teixeira, et al., "Impact of Hot-Potato Routing Changes in IP Networks."

Segment Routing with a Controller Overlay

SR is perhaps the simplest possible use of Multiprotocol Label Switching (MPLS) short of manually configured point-to-point tunnels through a network. The general idea behind SR is to stack a set of labels at one edge of the network, so each device along the path can forward based on the outermost label exposed on the stack. As each device pops the outermost label off the stack, a new label is exposed describing the next hop in the path. Figure 20-11 is used for illustration.

Note

SR is described here at a very high level. There are many more details in the design, deployment, and operation of SR; please refer to the "Further Reading" section at the end of the chapter for good references on SR.

In Figure 20-11, A receives a packet destined to 100::/64. Based on IP routing, this packet will be forwarded across the lowest-cost path, along [A,B,D,E,F]. What if the network operator wants the packet to travel along the alternate path, [A,B,C,E,F]? It

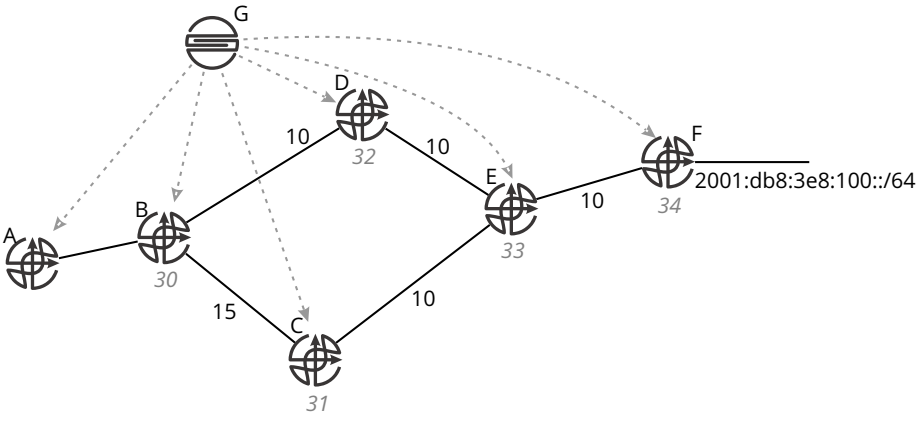


Figure 20-11 *Segment routing sample network*

is possible to modify the metrics along the paths, of course, but this would impact *all* the traffic entering the network at A and destined to 100::/64.

Several overlay technologies can solve this type of problem using a number of different control planes and a number of different encapsulations. If you multiply every possible encapsulation with every possible control plane, you will probably find there are more ways to solve this problem than can be explained in normal terms. Perhaps the real explanation is bored engineers who enjoy the challenge of solving the same problem in as many ways as possible. SR is a relatively simple way to solve this problem. If the operator has SR deployed on his network, he can

- Compute the path from A to F through [B,C,E].
- Discover the MPLS label for each device along the way; the resulting label stack would be [30,31,33,34].
- Impose this label stack on the packet while it is being switched at A.

Once this label stack has been imposed at A, the switching path would be

- A would forward the packet to label 30, which is B.
- When B receives this packet, it pops the outermost label on the stack; the stack is now [31,33,34].
- B will switch the packet toward 31, the outermost label on the stack, which is C.

- When C receives this packet, it pops the outermost label on the stack; the stack is now [33,34].
- C will switch the packet toward 33, which is E.
- When E receives this packet, it pops the outermost label on the stack; the stack is now [34].
- E will switch the packet toward 34, which is F.

Finally, F will pop the final label off the stack and forward the traffic based on the destination IP address. Where does the label that stack A imposes come from? There are, as always, a large number of possibilities, but in order to keep the example as simple as possible, assume there is a controller located someplace on the network, labeled as G in Figure 20-11.

This controller can participate in the routing protocol to discover the topology of the network and the reachable destinations (including what MPLS label has been assigned to each device). After combining this information with a set of policies, it can calculate the correct traffic-engineered path through the network and then signal A about what label stack to impose on this particular flow.

This kind of layering reduces state (hides information) by

- Allowing the traffic engineering policy to be pulled out of the distributed control plane, reducing the state in the distributed control plane considerably
- Removing the process of neighbor discovery and other distributed elements from the purview of the distributed control plane

Even if the policy controller fails, the network will still forward traffic, which means the controller has been placed into a different failure domain.

Slowing Down State Velocity

A third technique often used in protocols to hide information is to simply slow down the rate at which information is distributed through the network. Slowing down state velocity does not technically *hide* information in the permanent sense: it either allows network devices to "bunch up information" requiring shorter bursts of processing spaced farther apart, it allows network devices to take on information at a steady pace, or it removes duplicate copies of control plane state from the network. There are many different ways to reduce the velocity of state in a network control plane, but two examples are considered here: exponential backoff and flooding reduction.

Exponential Backoff

Exponential backoff is used in a wide variety of contexts, including

- In slowing down (*dampening*) the speed at which routes are propagated throughout an internetwork
- In slowing down (*dampening*) the speed at which interface state change is allowed to propagate in some network operating system implementations
- In slowing down the speed at which a link state protocol will compute a new Shortest Path Tree (SPT) on receiving new topology information
- In slowing down the speed at which routing information is distributed by a link state protocol in response to a change in the state of a link

This section will use running SPF in a link state protocol as an example, but you should keep in mind there are many places where exponential backoff can be used. To understand exponential backoff, several definitions will be needed:

- **Initial wait:** The amount of time the implementation will wait after receiving an event before processing it
- **Second wait:** Multiplied by an exponential offset to set the wait time on subsequent events
- **Max wait:** Used for two purposes:
  - The amount of time the implementation will wait after an event before setting the wait timer back to initial wait
  - The maximum amount of time the implementation will *ever* set the wait timer
- **Wait timer:** The amount of time the implementation will wait before processing the items currently in the processing queue

The exponential backoff process looks something like this in pseudocode:

```
// initial_wait == initial wait time
// second_wait == second wait time
// max_wait == max wait time
// next_wait == how long before the wait_timer expires
// begin by expiring the reset_timer
```

```
when reset_timer expires {
  stop wait_timer
  next_wait = initial_wait
  backoff = 1
}
when event_occurs {
  if wait_timer is running {
    next_wait = backoff * second_wait
    if next_wait > max_wait {
      next_wait = max_wait
    }
    backoff = backoff * 2
  } else {
    next_wait = initial_wait
  }
  set event to process at wait_time
  start reset_timer to expire in max_wait * 2
  start wait_timer to expire in next_wait
}
```

Figure 20-12 is used to explain exponential backoff in determining when to run SPF.

Assume some router is configured to use exponential backoff to reduce the amount of processing required for running SPF. Using Figure 20-12, the sequence of events might look like this:

1. The router begins with the *wait_time* set to *initial_wait*.
2. The router receives a new link state entry (whether an LSA or an LSP doesn't matter for this example; it could be OSPF or IS-IS).
3. The event is accepted, and
   a. A timer is set to *max_wait * 2*; this can be called the *reset_timer*.
   b. A timer is set for this specific event; once this timer expires, the event will be processed.
4. Before *reset_timer* expires, a second event is received.
   a. The *reset_timer* is restarted, so it will again expire in *max_wait * 2*.
   b. A timer is set for this specific event; once this timer expires, the event will be processed.
   c. The *wait_time* is set to *second_timer * 1*, as this is the second event.

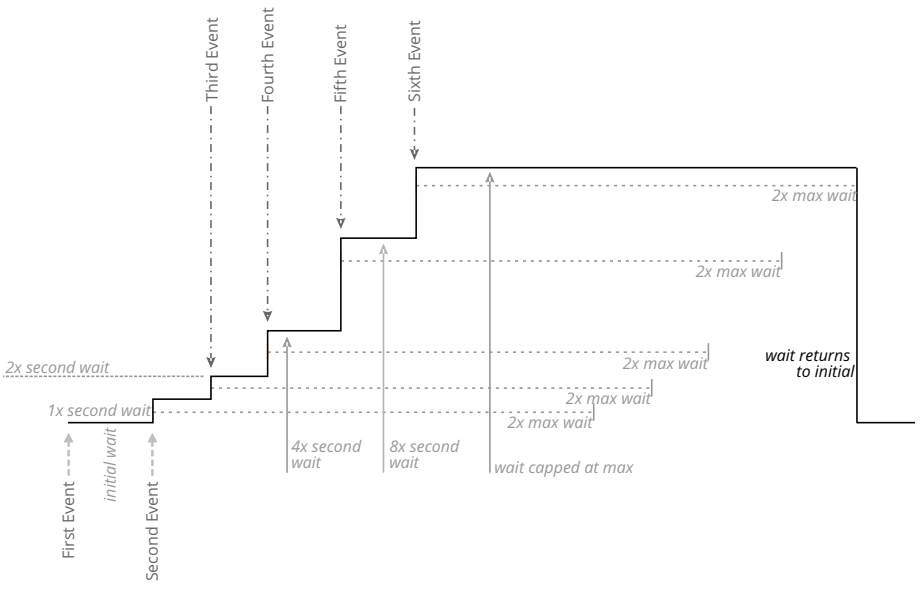


Figure 20-12 *Exponential backoff*

5. Before *reset_timer* expires, a second event is received.
   a. The *reset_timer* is restarted, so it will again expire in *max_wait* * 2.
   b. A timer is set for this specific event; once this timer expires, the event will be processed.
   c. The *wait_time* is set to *second_timer* * 2, as this is the third event.

The *wait_time* is set to the event number multiplied by 2 (this can be configured on some implementations), so the *wait_time* doubles with each event. If the *wait_time* ever reaches the *max_wait*, it will be capped at *max_wait*. And if the reset timer ever expires, which is *max_wait* * 2 from the last event, the entire system resets to its initial state. This process produces a *wait_time* as shown in Figure 20-12; the timer value increases exponentially until it reaches a cap. If no events happen for some long period, the entire system resets.

Why an exponential backoff? Because it allows the system to react quickly at first, but then to slow down its reaction time until the system is at the slowest acceptable speed. In the case of an SPF run, the first SPF run would occur quickly, but as more link state updates are received, the SPF runs are spread farther apart until some maximum is reached. This allows for fast reactions to individual events, while dampening the rate at which a large number of quickly occurring events is processed.

Link State Flooding Reduction

In deploying link state protocols onto highly meshed mobile networks, the amount of flooding required to converge can be a limiting factor. Figure 20-13 is used to illustrate.

In Figure 20-13:

- The columns and rows are marked instead of each individual router being marked; A1, for instance, is at the top-left corner, while D5 is at the lower-right corner.
- The tier numbers are marked on the right side; routers in T0 are Top of Rack (ToR) switches (or routers).

If some change occurs at A5, then

- A5 will flood a link state change to every router in row 4.
- Every router in row 4 will flood a link state change to A3.

A3, then, will receive four copies of the same link state change; in fact, every router in the fabric will receive at least four copies of the same link state change, and some will receive more copies. How can the number of copies be reduced in this topology? By building a view of the routers two hops away from any router that is flooding a change, and somehow signaling just one of them to reflood the link state

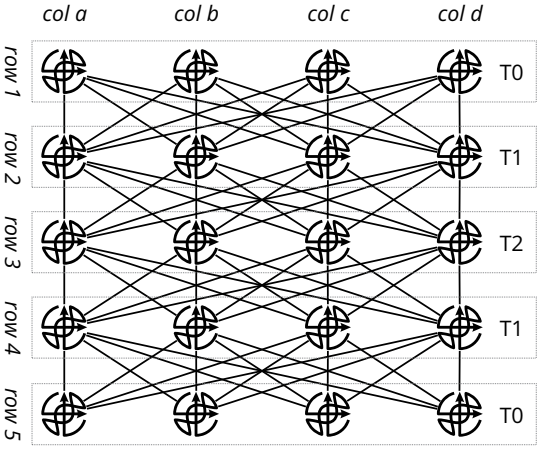


Figure 20-13 *A spine and leaf fabric as a flooding reduction example*

change to its neighbors. For instance, if A5 can discover that A4 can reach every router two hops away from A5 itself, A5 can send the link state change to B4-D4 formatted so they will not reflood it, while sending the update to A4 in a way that allows A4 to reflood the change.

But how can A5 determine which routers are two hops away? At least two methods have been devised and implemented:

- Each router can report its full set of neighbors to every neighbor, rather than just the neighbors on this link. For instance, A4 can report the set of neighbors [A3,B3,C3,D3,B5,C5,D5] to A5, rather than just [A5] (as it would normally do to verify two-way connectivity during adjacency formation).
- Once the initial adjacencies are formed, an SPF can be run at A5 that is restricted to two hops.

Once A5 discovers all of its neighbor's neighbors, it can build a minimum list of neighbors to flood to "cover" the entire set of two-hop neighbors. Given A4, B4, C4, and D4 *all have the same set of neighbors*, designating any of these neighbors as a "reflooder" will ensure the changes to the LSDB are synchronized through the network. Any neighbors on this list should receive changes in a way that allows them to reflood; neighbors not on this list should receive link state information in a way that does not allow them to reflood the changes. There are a number of ways a link state protocol can be modified to limit flooding scope in this way.

To outline the process in the network shown in Figure 20-13:

1. A5 discovers some change to the topology or reachability information.
2. A5 determines that A4, B4, C4, and D4 all have the same set of two-hop neighbors.
3. A5 selects one neighbor as a reflooder (or designated flooder); assume this is A4.
4. A5 floods to A4 normally.
5. A5 floods to B4, C4, and D4 using a mechanism that does not allow them to reflood the change.
6. A4 determines that A3, B3, C3, and D3 all have the same set of two-hop neighbors.
7. A4 selects one neighbor as a reflooder (or designated flooder); assume this is A3.

8. A4 floods to A3 normally.
9. A4 floods to B3, C3, and D3 using a mechanism that does not allow them to reflood the change.

There is a bit more to this technique than is outlined here; refer to the "Further Reading" section to find out more about this technique, as well as other flooding reduction mechanisms similar to this technique.

Final Thoughts on Failure Domains

The initial problem of summarizing information appears to be fairly simple; working from within the framework of a distance protocol in a simple network, it can be. In link state protocols, however, the ability to summarize without aggregation and the requirement for aggregation and summarization to take place at a specific place in the network make summarization and aggregation more difficult. Quite often, ideas and concepts are agglutinated, causing each idea to be difficult to understand on its own. Disentangling the ideas, however, to make them easier to understand can prove difficult as well. Adding in external routing information makes summarization and aggregation more complex in a link state protocol.

All of these concepts are extremely important for you to understand as a network engineer. Combining a base knowledge of how any given method of carrying control plane information, how each shortest path algorithm works on any given topology, and how and where information is aggregated and/or summarized can give you a quick read of how a network will work normally, as well in various failure situations.

This line of thinking provides a good segue to move from thinking about protocols to thinking about network operation.

Further Reading

Dearlove, Christopher, Thomas H. Clausen, Ulrich Herberg, and Philippe Jacquet. *The Optimized Link State Routing Protocol Version 2*. Request for Comments 7181. RFC Editor, 2014. doi:10.17487/rfc7181.

Ferguson, Dennis, Acee Lindem, and John Moy. *OSPF for IPv6*. Request for Comments 5340. RFC Editor, 2008. doi:10.17487/rfc5340.

Hares, Susan. "Deprecate Atomic Aggregate." Internet-Draft. Internet Engineering Task Force, March 2017. https://datatracker.ietf.org/doc/html/draft-hares-deprecate-atomic-aggregate-00.

"Intermediate System to Intermediate System Intra-Domain Routing Information Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service." Standard. Geneva, CH: International Organization for Standardization, 2002. http://standards.iso.org/ittf/PubliclyAvailableStandards/.

Jacquet, Philippe. *Optimized Link State Routing Protocol (OLSR)*. Request for Comments 3626. RFC Editor, 2003. doi:10.17487/rfc3626.

Katz, Dave. "OSPF and IS-IS: A Comparative Anatomy." Presented at the NANOG19, Albuquerque, NM, June 12, 2000. https://nanog.org/meetings/abstract?id=1084.

Moy, John. *OSPF Version 2*. Request for Comments. RFC Editor, April 1998. doi:10.17487/RFC2328.

Nguyen, Dang-Quan, Thomas H. Clausen, Philippe Jacquet, and Emmanuel Baccelli. *OSPF Multipoint Relay (MPR) Extension for Ad Hoc Networks*. Request for Comments 5449. RFC Editor, 2009. doi:10.17487/rfc5449.

Ogier, Richard G. *Use of OSPF-MDR in Single-Hop Broadcast Networks*. Request for Comments 7038. RFC Editor, 2013. doi:10.17487/rfc7038.

Ogier, Richard, and Phil Spagnolo. *Mobile Ad Hoc Network (MANET) Extension of OSPF Using Connected Dominating Set (CDS) Flooding*. Request for Comments 5614. RFC Editor, 2009. doi:10.17487/rfc5614.

Pelsser, Cristel, Randy Bush, Keyur Patel, Prodosh Mohapatra, and Olaf Maennel. *Making Route Flap Dampening Usable*. Request for Comments 7196. RFC Editor, 2014. doi:10.17487/RFC7196.

Przygienda, Tony, John Drake, and Alia Atlas. "RIFT: Routing in Fat Trees." Internet-Draft. Internet Engineering Task Force, January 2017. https://datatracker.ietf.org/doc/html/draft-przygienda-rift-01.

Rekhter, Yakov, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. Request for Comments 4271. RFC Editor, 2006. doi:10.17487/rfc4271.

Retana, Alvaro, and Stan Ratliff. *Use of the OSPF-MANET Interface in Single-Hop Broadcast Networks*. Request for Comments 7137. RFC Editor, 2014. doi:10.17487/rfc7137.

Roy, Abhay, Yi Yang, and Alvaro Retana. *Hiding Transit-Only Networks in OSPF*. Request for Comments 6860. RFC Editor, 2013. doi:10.17487/rfc6860.

Shen, Naiming, Les Ginsberg, and Sanjay Thyamagundalu. "IS-IS Routing for Spine-Leaf Topology." Internet-Draft. Internet Engineering Task Force, March 2017. https://datatracker.ietf.org/doc/html/draft-shen-isis-spine-leaf-ext-03.

Teixeira, Renata, et al., "Impact of Hot-Potato Routing Changes in IP Networks," *IEEE/ACM Transactions on Networking* 16, no. 6 (December 2008): 1295–307, doi:10.1109/TNET.2008.919333.

Wang, Lili, Zhaohui (Jeffrey) Zhang, and Nischal Sheth. *OSPF Hybrid Broadcast and Point-to-Multipoint Interface Type*. Request for Comments 6845. RFC Editor, 2013. doi:10.17487/rfc6845.

White, Russ. *Intermediate System to Intermediate System (IS-IS) Routing Protocol LiveLessons*. Video. LiveLessons. Cisco Press, 2016. http://www.ciscopress.com/store/intermediate-system-to-intermediate-system-is-is-routing-9780134465326?link=text&cmpid=2017_02_02_CP_RussWhiteVideo.

White, Russ, and Shawn Zandi. "IS-IS Support for Openfabric." Internet Draft. Internet Engineering Task Force, October 2017. https://datatracker.ietf.org/doc/html/draft-white-openfabric-03.

White, Russ, Danny McPherson, and Srihari Sangli. *Practical BGP*. Boston, MA: Addison-Wesley Professional, 2004.

Review Questions

1. Read the OpenFabric documentation provided in the "Further Reading" section. Does OpenFabric concentrate on aggregation or summarization? How does OpenFabric reduce the amount of control plane information without dividing up the network into flooding domains?
2. Read the Routing in Fat Trees (RIFT) documentation provided in the "Further Reading" section. Does RIFT concentrate on aggregation or summarization? Describe one technique that RIFT uses to summarize state and how RIFT handles aggregation.
3. When might it be useful to be able to configure overlapping flooding domains in IS-IS?
4. "Stub" in OSPF means what kinds of routing information will always be blocked at an ABR?
5. "Totally" in OSPF means what kinds of routing information will always be blocked at an ABR?
6. "Not so" in OSPF means what kinds of routing information will always be blocked at an ABR?
7. Read RFC7196, *Making Route Flap Dampening Usable*. What problems does this document pose for exponential backoff schemes, and what solutions does it propose to resolve these problems?

PART III

Network Design

Understanding the design and operation of the transport and control plane subsystems of a network is a good start toward being a network engineer. Design involves adding several more points and integrating them into a whole. For instance, network design also involves the following tasks:

- Building security into the design and operation of a network
- Using the network as a tool (where it makes sense) to help secure the attached hosts and applications
- The design patterns used in network design, and where and how to apply those patterns
- Resilience at a system-wide level
- Choosing between the many technologies available to solve the set of problems presented by applications and business drivers (this reaches into the world of the network designer)
- How network design interacts with strategic business interests in the long term, including how the network impacts the directions the company may be able to take in the future (this reaches into the realm of the network architect)

Network design and architecture are very broad fields, far outside the scope of this book. In fact, very little has been written specifically on these larger fields—unlike the other introductory sections in this book. Be sure to consult the "Further Reading" section at the end of each chapter in this part so that the key works in this space are called out in the larger scope.

Three Underlying Models

The field of design relies heavily on models and abstractions; design tends to be more of a "seat of the pants" affair, grounded in experience and a broad knowledge set. Although they have been covered in other areas of this book, it is important to keep three specific models in mind when reading these chapters on design.

The Law of Leaky Abstractions

Many abstractions are meant to be "perfect," in that they completely contain information within a single system. For instance, the Transmission Control Protocol (TCP) is designed to provide what appears to be a connection between two hosts (or two applications) across a network that does not guarantee packet delivery—in order, or at all. In reality, there are many situations where the operation of the Internet Protocol (IP)—which underlies TCP, and the physical links that underlie IP—will be directly visible in the operation of TCP. The law of leaky abstractions applies to almost every type of abstraction undertaken in a network, from layering protocols, to aggregating reachability and topology information, to building an overlay network "over the top." A lot of complexity is driven into network protocols and design through various attempts to account for state leaking outside what should be a fairly watertight abstraction.

The State/Optimization/Surface (SOS) Triad

This was originally explained in Chapter 1, "Fundamental Concepts," and is referenced throughout the rest of the book. Much of the art of design is consciously considering this set of tradeoffs at a system level; many designs have become overly complex, and hence overly fragile, because designers tend to focus on goals rather than tradeoffs. Consider this in terms of the decision to deploy an overlay (or even what kind of overlay to deploy). Deploying an overlay certainly decreases the amount of state, and the speed at which state changes, in the resulting underlay and overlay. The overlay can inject additional state at the overlay layer to use resources more efficiently (as an example, see Chapter 25, "Disaggregation, Hyperconvergence, and the Changing Network," on network function virtualization). But introducing a second control plane and an "over the top" transport layer also creates a broad, and often deep, interaction surface. Will deploying the overlay ultimately increase overall complexity, or reduce it? What can be done to mitigate this additional complexity? Where will the underlay, as an abstraction, leak? What steps might need to be taken to stem this leak, and how much more complexity will this add?

Every design discussion, every design decision, needs to be driven by asking questions like these about tradeoffs.

The Consistency/Accessibility/Partitioning (CAP) Triad

The CAP theorem is widely known and appreciated in the database design field, but is not often considered in the world of network design. In reality, CAP tells the designer that there is a time cost to distance and processing. The more distance and processing separating a data source from the ultimate data destination, the more time it will take for the data to get there. When decisions are dependent on the presence of data, this means that distance and processing requirements will ultimately slow down the pace at which decisions can be made. Hence, the ideal situation is where decisions are distributed to the point closest to where the data required to make the decision "lives"—this is known as the subsidiarity principle. The key point to remember is the source of the data; the source of business policy is the business, so decisions about policy need to be close to the business. On the other hand, the source of routing information to find loop-free paths is the network devices that have near-real-time access to the state of topology and reachability in the network, so it makes sense to put decisions based on topology changes close to the network devices that actually forward traffic.

The chapters in Part III assume all of these factors need to be considered in each design realm; knowing and applying them will speed your capabilities as a network designer. The chapters in this part include:

- **Chapter 21: Security: A Broader Sweep**, with discussions of the different components of the security environment, defense in depth, information privacy, and the OODA loop
- **Chapter 22: Network Design Patterns**, with discussions of the relationship between business and network design, network ownership models, choke points, hierarchical design, layering, common network topologies, and regular topologies
- **Chapter 23: Redundant and Resilient**, with discussions of control plane failures, control plane convergence, measuring network availability, graceful restart, in service software upgrades, and modularization for resilience
- **Chapter 24: Troubleshooting**, with discussions of the narrowing process, breaking networks into components, the *how* model, the *what* model, troubleshooting tools, models in troubleshooting, the half split method, and technical debt

This page intentionally left blank

Chapter 21

Security: A Broader Sweep

Learning Objectives

When you are finished reading this chapter, you should understand:

- The difference between a threat actor, an exploit, an attack, a vulnerability, an asset, and a risk
- The concept of defense in depth, and how it relates to security
- The concept of AAA and what AAA systems are designed to accomplish
- The concept of data exhaust
- The process used to exchange private keys
- The concept of a distributed denial of service attack and protection mechanisms
- The relationship between control plane security and securing traffic passing through the network
- The OODA loop and its application to network security

Security is often placed last in any discussion of network design principles; it is often thought of as an add-on to the main focus of the design process. The modern world, however, is a dangerous place for data, particularly data that can ruin people's lives *permanently*.

The Scope of the Problem

Security is a very broad and important topic for network engineers; the following sections will outline why this is so.

The Biometric Identity Conundrum

Consider this simple example: many electronic devices have a fingerprint reader that is (often) used in the place of a password. Gaining access to such devices is much simpler than password or pin-driven access; there is no password to remember. It is also (theoretically) more secure. You cannot "steal" someone's fingerprint.

Or can you? There are two less than obvious lines of attack. First, you leave your fingerprint everywhere in everyday life. It is on the screen of your cell phone, the doorknob to any building you enter, the door handles on your car (or handlebars on your scooter or bike), and in many other places. People have long been able to lift such prints from a wide array of services. How much of a secret is your fingerprint, really? This same problem applies to any externally visible body characteristic used to identify you: cameras are everywhere, and at least some of them capture just about any part of your body used for identification on a regular basis. Figure 21-1 illustrates this problem.

The second line of attack is, perhaps, less obvious but maybe more dangerous. No system stores fingerprints as images, per se. Rather, all fingerprint systems store fingerprints as a digitized version of the key characteristics of each fingerprint.

Figure 21-1 *The fingerprint as a password problem*

Certainly such files will be encrypted and protected, and perhaps even just stored locally. Once fingerprint data is taken online, however, all security bets are off. No matter how well protected, data moved across the public Internet has some percentage probability of being stolen at some point. Data breaches are common; for instance, here are a few sample breaches from 2016:

- FACC, a manufacturer of lightweight composites, was the victim of cyber theft worth at least $54.5 million.[1]
- The University of Florida, exposing the records of about 63,000 students and staff.[2]
- The FBI, exposing the contact information of about 20,000 employees.[3]
- The United States Internal Revenue Service, exposing the information of about 220,000 tax payers.[4]
- The University of California at Berkley, exposing information of about 80,000 students, faculty, and alumni.[5]
- Premier Health Care, exposing information about 200,000 patients.[6]
- Verizon Enterprise Services, potentially exposing information about 1.5 million customers.[7]
- The City of Salt Lake City, Utah, exposing information about 14,200 people.[8]
- Tidewater Community College, exposing information about 3,000 employees.[9]
- The voting system of the Philippines, exposing information about 55 million citizens.[10]
- Yahoo, exposing the information of between 500 million and 1 billion users.[11]

1. "EANS-Adhoc: FACC AG / UPDATE: FACC AG - Cyber-Fraud."
2. Leary, "UCF Data Breach."
3. CNN and Mallonee, "Hackers Publish 20,000 FBI Employees' Contact Information."
4. Leary, "IRS Data Breach Grows."
5. "Data Breach Affects 80,000 UC Berkeley Faculty, Students and Alumni."
6. "Premier Healthcare Faces Possible Data Breach That Could Affect 200,000 Patients."
7. Leary, "Verizon Enterprise Data Breach."
8. Gorrell, "Salt Lake County Data Breach Exposed Info of 14,200 People."
9. McKinney, "Data Breach Exposes Information on More than 3,000 TCC Employees."
10. Muncaster, "Every Voter in Philippines Exposed in Mega Hack."
11. Siciliano, "Yahoo Data Breach: Almost 500 Million Affected"; "1 Billion Yahoo Accounts Compromised in Data Breach | IdentityForce.

There are hundreds (or thousands) of such breaches each year, many of which escape the notice of the wider public or are not reported at all. Once fingerprints are stored like credit card and other information, it is a matter of time before large caches of fingerprint information are stolen. Of course, not every fingerprint for every person in the world will be stolen in such a breach, but this will be small comfort to those whose fingerprints were stolen.

Definitions

To understand the world of security, it is important to understand some basic terms. Figure 21-2 illustrates the first set of important definitions.

In Figure 21-2, working from left to right:

- The **threat actor**, or **attacker**, is the individual or organization initiating the attack(s). The identity of the threat actor can help you understand motivations, skill level, and possible plan of attack.
- The **exploit** takes advantage of the vulnerability using a process or tool, such as manually entering a specific string, running a script or piece of software, etc.
- The **attack** or **threat** is the potential or actual attack performed by the threat actor using the exploit.
- **Vulnerabilities** are potential weak spots in the defense that can be exploited to achieve an objective. For instance, a missing or misconfigured packet filter, someone inside the network with rights to the target system who can be social-engineered to provide access in some way, or a defect in a system's code allowing a threat actor to attack a system in some way.

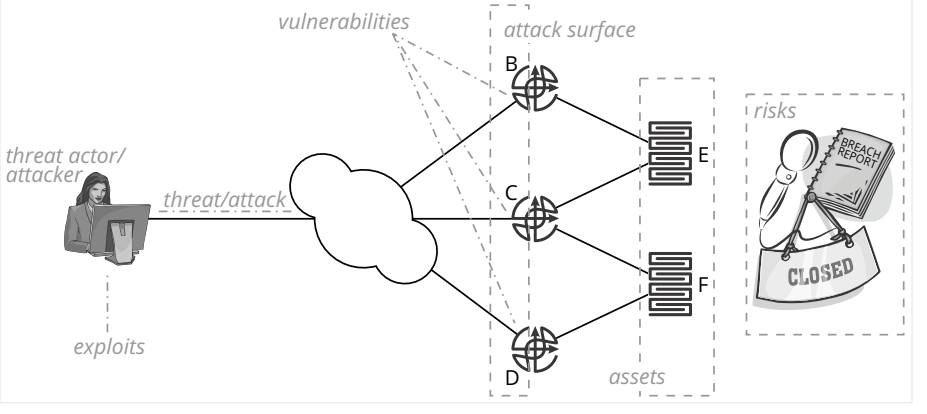


Figure 21-2 *Security definitions*

- The **attack surface** is the total set of systems, port numbers, applications, etc., the threat actor has access to in some way.
- **Assets** are the network elements and information within the network that the threat actor would either like to access or prevent access to (in the case of a denial of service attack).
- **Risks** are the potential negative results of an attack, such as bad publicity (represented by the microphone), a major business consequence (represented by the breach report), or even a complete failure of the business itself. Risk is often quantified as potential times impact.

The Problem Space

The security problem space, from a network perspective, can be divided into three broad areas:

- How can users and processes access the data they need to do their jobs?
- How can the information carried over the network and stored on devices connected to the network remain confidential?
- How can the network remain accessible? Many attackers would like to disrupt a business or organization by removing the network as a usable resource; this is called a denial of service (DoS) attack.

The following section will consider a wide scope of possible solutions to each of these problems. After this, some examples of solutions in the security space will be considered, and then a useful model for considering network security, the Observe, Orient, Decide, Act (OODA) loop, will be discussed.

The Solution Space

Many books, articles, and research papers have been written addressing different elements of security since the first network break-in, which probably happened the day after the first network was operational. The "Further Reading" section will be helpful if you are interested in learning more about security than what is contained in this and later sections in this chapter.

This section will begin by looking at the concept of defense in depth and then consider three broad security solution spaces: access control, data protection, and service availability assurance.

Defense in Depth

The first, and most important, solution is more conceptual than tool or method oriented. Defense in depth is the concept of having multiple, overlapping layers of defenses interacting in a way that poses multiple challenges to an attacker. Figure 21-3 illustrates one possible set of lines of defense.

In Figure 21-3, there are a number of lines of defense, including

- Packet filters and access controls at the routed edge to the network; these are "basic" controls that just check to see if a user is authorized to access the network in general, block some basic (obvious) packet flows, and even limit the rate at which hosts outside the network can transfer data.
- Route validation at the routed edge to the network; this will help prevent access from hijacked address space.
- General telemetry throughout the network, which will indicate top talkers, provide information on the most common source/destination pairs, note unusual spikes in utilization, etc.
- Stateful packet filters at the middlebox C, which will only allow traffic into the network on some ports if there is an existing connection.
- Exfiltration monitoring deployed in several locations; this will raise an alert when specific access patterns occur, such as large amounts of data being transferred toward a destination outside the network from a database containing sensitive user information.
- Access control on individual services and/or servers, such as G, which ensures the user at D is authorized to access individual resources.

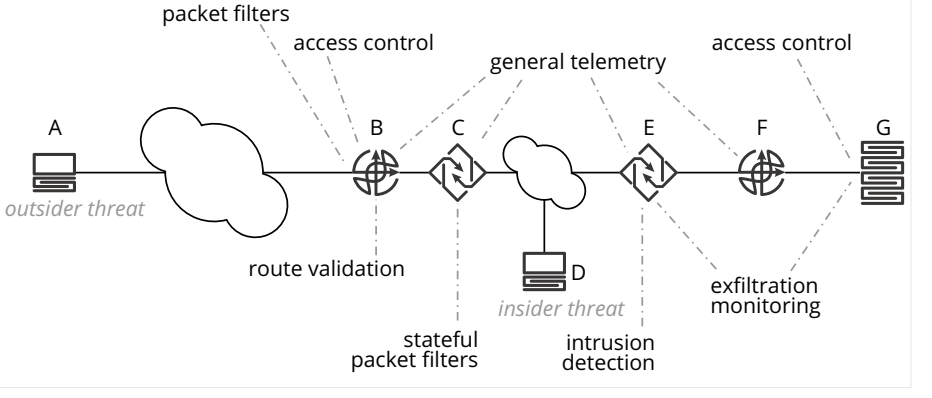


Figure 21-3 *Defense in depth*

Although none of these systems will stop an intruder from breaching your network or data, used together they can provide a fairly effective defense system against many different forms of attack. First, the time a threat actor spends moving through one layer gives the network operator an opportunity to discover and counter the attack with more specific controls. Second, the layering of systems and filters forms a kind of "layer of grids" through which traffic must pass to reach important resources. An attack not blocked by the first grid layer may be blocked by the second, etc.

Understanding the in-depth defensive posture of the complete set of systems in a network and using every available resource as a potential defensive system are important skills in the network design space.

Access Control

Access control tries to ensure

- Users (or processes) are who they claim to be—to verify identity. This is often called authentication.
- Users (or processes) are able to access the data they are attempting to access, or rather whether or not they are authorized to use a particular service or access a particular piece of data.
- Information about user actions are recorded so they can be used to trace back to failures and breaches. This is generally called accounting.

Access control systems are often called AAA systems, because of the three A's: authentication, authorization, and accounting. These systems are often specialized applications, interacting with devices through a protocol such as Remote Authentication Dial-In User Service (RADIUS) to ensure users are properly logged in and applications have enough information to ensure that only valid and authorized users are accessing information and services.

Access control can be implemented in many different places in a system; for instance:

- Before the user connects to the network, or rather before the user can log on to a device that is able to obtain an Internet Protocol (IP) address, connect to an upstream switch, connect to a wireless network, etc.
- After the user connects to the network but before the user can access any service.
- After the user connects to the network and before the user accesses each individual service.

These three different options can be combined; for instance, a user may be asked to provide a username and password before connecting to the network, and again before accessing any service on the network, and then again before accessing particular, more highly restricted, systems or information.

Data Protection

Consider the protection around a safe holding classified documents. Is it possible the safe might be breached in some way? What if someone calls in a bomb threat to the building where the safe is housed? Will the occupants of the building gather up any necessary equipment and information, including the contents of the safe, and leave the building? Or what if someone who appears to be authorized calls and asks for the data? The safe being breached is similar to an access control failure, and the bomb threat or request for information that drives the data out into the street is similar to a request pulling the information across the network.

Ultimately, there must be a line of defense to protect data in these sorts of situations; systems and applications will be breached, and data will be requested over the network. Encryption is generally the last line of defense for these situations.

The Man in the Middle and the Control Plane

When data leaves the safe space, many new vulnerabilities come into play. While encryption can help protect data in these situations, it is useful to know what these situations look like, and hence (potentially) what needs to be protected against and what the limits of protection might be. Man-in-the-middle (MitM) attacks are a common sort of attack in this realm; these attacks are sometimes enabled by an attack on the control plane (an attack on one system that enables an attack on another is called a side attack). Figure 21-4 is used to illustrate.

In Figure 21-4, host A opens a secure session to G. As part of the secure session configuration, A and G must exchange keys and other information to allow an encrypted session to operate. If E, the attacker, can intercept these communications, it can

Make A believe it is exchanging information with G, even though it is exchanging information with E

Make G believe it is exchanging information with A, even though it is exchanging information with E

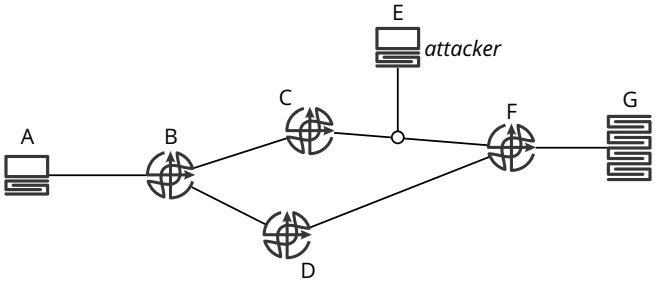


Figure 21-4 *A man-in-the-middle attack*

By acting as a man in the middle, E can make A and G believe they are exchanging information securely, when the entire connection is being monitored by E. The problem, for the attacker, is to intercept the traffic between A and G at just the right point in time to allow this kind of attack to work. This may be fairly easy if there is just one path between A and G, but in Figure 21-4 there are two paths. How can the attacker ensure it will see the traffic flowing between A and G? The attacker must exploit some vulnerability in the routing system to ensure traffic between A and G is flowing across the [C,F] link.

While not all man-in-the-middle attacks will require compromising the routing system in this way, at least some will; hence the security of the routing system itself can be a security issue, as well. Securing the routing system is very difficult, however. For instance:

- If the destination is only reachable through the public Internet, it is next to impossible to know what the "correct" route is. The Internet is made up of individual networks, each of which has policies intentionally unavailable to other operators.
- If the destination is reachable entirely through an internal network, it would be difficult to detect the [B,C,F] path as an invalid path. Both available paths are "valid" in the sense of being available paths between the source and destination; why one should be considered "more valid" than the other is difficult to discern.

These and many other problems plague security in the control plane. While some information can be verified, such as existing connectivity, it is difficult to determine whether or not traffic is following these validated paths. While the network can help prevent man-in-the-middle attacks of this kind, some attacks can only be resolved at the protocol and application level.

Encryption takes a block of information (the plaintext) and encodes it using some form of mathematical operation to obscure the text, resulting in a ciphertext. To recover the original plaintext, the mathematical operations must be reversed. Most encryption is based on the difficulty involved in factoring a large integer composed of two or more prime factors. An integer is calculated based on the key (a prime factor) and some portion of the plaintext, resulting in the ciphertext. To recover the plaintext from the ciphertext, the process is reversed; the key is used to find the factor of the large integers in the ciphertext, ultimately calculating the original plaintext.

There are two kinds of widely used encryption: public key and private key. In public key cryptography, more properly called asymmetric cryptography, there are two factors or keys; if the plaintext is encrypted using one of the keys, it can be unencrypted using the second key. This is useful because it allows one of the two keys to be published publicly. In private key cryptography, more properly called symmetric key cryptography, the same key is used to encrypt and unencrypt the plaintext; hence the sender and receiver must share the same key to communicate.

Public and private key cryptography are often used together to form a single system. Figure 21-5 is used to illustrate.

In Figure 21-5:

1. Assume A begins the process. A will encrypt a nonce, or rather a large random number, using B's public key. Because the nonce has been encrypted with B's public key, in theory only B can unencrypt the nonce, as only B should know B's private key.

2. B, on unencrypting the nonce, will now send some new nonce to A. This may include A's original nonce, or A's original nonce plus some other information. The point is that A must know, for certain, the original message, including A's nonce was received by B—and not some other system acting as B. This is ensured by B including some piece of information encrypted using its public key, as B is the only system able to unencrypt it.

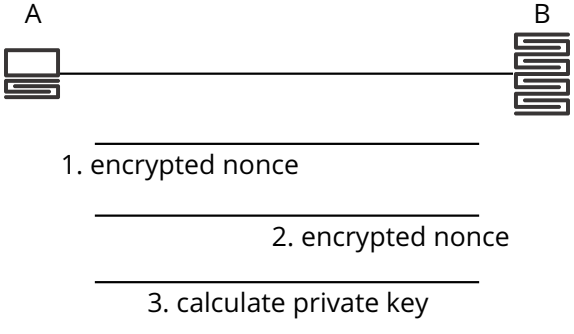


Figure 21-5 *Using public keys to either exchange or calculate a private session key*

3. A and B, using the nonces and other information exchanged to this point, will calculate a private key, which is then used to encrypt/unencrypt information transferred between the two systems.

The steps outlined here are somewhat naive; there are better, more secure, systems, such as the Internet Key Exchange (IKE) protocol; see the "Further Reading" section for resources in this area. Why not just use asymmetric (or public key) cryptography all the time? Because the computational costs of using asymmetric key cryptography are much higher than using symmetric cryptography.

A second area to be concerned about in data protection is data exhaust. There are many other terms for this, of course, but the general idea is vulnerabilities in the communication patterns. For instance, assume a bank configures an automated backup for a particular database table; when the balances in the account held in the table change by a particular amount, the backup is kicked off automatically. This might seem like a perfectly reasonable sort of backup job, but it does involve some amount of data exhaust. If a threat actor puts the backup together with the change in account value, he will know specifically what the pattern of account activity is. Enough clues of this sort can be developed into an entire set of attack plans.

How can network engineers protect against data exhaust? There are no real good ways to protect against leaking information unintentionally into the public domain through such actions; even in security, the law of leaky abstractions applies. The best you can do is to be aware of such problems, potentially profiling your network the same way an attacker would, and noting any patterns that might be used against your defense system.

Security and Obscurity

No security through obscurity. If you get close enough to a security engineer for long enough, or involved in any sort of debate over proper security, you will likely hear these words somewhere along the way. There is one problem with this phrase, however: it is often used out of context. To understand the real meaning of the phrase, you need to go back in time to the origin of encryption algorithms. In the physical lock world, revealing the plans of a lock will often reveal various passageways to bypassing or defeating the lock. This habit was carried over to early software security vendors; if an attacker knows how the encryption algorithm works, she will be able to find ways to defeat the encryption algorithm.

But encryption algorithms are not door locks; what is an important safeguard in one realm can be a dangerous crutch in another. Hiding code developed to encrypt plaintext does not make the code more secure; in fact, just the opposite happens. Instead of improving security, obscuring encryption code and processes just prevents experts in the field from finding flaws and possible ways to defeat the code before these are exposed in real deployments. Ultimately, security by obscurity is dangerous in this particular context.

So this is a good principle, but it can be misapplied. For instance, if a network operator attempts to hide the internal network architecture or addressing, or even block external hosts from reaching internal ones, at least some security experts will counter with "That is security by obscurity; you should not do this." Taken in this sense, however, *encrypting data is also security by obscurity*. Hiding information and hiding information about your infrastructure are both essentially hiding information, and hiding information is essentially a form of obscurity.

How can you tell when you should apply "no security by obscurity" and when you should not? Perhaps the best rule of thumb is this: hiding processes, algorithms, and implementations is not a useful addition to security in the cyberworld. Hiding information, however, often is. It can be hard to apply this rule of thumb in many situations, but it should be a good start in thinking through the issues and making the right decision in each particular case.

Service Availability Assurance

Distributed denial of service (DDoS) attacks are on the rise, with the largest reaching over 1 terabits per second in late 2016,[12] using hijacked Internet of Things (IoT) devices, called a botnet. Figure 21-6 illustrates one way such an attack can be built.

The process in Figure 21-6 begins before the attack, with the creation of a botnet to use as a platform. Building botnets is mostly a matter of getting as many devices as possible infected with a virus, allowing a controller to instruct the device to send a stream of packets to some IP address on demand. Viruses are designed to infect a wide range of devices, including IoT devices (like light bulbs, refrigerators, video cameras, television sets, etc.), personal computers, cell phones, and web servers (which normally run inside a virtual machine), ultimately allowing them to be controlled in some limited sense by the botnet controller. Such botnets can be rented by the hour fairly easily.

12. Khandelwal, "World's Largest 1 Tbps DDoS Attack Launched from 152,000 Hacked Smart Devices."

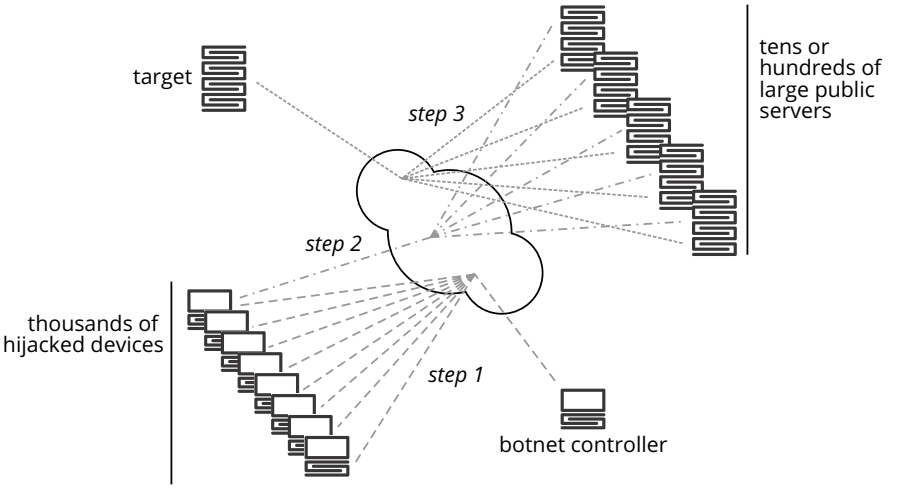


Figure 21-6 *A DDoS reflection attack*

Once the botnet is built:

1. The botnet controller sends a command for each of the devices, potentially hundreds of thousands of them, to send a series of packets to a set of well-known servers. Any sort of server hosting a widely used public service with a lot of bandwidth and processing power will do; favorites are Domain Name System (DNS) and Network Time Protocol (NTP) servers.
2. The botnet devices send requests for some piece of information to each server being used as a reflector in the attack. Typically, this is a request for a DNS resolution, or a large text record stored in the DNS table, or something similar. The source of the request is forged or set to the target's IP address.
3. The servers respond to the request with large amounts of data, which is then sent to the target device. Some resource, such as available bandwidth, available Transmission Control Protocol (TCP) connection buffers, or something else with a limited scale, is consumed, preventing the server from operating properly (such as preventing a web server from serving web pages to visitors).

Why do threat actors build and launch these kinds of attacks? There are a number of reasons, including

- To extort money from businesses. A large-scale attack against a well-known target is particularly effective for extortion; a threat actor can send an email to hundreds of companies saying something like: "Did you see the news about

the big DDoS attack? That was me. If you do not pay me (some large amount of money), you will be next." If the targeted company perceives it has fewer resources and skills than the well-known target, it may pay rather than trying to defend itself against such a large attack.

- To make a political point. Some attacks are targeted at organizations that the threat actor disagrees with politically, such as a company failing to support a specific cause, a rival political party, etc.
- To bring down a competitor. Some threat actors sell the service of taking down a rival's website for some period of time, in order to embarrass the company or drive users to a competitor.
- To distract the security team at a company while some other attack is occurring. DDoS attacks are often a useful feint to distract the corporate security team while some form of back door or other vulnerability is created in the victim's network.
- Because they can. Some people just seem to enjoy wreaking havoc, or they believe it is the only way they will ever become famous.

There are a number of ways to defend systems against DDoS attacks, many of which can (and should) be used in parallel.

Reflection, Amplification, and Burner Attacks

Why is the indirection off a public server used, rather than using the botnet to attack targets directly? There are generally three reasons. First, servers designed to support large-scale public services, like DNS and NTP, are generally well-connected, high-powered systems. In fact, these are often multiple systems operating behind a single anycast address; such servers will be able to generate a lot more traffic than a collection of hosts connected at the edge of the Internet, and often represent services able to amplify the attack. In an amplification attack, the attacker sends a small request that will result in a large response, directing the response to the victim. Second, these services are not often blocked; since they are large, publicly known services, and normally crucial to the proper operation of the Internet as a whole, they are normally given a "special pass" when it comes to any sort of packet filter. Third, if the actual botnet devices are used, their use reveals their locations to the device and/or network being attacked, as well as the upstream provider to which the systems within the botnet are connected. Once their locations have been revealed, they are often blocked, or some form of mitigation takes place, making the devices used in a direct attack much less useful in future attacks. Direct attacks are sometimes called burner attacks because of how they reveal the botnet itself; the resources have been outed and hence burned.

Modifications to Host Operating Systems

Some modifications can be made to host operating systems that will allow the server to withstand the traffic of a DDoS attack while continuing to provide service (though perhaps more slowly). These modifications primarily relate to making more resources available, closing incomplete connection requests more quickly, more quickly aging out cached information that is not currently being used, and other measures.

Blocking Half-Open and Malformed Sessions

Server protocol implementations, and even (to some degree) on-edge routers, can block half-open and malformed sessions. A normal Transmission Control Protocol (TCP) session setup has multiple steps:

1. The client requests a connection by sending a synchronize (SYN) packet to the server.
2. The server replies with an acknowledgment of the connection request (SYN-ACK).
3. The client acknowledges receipt of the SYN-ACK with an ACK; the three-way handshake is complete, and data can be transmitted over the session.

In some TCP DDoS attacks, the client will send the SYN but never acknowledge the SYN-ACK. This is called a half-open session. Open ports represent consumed resources on the server while costing the attacker very little. Many routers and stateful packet inspection devices can drop half-open TCP sessions.

Another option in the case of TCP-based DDoS attacks is for the server to push processing work back onto the systems used in the attack. One way to do this is to allow the server to respond to TCP SYN messages with a malformed SYN-ACK. If the client is running a well-designed, unmodified TCP implementation, this will cause the system used in the attack to spend processing and memory resources reporting the error back to the server. This additional load will reduce the amount of bandwidth and processing power the botnet has available to pursue the attack.

Very few of these responses will work as responses to attacks based on sessionless transport protocols, such as the User Datagram Protocol (UDP).

Rate Limiting

Many operating systems offer the ability to limit the number of incoming connection requests over a specific time scale (usually something like *x hundred/thousand connection requests per second*). Some network devices extend this concept to control plane protection, which limits the rate at which information is transmitted from

the data plane into the control plane for processing. These schemes do save resources but often at a cost: both good and bad traffic are dropped. Schemes may be applied to drop just bad traffic, but defining bad traffic is difficult. *There is no evil bit in the IP packet*.

Spreading Traffic across Multiple Servers

For operators with a very large (or dispersed) edge, it is possible to use routing controls to spread the DDoS traffic among as many entry points into the network as possible. For instance, a 1T attack, if spread across 1,000 servers/network edge entry points, becomes a 1k data stream at each server/entry point, which can be ignored. Figure 21-7 is used to illustrate.

In Figure 21-7, AS65000 has six entry points, each feeding a separate server (or set of servers).

Assume the attacker has IoT devices scattered throughout AS65002 that are being used to launch an attack. Due to policies within AS65002, the DDoS attack streams are forwarded into AS65001, and thence to A and B. It would be easy to shut down these two links, forcing the traffic to disperse across five entries rather than two (B, C, D, E, and F). If you split the traffic among five entry points, it may be possible to eat the traffic. Each flow is now less than one-half the size of the original DDoS attack, perhaps within the range of the servers at these entry points to discard the DDoS traffic.

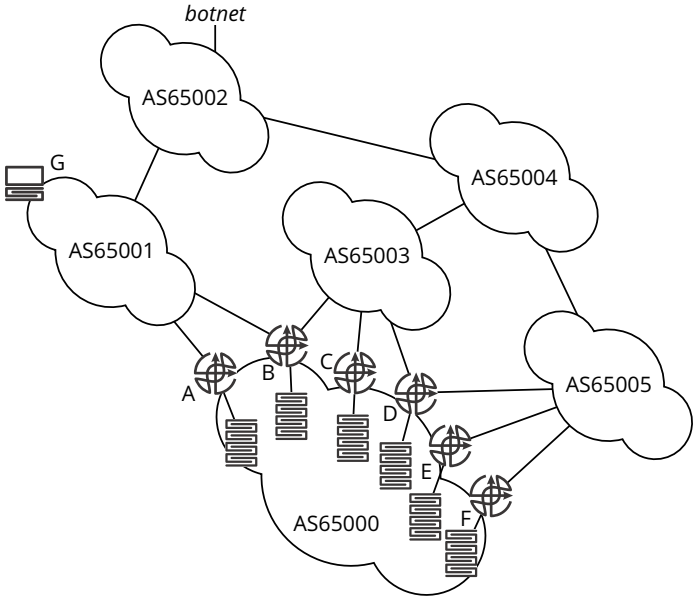


Figure 21-7 *Dispersing a DDoS attack*

However, this kind of response plays into the attacker's hand, as well. Now any customer directly attached to AS65001, such as G, will need to pass through AS65002, from whence the attacker has launched the DDoS, and enter into the same five entry points. How happy do you think the customer at G would be in this situation? The most likely answer is *not very*.

Is there another option? Instead of shutting down these two links, it would make more sense to try to reduce the volume of traffic coming through the links and leave them up. To put it more shortly, if the DDoS attack is reducing the total amount of available bandwidth you have at the edge of your network, it does not make a lot of sense to reduce the available amount of bandwidth at your edge in response. What you want to do, instead, is reapportion the traffic coming in to each edge so you have a better chance of allowing the existing servers to discard the DDoS attack.

One possible solution is to prepend the Autonomous System (AS) path of the anycast address being advertised from one of the service instances. Here, you could add one prepend to the route advertisement from C and check to see if the attack traffic is spread more evenly across the three sites. However, this isn't always an effective solution. Further, if this is an anycast service, the address space cannot be broken up into smaller bits. So what else can be done?

There is a way to do this with the Border Gateway Protocol (BGP): using communities to restrict the scope of the routes being advertised by A and B. For instance, you could begin by advertising the routes to the destinations under attack toward AS65001 with the NO_PEER community. Given that AS65002 is a transit AS (assume it is for this exercise), AS65001 would accept the routes from A and B but would not advertise them toward AS65002. This means G would still be able to reach the destinations behind A and B through AS65001, but the attack traffic would still be dispersed across five entry points, rather than two. There are other mechanisms you could use here; specifically, some providers allow you to set a community telling them not to advertise a route toward a specific AS, whether the AS is a peer or a customer. You should consult with your provider about this, as every provider uses a different set of communities, formatted in slightly different ways; your provider will probably point you to a web page explaining its formatting.

If NO_PEER does not work, it is possible to use NO_ADVERTISE, which blocks the advertisement of the destinations under attack to *any* of AS65001's connections of whatever kind. G may well still be able to use the connections to A and B from AS65001 if it is using a default route to reach the Internet at large.

It is possible to automate this reaction through a set of scripts, but as always, it is important to keep a short leash on such scripts. Humans need to be alerted to either make the decision to use these communities or to continue using these communities; it is too easy for a false positive to lead to a real problem.

Filtering Unroutable Addresses

Since (at least some) attack traffic is originated from unused and/or unroutable address space (called bogon routes), filtering these routes can be useful in blocking some amount of DDoS attack traffic.

Unicast Reverse Path Forwarding (uRPF)

Figure 21-8 is used to explain uRPF filters.

Assume A is infected with a virus, making it part of a botnet; at some point, the host is going to be configured to send some stream of packets to a public server, which will then be reflected to a target machine. The botnet could instruct the host to use its actual address, but this will not work for some forms of attack. For instance, a DNS server will respond to the source address in the packet containing the DNS request.

The preferred method for an attacker is this: instruct A to use a spoofed, or hijacked, address. For instance, the botnet controller may instruct A to use an address in 2001:db8:3e8:100::/64 address space because C is the attack's target.

There is a somewhat simple way for B to block this spoofed traffic. When switching traffic, B can look up the route to the source address of the packet being switched. If the source address is

- Not reachable, the packet should be dropped; this is loose uRPF.
- Reachable only through some interface other than the one the packet was received on, drop the packet; this is strict uRPF.

If B is configured with strict uRPF, at least on the ports to which customers are connected (such as the port that A is connected to), then traffic sourced from A, with B's source address, would be dropped.

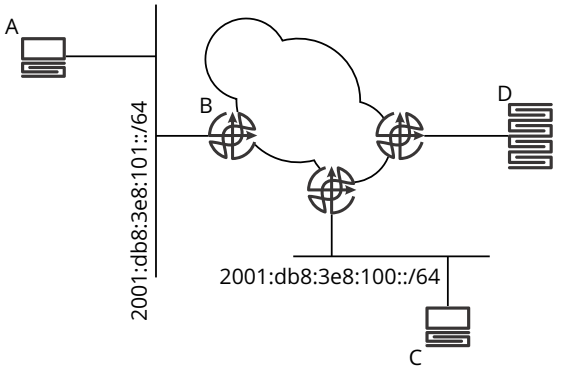


Figure 21-8 *Unicast reverse path forwarding filters*

If uRPF can prevent many forms of reflection DDoS attacks, why is it not configured on every port? Strict uRPF does not work in all situations; there are many legitimate reasons why a packet may not be entering the same interface the router would use to reach the source address. The primary reason for this is dual-homing situations, where the provider installs just one route to the destination, but packets are transmitted along both routes by the actual hosts. It is also difficult to implement uRPF in a way that does not impact the performance of very high-speed links.

Blocking a DDoS Upstream

One of the problems with a large-scale DDoS attack is that your entire upstream link can be consumed in the attack. One solution is to signal your upstream provider to block the DDoS flows. Flowspec can be used to carry packet-level filter rules in BGP. The general idea is this: you send a set of specially formatted communities to your provider, who then automagically uses those communities to create filters at the inbound side of your link to the Internet. There are two parts to the flowspec encoding, as outlined in RFC5575bis: the match rule and the action rule. The match rule is encoded as shown in Figure 21-9.

There are a wide range of conditions you can match on. The source and destination addresses are pretty straightforward. For the IP protocol and port numbers, the operator sub-TLVs allow you to specify a set of conditions to match on, and whether to *AND* the conditions (all conditions must match) or *OR* the conditions (any condition in the list may match). Ranges of ports, greater than, less

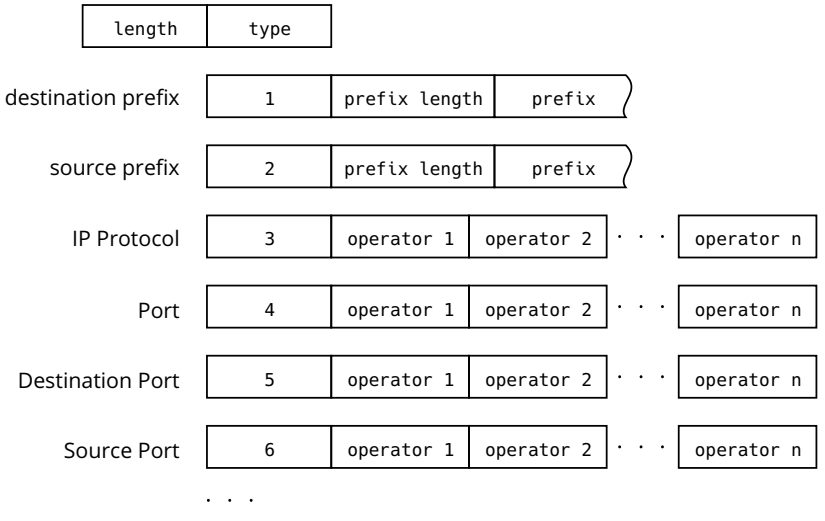


Figure 21-9 *Flowspec encoding*

than, greater than or equal to, less than or equal to, and equal to are all supported. Fragments, TCP header flags, and a number of other header information can be matched on, as well.

Once the traffic is matched, what do you do with it? There are a number of rules, including

- Control the traffic rate in either bytes per second or packets per second
- Redirect the traffic to a VRF
- Mark the traffic with a particular DSCP bit
- Filter the traffic

If you think this must be complicated to encode, you are right. This is why most implementations allow you to set pretty simple rules, and handle all the encoding bits for you. Given flowspec encoding, you should just be able to detect the attack, set some simple rules in BGP, send the right "stuff" to your provider, and watch the DDoS go away. If you have been in network engineering since longer than "I started yesterday," you should know by now—nothing is ever this simple.

If you do not see a tradeoff, you have not looked hard enough.

First, from a provider's perspective, flowspec is an entirely new attack surface. You cannot let your customer just send you whatever flowspec rules it likes. For instance, what if your customer sends you a flowspec rule blocking traffic to one of your DNS servers? Or, perhaps, to one of its competitors? Or even to its own BGP session? Most providers, to prevent these types of problems, will apply any flowspec-initiated rules to just the port connecting to your network directly. This protects the link between your network and the provider, but there is little way to prevent abuse if the provider allows these flowspec rules to be implemented deeper in its network.

Second, filtering costs money. This might not be obvious at a single link scale, but when you start considering how to filter multiple gigabits of traffic based on deep packet inspection sorts of rules—particularly given the ability to combine a number of rules in a single flowspec filter rule—filtering requires a lot of resources during the actual packet switching process. There is a limited number of such resources on any given packet processing engine (ASIC) and a lot of customers who are likely going to want to filter. Since filtering costs the provider money, it is most likely going to charge for flowspec, limit which customers can send it flowspec rules (generally grounded in the provider's perception of the customer's cluefulness), and even limit the number of flowspec rules that can be implemented at any given time.

Using a DDoS Scrubber Appliance or Service

A number of appliances will use local information, along with analytics gathered from a wide range of networks, to discover and block DDoS-specific flows. These appliances can be deployed inside your network, in front of or behind your edge router, as a security device. DDoS protection services can scrub your inbound traffic, as well; Figure 21-10 illustrates one way in which these services work.

There are five steps in Figure 21-10:

1. A host, A, requests the IP address for some domain, say example.com, from a DNS server.
2. The DNS server responds with an IP address pointing to the DDoS scrubber service, hosted in a content or service provider's network.
3. The host sends its traffic to the scrubber service at B.
4. B removes any DDoS traffic, leaving just the goodput, and then tunnels the remaining traffic across the Internet to the original server, C.
5. The server responds to the request as normal, sending the information directly back to the requesting host.

Any device that is part of a botnet will also receive the scrubber's address as the correct one to reach the service under attack. The scrubber service is normally positioned in a network able to consume many gigabits of traffic, remove any traffic that appears to be part of a DDoS attack, and send the remaining traffic on to the original

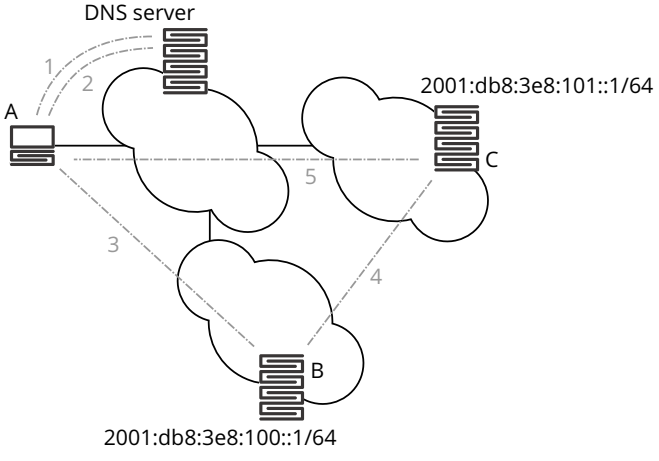


Figure 21-10 *DDoS scrubbing service operation*

server. Such scrubbing services go far beyond examining the traffic, using near-real-time information about active botnets, information from DNS queries, and other factors to determine which traffic is goodput and which is part of the DDoS attack.

The OODA Loop as a Security Model

The OODA loop was originally developed by Colonel John Byrd of the United States Air Force to help fighter pilots manage decisions quickly in life-or-death situations. While network security might seem to be far outside the realm of military aircraft, the OODA loop has proven useful in a number of different security-related (and more generally reaction-related) situations. The OODA loop consists of four steps:

- Observe
- Orient
- Decide
- Act

The four steps begin with the letters O, O, D, and A—hence the OODA loop. Figure 21-11 illustrates.

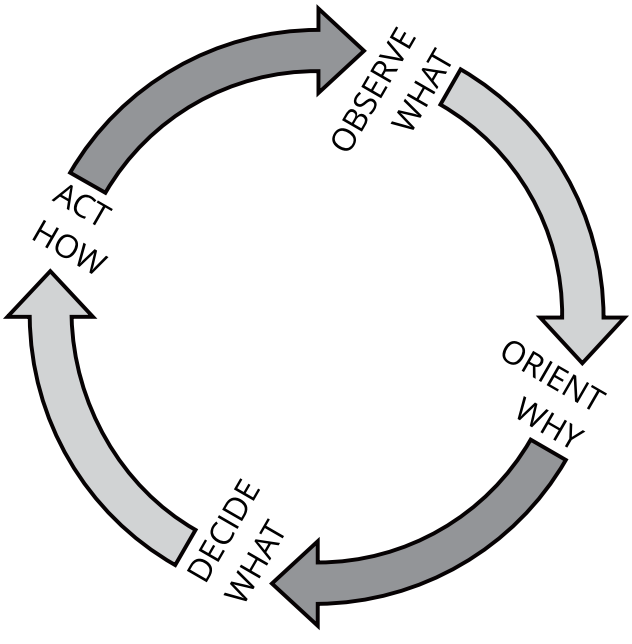


Figure 21-11 *The OODA loop*

If you have ever heard the expression *you need to get inside the loop*, this comes from the OODA loop. The person who has the "tightest loop," or who can move through the loop the fastest, will win the contest. In terms of network security, you must be able to get inside the threat actor's loop to get ahead of him and to find ways to stop the attack in progress.

Each of the four steps deserves a closer look.

Observe

What should you observe, and where should you observe it? In some cases, this is the most important question to ask and the hardest to answer. Should you measure the average traffic flow across specific points in the network? The average jitter across specific points? The average delay? The number of routes in the routing table? The rate at which the routing table changes?

The right answer is to measure everything that will give you a good feel for the day-to-day operation of the network—with some caution. There may seem to be little harm in acquiring telemetry data from every device and every part of the network possible, and throwing the information away after a short period of time if it does not prove useful. The realistic answer, however, is you must choose your observation points carefully, after much trial and error, and after much thought about traffic flows, failure modes, etc.

There is a second point hidden in observe, however: how do you know what you're observing unless you record? As the old saying goes, "if you didn't write it down, it didn't happen"—and nothing is truer than this in the world of observation. There's no point in knowing what's happening right now unless you know what has happened in the past.

Orient

Once you've made a set of observations, you need to decide what it is you're observing. Consider the simple optical illusion shown in Figure 21-12.

In Figure 21-12, there are two sets of squares, one of which is imposed on a background of geometric lines. On the right, the squares are actually square. On the left, however, the squares do not look square; they appear to be distorted. What you see is often determined by the context as much as what is there. Observing, therefore, is a *skill* you can develop over time. Observing a network to understand its normal state is like any other observational skill. How can engineers develop observation skills?

First, understand the operation of the network, protocols, and applications at a theoretical level. Reaching beyond the command line, and into the actual operation of the devices in the network—understanding how a router forwards packets, or how OSPF builds and processes packets, can make a huge difference in your ability to orient yourself to what you are observing.

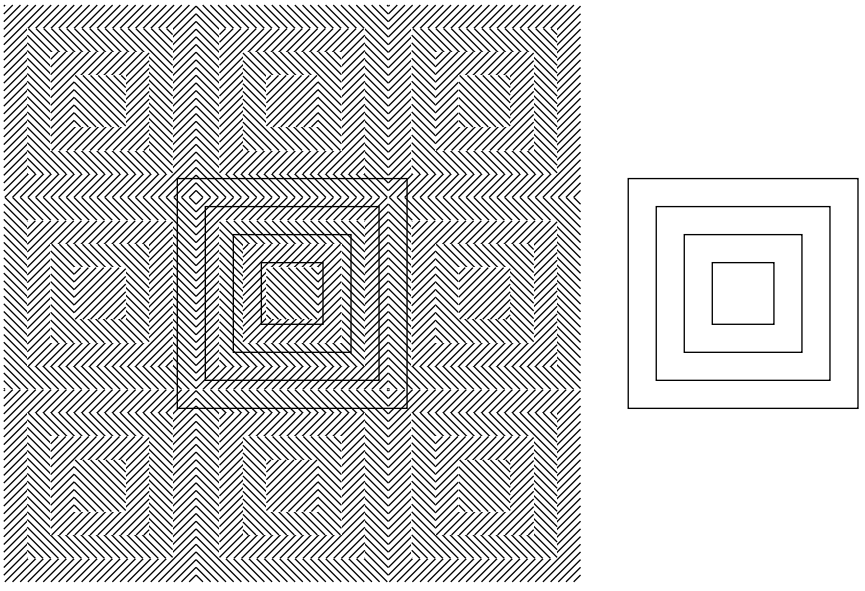


Figure 21-12 *An observational illusion*

Second, learning and applying models is a huge help. The reason you probably have trouble with the optical illusion in this figure is the boxes appear close to being squares, so you immediately think they must be squares. You have a "model of a square," in your head, and when you see things close to the model, you try to make the object fit. So it is important to know a wide array of models into which any problem can fit—or, in the networking world, a wide array of models you can use to "see" protocol, application, device, and network operation. Each additional model you add to your "mental model set" allows you to orient yourself a bit faster.

This entire process is much like orienteering. First, get the map pointing north. Then, find the features on the map matching your location, and work from there to the destination, feature by feature. Not orienting the map is failing to separate the background from the information. Not being able to see the surrounding area is failing to collect the information necessary to match the map to the reality. Not knowing the symbols on the map is failing to have enough mental models to make the match between map and reality happen.

Decide

The worst time to make any sort of decision is at 2 a.m., in the middle of a network outage, when you are under pressure to get the business back up and running. But given network outages never happen at a convenient time, how can you avoid making these kinds of decisions?

You can decide what you are going to decide before you must decide.

This might sound a little roundabout; perhaps an example from the world of self-defense training classes would be helpful. When is the best time to decide where you are going to go if someone attacks you? Before he does so, or while he is doing so? Defensive driving is no different: it is always best to know where you would go if the car in front of you suddenly spins out, or the wheels fall off, or some other thing you might not expect to happen.

This pre-decision process can be very helpful in a network environment. For instance:

- Where would you put a filter to block this particular type of traffic?
- Which parallel links would you remove to kill off the positive feedback loop keeping your routing protocol from converging?
- What servers can you shut down for a time while you are trying to figure out why the data center fabric has become so hot all of a sudden?

All of these decisions are choices you can make before the action starts—before you have to decide to do something. In other words, decide what you need to do so that when it comes time to do it, you will have a plan in place.

Act

It should be easy to act once you have observed the situation, oriented yourself to what is happening, and considered the preplanned decisions you made before the 2 a.m. call that the network is down—but it often harder to act than it should be. Why?

First, it is often hard to believe "this is actually happening." This is a common problem in self-defense situations; when you first encounter a problem, you do not want to adjust to the new situation. Rather, you would rather just ignore the problem and "move on with life." This is Scrooge, in *A Christmas Carol*, saying to Marley, "there is more gravy than grave about you." In the real world, however, this can be a very costly way to react.

Second, a storm of doubts will naturally accompany the actual moment of decision. Did you really observe this particular attack? What if you are wrong—will the consequences be worse than the attack itself?

The answer to both of these problems lies in the OODA loop itself. If you have staked out observation points, if you have oriented yourself against the background information you have, if you are following premade decisions made in more rational times, then acting is the right next step to take.

Hone your skills, know your network, know your monitoring points, know what you are looking at, know your plan, and do it. Make your plan, and then trust your plan.

Final Thoughts on Security

Security is not simple; it is a broad field with a lot of potholes you can step in to, often without realizing you have, in fact, just stepped into a pothole. The castle walls of firewalls and demilitarized zones (DMZs) are in the distant past. Cannons have long since been invented, and the castle walls of the older world of network engineering are just testaments to a time long gone. Threat actors are everywhere, so defense must be everywhere, as well.

This chapter has not considered every possible defense, including such popular topics as microsegmentation and white listing, but it has described a set of helpful mental tools for understanding the world of network security.

Further Reading

Bacher, Martin, Robert Raszuk, Susan Hares, Danny R. McPherson, and Christoph Loibl. "Dissemination of Flow Specification Rules." Internet-Draft. Internet Engineering Task Force, February 2017. https://datatracker.ietf.org/doc/html/draft-hr-idr-rfc5575bis-03.

CNN and Mary Kay Mallonee. "Hackers Publish 20,000 FBI Employees' Contact Information." *CNN*. Accessed April 23, 2017. http://www.cnn.com/2016/02/08/politics/hackers-fbi-employee-info/index.html.

Czyz, J., M. Luckie, M. Allman, and M. Bailey. "Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy." In *Network and Distributed Systems Security (NDSS)*, 2016. https://www.caida.org/publications/papers/2016/dont_forget_lock/.

"Data Breach Affects 80,000 UC Berkeley Faculty, Students and Alumni." Text.Article. *FoxNews.com*, February 28, 2016. http://www.foxnews.com/tech/2016/02/28/data-breach-affects-80000-uc-berkeley-faculty-students-and-alumni.html.

Dobbins, Roland, Robert Moskowitz, Nik Teague, Liang Xia, Kaname Nishizuka, Stefan Fouant, and Daniel Migault. "Use Cases for DDoS Open Threat Signaling." Internet-Draft. Internet Engineering Task Force, March 2017. https://datatracker.ietf.org/doc/html/draft-ietf-dots-use-cases-04.

"EANS-Adhoc: FACC AG / UPDATE: FACC AG—Cyber-Fraud," January 20, 2016. http://www.facc.com/en/content/view/full/3958.

Ferguson, Paul. *Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing*. Request for Comments 2827. RFC Editor, 2000. doi:10.17487/rfc2827.

Gorrell, Mike. "Salt Lake County Data Breach Exposed Info of 14,200 People." *Salt Lake Tribune*. Accessed April 23, 2017. http://www.sltrib.com/home/3705923-155/data-breach-exposed-info-of-14200.

Herley, Cormac. "Unfalsifiability of Security Claims." *Proceedings of the National Academy of Sciences of the United States of America* 113, no. 23 (June 7, 2016): 6415–20. doi:10.1073/pnas.1517797113.

"How Does Micro-Segmentation Help Security? Explanation." *SDxCentral*, March 8, 2016. https://www.sdxcentral.com/sdn/network-virtualization/definitions/how-does-micro-segmentation-help-security-explanation/.

Khandelwal, Swati. "World's Largest 1 Tbps DDoS Attack Launched from 152,000 Hacked Smart Devices." *Hacker News*. Accessed April 25, 2017. http://thehackernews.com/2016/09/ddos-attack-iot.html.

Krupp, Johannes, Michael Backes, and Christian Rossow. "Identifying the Scan and Attack Infrastructures Behind Amplification DDoS Attacks." In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 1426–37. CCS '16. New York, NY, USA: ACM, 2016. doi:10.1145/2976749.2978293.

Leary, Judy. "IRS Data Breach Grows." *IdentityForce*®, February 29, 2016. https://www.identityforce.com/blog/irs-data-breach-more-taxpayers-affected.

———. "UCF Data Breach." *IdentityForce*, February 8, 2016. https://www.identityforce.com/blog/ucf-data-breach-affects-63000.

———. "Verizon Enterprise Data Breach." *IdentityForce*, March 25, 2016. https://www.identityforce.com/blog/verizon-enterprise-data-breach.

Marsh, Jennifer. "How to Detect and Analyze DDoS Attacks Using Log Analysis." *Loggly*, March 2, 2016. https://www.loggly.com/blog/how-to-detect-and-analyze-ddos-attacks-using-log-analysis/.

McKinney, Matt. "Data Breach Exposes Information on More than 3,000 TCC Employees." *Virginian-Pilot*. Accessed April 23, 2017. http://pilotonline.com/news/local/crime/data-breach-exposes-information-on-more-than-tcc-employees/article_6ab72a2f-52a0-533e-8060-a2d245c7f151.html.

Mortensen, Andrew, Flemming Andreasen, Tirumaleswar Reddy, Christopher Gray, Rich Compton, and Nik Teague. "Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture." Internet-Draft. Internet Engineering Task Force, October 2016. https://datatracker.ietf.org/doc/html/draft-ietf-dots-architecture-01.

Mortensen, Andrew, Robert Moskowitz, and Tirumaleswar Reddy. "Distributed Denial of Service (DDoS) Open Threat Signaling Requirements."

Internet-Draft. Internet Engineering Task Force, March 2017. https://datatracker.ietf.org/doc/html/draft-ietf-dots-requirements-04.

Muncaster, Phil. "Every Voter in Philippines Exposed in Mega Hack." *Infosecurity Magazine*, April 8, 2016. https://www.infosecurity-magazine.com/news/every-voter-in-philippines-exposed/.

"1 Billion Yahoo Accounts Compromised in Data Breach | IdentityForce." Accessed April 23, 2017. https://www.identityforce.com/blog/one-billion-yahoo-accounts-compromised-new-data-breach.

"Premier Healthcare Faces Possible Data Breach That Could Affect 200,000 Patients." *Healthcare IT News*, March 9, 2016. http://www.healthcareitnews.com/news/premier-healthcare-faces-possible-data-breach-could-affect-200000-patients.

Richter, Andy, and Jeremy Wood. *Practical Deployment of Cisco Identity Services Engine (ISE): Real-World Examples of AAA Deployments*. 1st edition. Waltham, MA: Syngress, 2015.

Rigney, Carl. *RADIUS Accounting*. Request for Comments 2866. RFC Editor, 2000. doi:10.17487/rfc2866.

Rubens, Allan, Carl Rigney, Steve Willens, and William A. Simpson. *Remote Authentication Dial In User Service (RADIUS)*. Request for Comments 2865. RFC Editor, 2000. doi:10.17487/rfc2865.

Santuka, Vivek, Premdeep Banga, and Brandon James Carroll. *AAA Identity Management Security*. 1st edition. Indianapolis, IN: Cisco Press, 2010.

"Securing Apache, Part 8: DoS & DDoS Attacks." *Open Source for You*, April 1, 2011. http://opensourceforu.com/2011/04/securing-apache-part-8-dos-ddos-attacks/.

Siciliano, Robert. "Yahoo Data Breach: Almost 500 Million Affected." *IdentityForce*, September 22, 2016. https://www.identityforce.com/blog/yahoo-data-breach-almost-500-million-affected.

Simeonovski, Milivoj, Giancarlo Pellegrino, Christian Rossow, and Michael Backes. "Who Controls the Internet?: Analyzing Global Threats Using Property Graph Traversals." In *Proceedings of the 26th International Conference on World Wide Web*, 647–56. WWW '17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017. doi:10.1145/3038912.3052587.

"Understanding and Mitigating NTP-Based DDoS Attacks." *Cloudflare Blog*, January 9, 2014. http://blog.cloudflare.com/understanding-and-mitigating-ntp-based-ddos-attacks/.

Vacca, John R., ed. *Network and System Security*. 2nd edition. Waltham, MA: Syngress, 2014.

Vempati, Jagannadh, Mark Thompson, and Ram Dantu. "Feedback Control for Resiliency in Face of an Attack." In *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, 17:1–17:7. CISRC '17. New York, NY: ACM, 2017. doi:10.1145/3064814.3064815.

White, Russ, and Bora Akyol. *Considerations in Validating the Path in BGP*. Request for Comments 5123. RFC Editor, 2008. doi:10.17487/RFC5123.

Review Questions

1. Find a real-life data breach; identify the threat actor, the exploit, the vulnerability, the assets, and the risks.
2. While many engineers argue validating the path calculated through routing will improve the security of the traffic flowing over the path, there appear to be a number of problems with this approach, some of which are outlined in RFC5123. Explain what you think is the relationship between securing routing and securing traffic flowing through the network.
3. RADIUS is called out as a form of AAA in the text; find one other form of AAA and briefly describe it.
4. What is an intrusion detection system? Describe what it does.
5. What is a data exfiltration detection system? Describe what it does.
6. If unicast RPF would be effective at blocking attacks in the global Internet, why do so few providers offering transit connectivity in the global Internet deploy it?

This page intentionally left blank

Chapter 22

Network Design Patterns

Learning Objectives

When you are finished reading this chapter, you should understand:

- The relationship between business problems and network design
- The different network ownership models
- How network investment cycles relate to waste
- What a choke point is and how it is used in network design
- How hierarchical design is used
- Common network topologies
- The concepts of planar, nonplanar, and regular topologies

After "yet another outage," a large service provider called on its primary vendor with a demand: *send these 16 (specific, by name) people to our office for one week so they can redesign our network and prevent this kind of outage from ever happening again*. On hearing of the plan, one of the 16 engineers chosen for the "field trip" convinced the vendor's management not to send such a large group of designers in to rebuild this network. The reason? *When you get 16 designers in a room, what you will have is 1 person drawing on the white board, and the other 15 erasing*.

This story encapsulates one of the primary truths about network design: there is no *one right way* to design a network. Of course, there are *better* designs and *worse designs*, but the protocols and systems that networks are built out of are designed

to be very forgiving in the face of imperfect conditions. If they were not, networks would be very fragile, failing completely with the first network device or link failure. So, given you can pretty much "slap stuff together" and "make it work," what makes a design better or worse? This chapter will focus on answering this question.

Getting to the answer, however, will require working through a range of ideas and concepts, beginning with the problem space itself.

The Problem Space

What problem are designers trying to solve? While this might seem like an obvious question, it is far too often forgotten in the design process. What generally happens is this:

1. Engineers are given a set of objectives to fulfill.
2. A rough sketch is made of two or three possible solutions.
3. These two or three solutions are considered and compared at a technical level.
4. A solution is chosen, and the equipment and configurations are determined and deployed.

This is not a *bad* process, necessarily; rather, this process tends to take some points that should be considered in every step and push them into a "corner." Once the first stage is passed, it is *never revisited*. Just like security is often left to the last, the larger questions of network design are often pushed to the first part of the process, and forgotten about when the real "geek-level stuff" of selecting equipment and building configurations arrives. It is better to use the basic problem set as a backdrop in *every* stage of the design.

Solving Business Problems

The network exists to solve business and real-world problems. When you are in the thick of choosing which forwarding engine is better, or which network software package has the greatest number of features, what is the fundamental question that network design needs to answer—the one that should be kept in mind at every stage of the design process?

What is the least expensive and most flexible way to provide transport for the applications required to solve a real-world problem?

This single question has many components, of course; it is worth looking at some of them in more detail.

What does the least expensive really mean? This is actually a difficult question with many different facets—and any answer given is probably the result of a good deal of crystal ball gazing and reliance on assumptions. Some expenses that engineers often remember to include are

- **Hardware:** The cost of the actual, physical devices, cabling, power, racks, and other gear required to physically build the network. This should include the cost of physical devices to connect the network through providers, for instance, spare equipment and tools.
- **Software:** The cost of licenses for the network operating system, routing stack, monitoring tools, and any ongoing maintenance costs.
- **Services:** The cost of having a technical assistance center to call, design services, offsite backup services, etc.

These kinds of costs, both in terms of capital expenses (CAPEX) and operational expenses (OPEX) are generally well understood, even if they are difficult to predict. A number of costs, however, are not generally included in any sort of planning, nor are they even well understood.

Many of these can be described as different forms of opportunity costs, and they often revolve around difficult operations and network modification driven by design complexity. Specifically, during the time it takes to bring the network up, modify the network, or troubleshoot and repair the network when there is a failure, the network is either not operational or is not fully supporting the business.

The key problem with opportunity costs is how difficult they are to measure. Content providers, for instance, tend to have very good systems for measuring the impact of a less than optimal network, because it is actually possible to translate slower page speed (for instance) to reduced engagement or reduced product purchases (the conversion rate). The key is to learn the specific business that the network you are building supports, and find some way to measure the impact of a network running in a less than optimal way. This kind of feedback comes back to the network engineering team as a set of unquantifiable complaints far too often. As a network engineer, you cannot wait for your customers to complain to find creative ways to measure how effective the network is at supporting the business. You must be proactive in this area, especially if you expect to convert the network from a cost center into a strategic asset for the company.

What does the most flexible really mean? Networks tend to be sold as "multiple-use systems," meaning they can support any application and any change

in the business. In real life, however, this is rarely true. There are two enemies of flexibility in network design: *ossification* and *the forklift*.

Ossification is the process of hardening (petrification) that turns wood, bones, and even sacks of flour into stone. What was originally a pliable, changeable object becomes something difficult to modify, and prone to massive, degenerative, and often unexpected failure states. Networks ossify over time, as well, in several ways:

- New systems are layered on top of old, creating interaction surfaces, which are difficult to understand, difficult to troubleshoot, and difficult to replace.
- The network is successively tuned to meet the requirements of an ever-expanding array of applications and requirements. On the other hand, as older applications and requirements are removed, the corresponding tweaks and nerd knobs are not removed from the network. The resulting accretion of configuration often goes undocumented, and also creates a lot of state and unnecessary interaction surfaces in the network.
- Network architectures are often built at the intersection of vendor designs and products, "industry best practices," and business needs. Vendors are (understandably) constantly trying to build future sales into current sales, and to convert the entire vertical to their product. At the same time, business leadership is often out reading reports and articles telling them about the latest trends and offerings, and then asking, "Why don't we deploy these?" The result is a crucible of politics, trends, and practical considerations, often resulting in a less than optimal design, particularly in the area of flexibility.

Ossified systems are generally fragile; while they appear to be "rock solid" from the outside, they break unexpectedly, and far too easily, in the face of what might appear to be small amounts of change or pressure.

The forklift is a related but different problem in building flexible networks. The first half of the forklift problem is the general tendency to build networks out of appliances containing all the hardware and software in a single appliance; a router is purchased as a single appliance containing the routing protocols, the forwarding software and hardware, the power and cooling components, etc. This tie between the various parts of the system tends to result in strong vertical integration. While many systems from many different vendors will work together in a significant way, most vendor-driven appliance-based systems will enable special features and modes of operation only when their appliances are used in a single vendor environment.

The process of upgrading such systems usually involves a forklift—hence the industry shorthand a *forklift upgrade*. To change the control plane architecture, the entire device must be replaced. If the control plane acts as an integrated whole in

some way, with special features available just on a small range of devices, the entire network must be forklift upgraded. This is a challenging situation, to say the least. Networks designed for the maximum flexibility are designed, instead, using some form of disaggregation. Figure 22-1 illustrates.

Figure 22-1 illustrates a number of different options in terms of *network ownership*:

- If you buy everything needed to build the network, including using vendor-proprietary extensions to the control plane, then your network is vendor driven. In this case, if the vendor changes its product architecture, or the philosophy behind its control plane in order to support some new network architecture, then you must change your network design to follow the vendor.
- You can, instead, buy all your hardware and software from vendors, but use open-standards-based protocols to interconnect the equipment and build a network. Theoretically, this should make your network vendor independent (although it does not always work out this way in real life). When you are deploying this kind of network, it is important to keep up with new standards, and whether new features implemented by a single vendor are implemented in a way that allows routers from multiple vendors to successfully interoperate.
- In the disaggregated model, you might purchase a network operating system and hardware from one or more vendors, and rely on an open source (whether "tweaked" for your network or not) implementation of the routing stack. There are various other "modes" within this model, as well, such as relying on an open source network operating system as well as an open source routing stack, or perhaps relying on an open source network operating system and creating your own control plane.
- In the roll your own model, you are just buying hardware from a vendor—and perhaps you are even giving the vendor a set of standards to build hardware to.

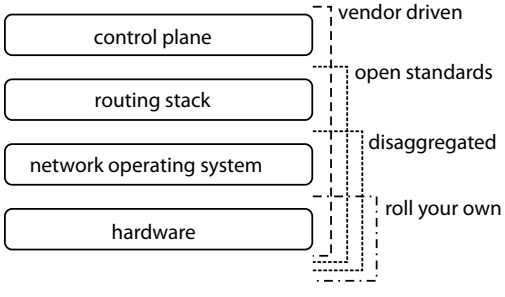


Figure 22-1 *Network ownership*

> **Note**
>
> In the real world, the model that most network operators use might be called old and moldy, which means you buy your equipment and leave it in place until it falls apart. This is the "normal" way of managing network growth and management over time for a lot of operators.

None of these models are as clear cut as Figure 22-1 implies; there are many different gradations between these various models. The important point for you, as a network engineer, designer, or architect, is to understand there are many other options than simply purchasing from a vendor. It is important to choose a model that provides the most value and flexibility for your company, rather than simply relying on what other companies are doing, or what has been done before.

Figure 22-2 illustrates another problem in the area of flexibility and fit to business in network design.

Figure 22-2 illustrates company and network size (or capacity) overlaid on top of one another. Networks, particularly appliance-driven networks, tend to be upgradable only in large chunks, and often through some form of forklift upgrade. The business, on the other hand, tends to grow in spurts, with occasional retrenchment. When the business size and network capability are mismatched—which is almost all the time in the real world—one of two situations is occurring:

- The network is undersized, which holds the business back from being able to support as many customers, or as much operational load; this creates an opportunity cost.

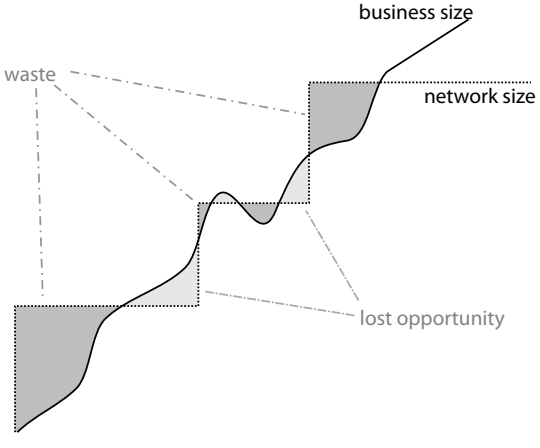


Figure 22-2 *Waste and lost opportunity in network scaling flexibility*

- The network is oversized, which means money is being spent unnecessarily—money invested in infrastructure that could have (most likely) been invested in some other way more profitably. This is another form of opportunity cost.

The more flexible a network design is, the more the network engineering staff will be able to make the business size and the network capacity track closely. This can reduce waste and lost opportunity.

Translating Business Requirements into Technical

Understanding the business side and translating business requirements into technical solutions are two different things. What tools does the network designer have to apply these business problems to network design? Modularity is the primary tool; just as nature often places a choke point between complex systems, network designers use choke points to separate complexity from complexity in network designs. Figure 22-3 illustrates this concept.

The idea behind modularity is to split up a single problem into multiple pieces, solving each piece separately, and then using a set of connectors to allow information to flow edge to edge in the overall system. This concept is used almost everywhere in network engineering; you might recognize at least the following:

- Layering protocols into functional and topological units; in Day's RINA model, there are two protocols, each with two functions, layered on top of one another. There are a pair of such protocols across each link, between each pair of hosts, and between each pair of applications.
- Building layers of virtual topologies on top of a single physical topology. Complexity is controlled by carrying a subset of the topology and reachability information in each virtual topology, and restricting policy to the destinations attached to the virtual topology.

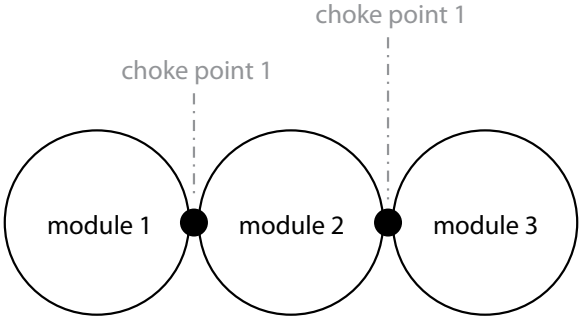


Figure 22-3 *Modularity*

- Breaking up a network into flooding domains in a link state protocol, and patching the network back together with L1/L2 intermediate systems, or Area Border Routers. The flooding domain boundary represents the boundary at which the topology information is summarized, and reachability information may potentially be.

Information hiding and modularity are closely related concepts. Any time information is being hidden, modules are being created. Just considering network design, modularizing a network can offer a number of solutions for business problems. Some specific examples:

- Choke points provide a point at which information can be hidden to control the scope and speed of state in the control plane. This, in turn, allows the network to scale.
- Choke points provide a point at which packets must move, providing a convenient place to implement packet forwarding policy, such as Quality of Service marking, security-focused filtering, etc.
- If modules can be sorted into "classes," with each class having repeatable designs and configurations within the module, then building at least some parts of the network can be simplified into deploying the right module for a particular purpose. For instance, if modules 1 and 3 in Figure 22-3 are both campus networks of a similar size and scope, then they can be built in the same way, saving design and deployment effort. Repeatable modules also help make the network scale more closely with the business, as modules can be added or removed as complete units as needed.
- Modules can be sorted into "generations," with newer designs replacing modules using older modules over time. This can help reduce the impact of the forklift upgrade problem, and hence allow the network to be more flexible in the face of business changes.

With all the positive points around modularization, are there negatives? In all areas of network design, it is important to remember:

If you have not found the tradeoff, you have not looked hard enough.

Although you have probably read this statement several times in this book, it applies to so many areas of network design and operation, it is well worth repeating. Several rules of thumb might be helpful.

The first is the size of the modules. If you make the modules too large, you are (probably) reducing the repeatability, not giving yourself enough places to hide information (which harms scaling), and not giving yourself enough places to insert policy into the network. If you make the modules too small, you reduce the effectiveness of the information hiding process. Finding the proper size is much like finding porridge that is just the right temperature—there is no clear-cut answer (or, perhaps a better answer, how many balloons fit in a bag?).

The second is the optimization tradeoff. Each time you hide information, you (more than likely) lose some form of optimization in the network. This is a fundamental rule, built into network and protocol design just because of the shape of reality.

The third is using network complexity as your guide in determining where to place module boundaries. In general, the best rule here is to separate complexity from complexity. For instance, if you have a large-scale spine and leaf data center fabric connected to a large partial mesh network core, it is probably best to put them into two different modules.

What Is a Good Network Design?

Finally, with this background, it is possible to return to the original question this chapter began by asking: what is a good network design? The first point should be obvious: it should fulfill the business requirements laid out previously. This means the network should provide the connectivity needed to run the business at the lowest practical cost, and it should be easily adaptable to the size and scope of the business. Flexibility inherently implies scale, as well.

A second point is this: good network designs degrade gracefully, rather than falling over a cliff. Figure 22-4 illustrates.

Any system that performs well under one set of conditions and fails to adapt to any change under less than optimal conditions is fragile; it has either ossified, or

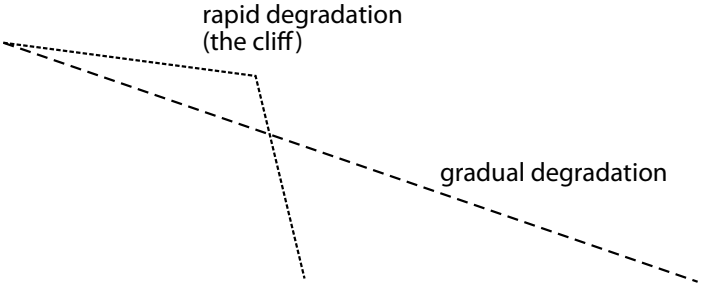


Figure 22-4 *Network degradation styles*

it was initially poorly designed. Another term sometimes applied to these sorts of systems is Robust Yet Fragile, which means these systems are apparently robust, and yet truly fragile under the right conditions.

A third point is this: good network designs allow operational staff to quickly find and repair failures. In other words, good network design interacts with the Mean Time to Repair (MTTR).

Hierarchical Design

The rules of thumb on how to break up a network into modules are a good start, but they do not give you an entire picture. While they do give you a good set of ideas around what the goals of modularization should be, they do not provide a framework grounded in an intentional, systemic view of the network. Hierarchy is one design pattern that can be overlaid on top of modularization, or rather one pattern of modularization, and is often used to build large-scale networks. The essential points of modularization are as follows:

- Break up the functionality of the network into distinct pieces.
- Build modules around each piece, or purpose.
- Connect these modules through a set of special-purpose "interconnection" modules in a roughly hub-and-spoke topology.

Figure 22-5 is used to consider the basic three-layer hierarchical design model, which is often used in medium-scale networks.

Module 1 in Figure 22-5 is the network core. The primary—really the only—function assigned to the core module is to forward traffic as quickly as possible between distribution layer modules. There should be *no* control plane policy in this core; there should only be forwarding policy aimed at differentiated forwarding rules for Quality of Service and virtualization. This module will tend to be one of the more complex in terms of transports and equipment, and is also "unique" because there is just one core, so offsetting simplification in other areas allows the core to remain maintainable.

Modules 2 through 5 are considered the distribution layer. This layer is primarily responsible for carrying traffic within a region and all control plane policy. Each of the four modules in this layer should be as similar as possible; the physical configuration, at least, should be completely repeatable within a generation of each module across the entire network. Standardizing the hardware configuration across all of the modules in the distribution layer simplifies one part of the problem, allowing the

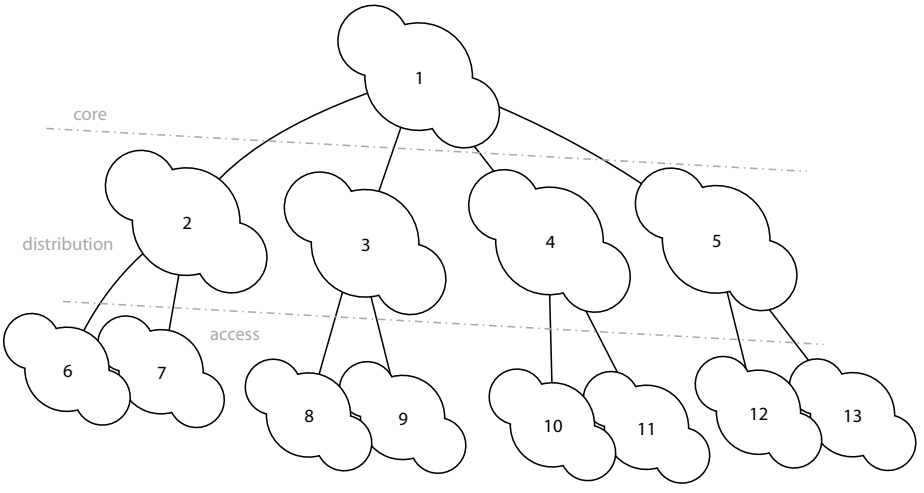


Figure 22-5 *A three-tier hierarchical design*

complexity in this layer to reside in control plane policy, which is normally a complex problem.

Modules 6 through 13 reside in the access layer. Modules in the access layer are primarily responsible for providing connections into the network. Because of this, there will likely be many kinds of modules in the access layer. For instance, there may be one kind of module for supporting campus environments, another for supporting data center fabrics, and another for supporting Internet and extranet access. Traffic classification and security access should be focused in this layer of the network. Physical and logical configurations should be as repeatable as possible between modules of the same kind within the access layer.

An alternate form of hierarchy is the two-layer hierarchy, often used in smaller networks, illustrated in Figure 22-6.

The two-layer hierarchy appears to be a three-layer hierarchy with the access layer removed—but there are other subtle changes involved, as well. The aggregation layer, modules 2 through 5, are primarily responsible for providing connectivity into the network, as well as packet filtering and classification. Security functions also tend to be focused in the aggregation layer. As with the three-layer hierarchy, these modules should be physically and logically repeatable where possible.

The core, as in the three-layer model, is focused on forwarding traffic at high speed. The missing piece seems to be control plane policy; in the core/aggregation model, control plane policy is implemented along the edge between the core and aggregation modules.

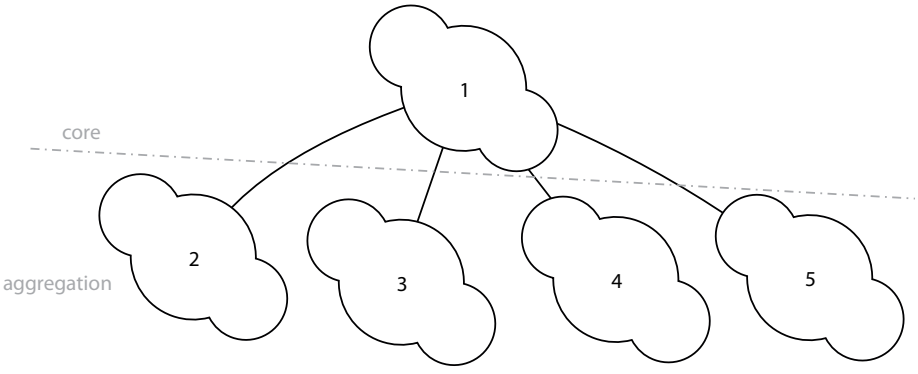


Figure 22-6 *A two-layer hierarchy*

Figure 22-7 shows an example of a recursively layered hierarchy, often used in very large-scale networks.

Figure 22-7 illustrates what appears to be a simpler core/aggregation hierarchy. Looking more closely at module 2 in the aggregation layer, however, exposes an internal core/aggregation hierarchy, as well. The layer functions are the same; the core within module 2 (6) is focused on forwarding traffic quickly between the different aggregation modules (7 through 9) within module 2; the aggregation modules within module 2 are focused on providing connection, security, and classification; and control plane policy is imposed at the core/aggregation edge. This may appear to be a slightly modified version of a three-layer hierarchy, but there are several important differences:

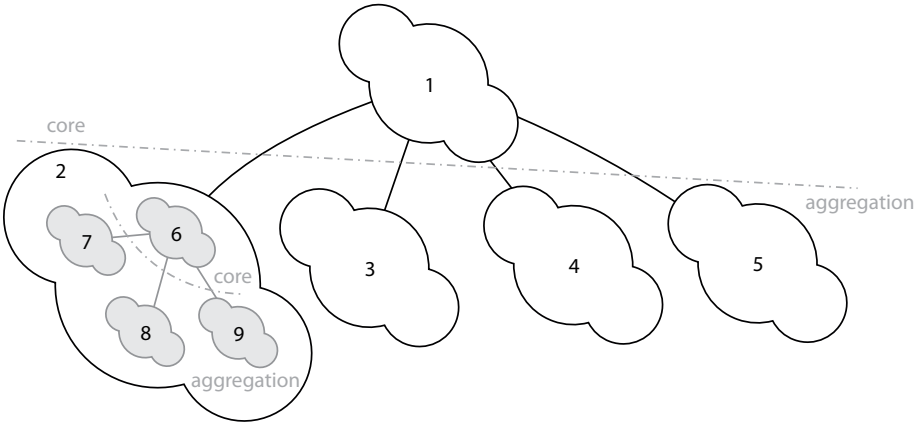


Figure 22-7 *A recursive hierarchy*

- Regional policy is handled regionally, providing more flexibility in this area. Of course, this also means the modules are less likely to be logically repeatable, so there is more complexity in this kind of design.
- It is possible to build layers within layers more than once; module 6 may contain core and aggregation modules, as well. Because of this, the recursive layering architecture is a very powerful paradigm for building and understanding network topologies.

Common Topologies

Another basic pattern in network design is the network topology. While it might seem there would be an infinite number of possible topologies, there are a few basic kinds that this infinite variety will fit into. Control planes tend to converge on any particular topology type based on the basic components—rings, meshes, and triangles. This section will consider a few basic topology types and some of their characteristics.

Ring Topologies

Ring topologies are among the simplest to design and understand. They are also the least expensive option, especially when long haul links are involved, so they tend to predominate in wide area networks.

Scaling Characteristics

Ring topologies have been scaled to large sizes; the additional cost to add a node is minimal. Generally, one new router (or switch), moving one circuit, and adding another new circuit is all that is needed. With careful planning, the addition of a new node into the ring can be accomplished without any real impact to overall network operations. Figure 22-8 depicts adding a node to a ring.

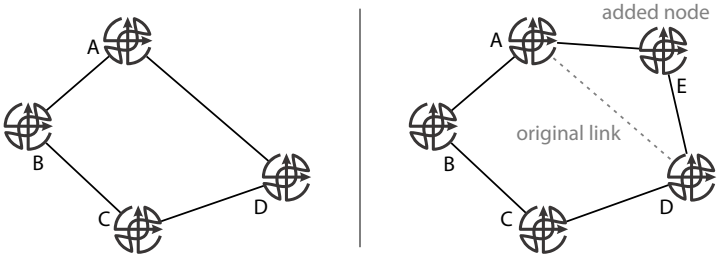


Figure 22-8 *Adding a node to a ring*

Adding new nodes to the ring increases the total hop count through the ring (from 4 to 5 in this case), and it does spread the available bandwidth across more devices. However, every device on the ring still has just two neighbors; this constant neighbor count is much of the secret behind the scaling properties of ring networks.

As ring size increases, it becomes difficult to manage Quality of Service and optimal traffic flows. Figure 22-9 illustrates.

In Figure 22-9, assume F has a voice over Internet Protocol (VoIP) stream connected to H, while G has some large file transfer (perhaps a complete backup). One of these streams requires very small amounts of bandwidth, low delay, and small amounts of jitter; the other requires large amounts of bandwidth but can tolerate a lot of delay and jitter. These two streams, however, share the [D,E] link. One option may be to force traffic along the "back side" of the ring to avoid the problem of having two kinds of traffic on the same link, but this would require tunneling one of the two streams using something like Multiprotocol Label Switching (MPLS). Another option may be to use a well-designed Quality of Service (QoS) mechanism to ensure the two streams can coexist. Either of these two solutions, however, adds complexity into the control plane and forwarding process. In terms of complexity, then, the ring topology requires just two neighbors per device, but it can require a lot more traffic engineering work to support all the requirements that applications place on the network.

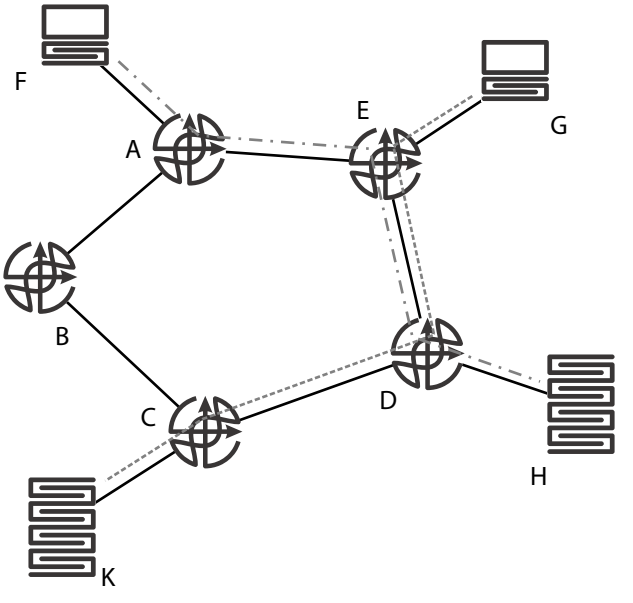


Figure 22-9 *Traffic engineering in a ring topology*

Resilience Characteristics

Rings can withstand a single failure anyplace in the ring; any two failures will cause the ring to split. However, a single failure can make the kinds of traffic engineering problems considered in the preceding section much more difficult to manage; a single failure essentially turns a ring into a bus, with just one path between each pair of connected devices. The "middle segment," in this case, will be a bandwidth choke point in a very negative way.

Convergence Characteristics

Convergence in ring topologies lays the foundation for truly understanding the convergence of every other network topology ever designed or deployed. After you understand the principles of routed convergence in a ring topology, it's simple to apply these same principles to quickly understand the convergence of any other topology you might encounter.

In other words, pay attention!

The crucial point to remember when considering how any control plane protocol—routed or switched—will operate on a particular topology is to think in terms of the "prime directive": *thou shalt not loop packets!* This aversion to looping packets explains the convergence properties of ring topologies. Consider the rings in Figure 22-10.

While it is common to think of routing as using *every* link in the network, protocols build a spanning tree per destination. For any given destination, specific links are blocked out of the path to prevent a packet forwarded toward the destination from looping in the network. In the two rings shown in Figure 22-10, the links over which a routing protocol will not forward packets toward 2001:db8:3e8:100::/64 are marked.

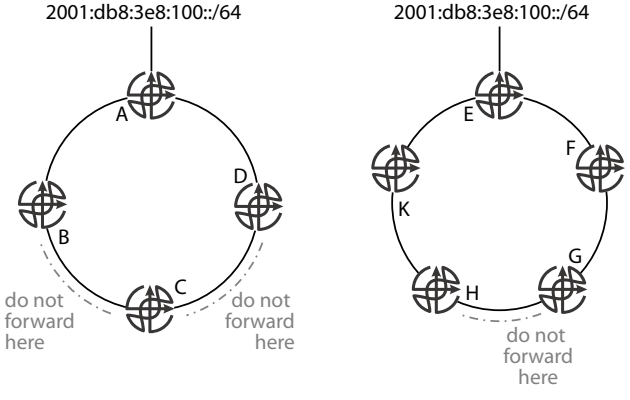


Figure 22-10 *Convergence in a ring topology*

In the case of the four-hop ring toward 100::/64:

- The link between B and C appears to be unidirectional toward B.
- The link between C and D appears to be unidirectional toward D.

Why are these links blocked in this way by the routing protocol? To follow the prime directive—thou shalt not loop!

- If a packet destined to 100::/64 is forwarded from D to C, the packet will loop back to D.
- If a packet destined to 100::/64 is forwarded from B to C, the packet will loop back to D.

In the case of the five-hop ring toward 100::/64, the link between G and H appears to be completely blocked. But why should this be? Suppose a packet destined to some host in 100::/64 is forwarded from G to H. This packet will be forwarded correctly to F, then to E, and finally to the destination itself.

But what if G is forwarding traffic to H for 100::/64, and H is also forwarding traffic to G for 100::/64? A permanent routing loop results. This means

- If the link between A and D fails, D has no way to forward traffic toward 100::/64 until the routing protocol converges.
- If the link between E and K fails, H has no way to forward traffic toward 100::/64 until the routing protocol converges.

Generalizing Ring Convergence

Why is all this so important to understand? Because virtually every topology you can envision with any sort of redundancy is, ultimately, made up of rings (full mesh designs are considered by many to be an exception, and Clos fabrics are exceptions). To put it another way, virtually every network topology in the world can be broken into some set of interconnected rings, and each of these rings is going to converge according to a very basic set of rules:

- Every ring has, for each destination, a set of links not used to forward traffic.
- The failure of any link or node on a ring will cause traffic to either be dropped (distance vector) or looped (link state) until the routing protocol converges.

Given the speed at which a routing protocol can converge is directly related to the number of routers notified of a particular topology change, and hence the number of routers that must recalculate their best paths to any given destination, a third rule for control plane convergence on a ring is

- The larger the ring, the more slowly the routing protocol will converge (and thus stop throwing packets on the floor or resolve the resulting microloop).

These three rules apply to virtually every topology you encounter. Find the rings, and you have found the most basic element of network convergence.

Mesh Topologies

While ring topologies are the cheapest to deploy and the simplest to scale, full mesh topologies tend to be the most expensive to deploy and the most difficult to scale. Full mesh topologies, however, are simpler to understand (in terms of convergence and scaling) than ring topologies. Figure 22-11 illustrates a full mesh topology.

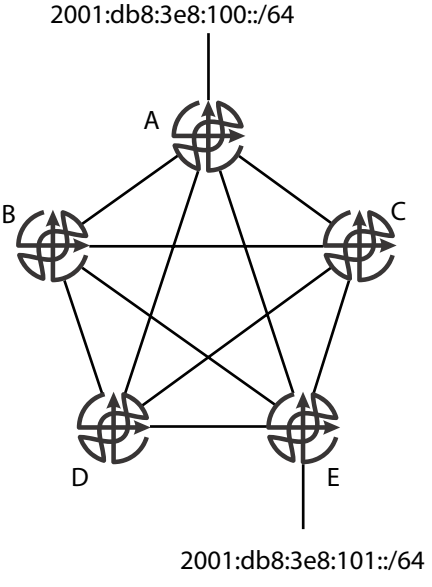


Figure 22-11 *A full mesh topology*

There are ten paths from 2001:db8:3e8:101::/64 to 2001:db8:3e8:100::/64:

1. [E,A]
2. [E,C,A]
3. [E,D,A]
4. [E,B,A]
5. [E,D,B,A]
6. [E,D,C,A]
7. [E,C,B,A]
8. [E,C,D,A]
9. [E,B,C,A]
10. [E,B,D,A]

Traffic engineering techniques can be used to direct specific traffic onto any of these paths, allowing the network designer to design traffic flows for optimal performance. The number of paths through the network and the number of links required to build a complete mesh between a set of nodes are given by

$$N(n-1)/2$$

For the five-node network in Figure 22-11, this is

$$5(5-1)/2 = 10$$

This property of full mesh networks, however, also points to the weaknesses of this topology: scale and expense. These two weaknesses are closely related. Each new node added to the network means adding as many links as there are nodes already in the network. Adding a new node to the network in Figure 22-11 would mean adding five new links to bring the new node fully into the mesh.

Each new link added means not only a cable, but also a new port used, and new ports mean new line cards, and so on. Each new link also represents a new set of neighbors for the control plane to manage, increasing memory utilization and processing requirements. There are protocol-level techniques that can reduce the control plane overhead in full mesh topologies, if they are properly managed and deployed. Open Shortest Path First (OSPF) and Intermediate System to Intermediate System

(IS-IS) both have the capability to build a mesh group, which treats the full mesh topology similar to a broadcast network; a small number of routers at the edge of the mesh are designated to flood topology information onto the mesh, while the remainder of the attached routers passively listen.

The one place where network engineers often encounter full mesh topologies is in virtual overlays, particularly in deployments where traffic engineering is a fundamental part of the reason for deploying the virtual overlay. Although the port and link costs are reduced (or eliminated) when building a full mesh of tunnels, the cost of managing and troubleshooting a full mesh remains.

A more commonly used mesh variant is a partial mesh topology. In a partial mesh, only some nodes are connected to one another, generally based on measured traffic patterns and perceived resilience requirements. Partial mesh topologies often reduce, in convergence and scaling terms, to a set of interacting ring topologies. Each "ring" within the partial mesh will scale and converge in the same way as the description of ring topologies already covered.

Hub-and-Spoke Topologies

Hub-and-spoke topologies are built in just the way they sound; there is one or more hub router that is connected to a much larger number of remote routers. Figure 22-12 illustrates two such topologies.

In Figure 22-12, there is just one path from 2001:db8:3e8:101::/64 and 2001:db8:3e8:100::/64, along [B,A]. If this path fails, connectivity fails between these two networks; this is called a single-homed network. To prevent a single point of failure from causing a complete outage for a particular site or application, many hub-and-spoke networks are designed with two hub routers, as shown in the network on the right side of Figure 22-12; this is called a dual-homed network. Special

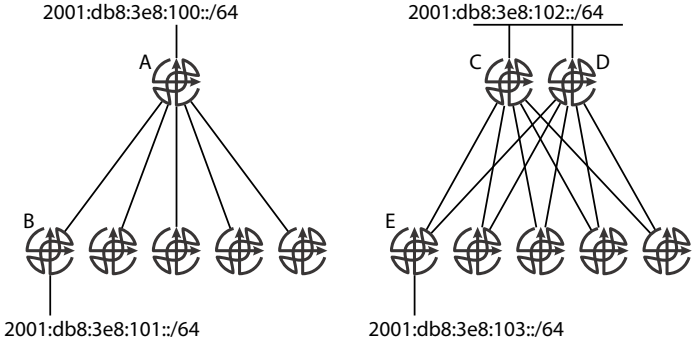


Figure 22-12 *Two hub-and-spoke topologies*

techniques are often used to scale such networks to support thousands of remote sites, such as

- Sending the remote site a minimal amount of routing information, such as just a default route.
- Reducing neighbor state toward these remote sites; for instance, Open Shortest Path First has the concept of a demand circuit, which allows the hub router to advertise its routing information once, blocking the periodic reflooding normally required to synchronize the database.
- Not calculating routes through the remote sites, as they should never be used to transit traffic. For instance, the Enhanced Interior Gateway Routing Protocol (EIGRP) has the ability to mark a remote site router as a stub, which blocks calculation of alternate paths through the remote site. OSPF has the ability to mark a route through a remote site with the maximum metric, which discourages routing through the remote site.

Without such special techniques, a dual-homed remote site will converge like a triangle; with them, it will converge more like a single-homed remote site. Because of the scaling, configuration, and management difficulties involved with managing large-scale hub-and-spoke networks, many such networks are now built using different options, such as

- A service-provider-provided service, where the hub and remote routers are actually managed by a service provider, and the customer receives the correct routing information and packets routed through the service provider network. This transfers the entire management load from the customer to the service provider.
- A Software-Defined Wide Area Network (SD-WAN) solution, which may be provided by a service provider, or installed and managed by the network operator. These services operate "over the top" of the standard Internet, using tunnels to build a virtual hub-and-spoke or full mesh network.

Planar, Nonplanar, and Regular

Network topologies can be described in terms of their properties, as well as the shape of the topology. Three important concepts are planar, nonplanar, and regular.

Planar topologies can be described using a single plane; this means links do not cross in a way that forces one link to "hop" over another link. In a nonplanar topology, at least two links will cross no matter how the topology is arranged. Figure 22-13 illustrates the difference between these two concepts.

Four networks are shown in Figure 22-13, marked A, B, C, and D. Network A is a planar topology; there are no points at which two links cross, and hence would require one link "jumping" over the other—or rather requiring a second plane to accurately represent. The topology in B is a nonplanar topology.

When examining networks to discover if they have a planar or nonplanar topology, try rearranging the links to see if they can be moved so that no two links will cross or overlap. For instance, network C in Figure 22-13 appears to be a nonplanar design, because of the two links crossing at the gray dashed circle; however, the same network is illustrated as D, but with one of the links moved so they no longer cross. The links in B cannot be rearranged to prevent any overlap in this way.

Regular topologies have one characteristic: they are made up of smaller, repeating topologies. Figure 22-14 illustrates a fabric of four hop rings (or cubes), which is a regular topology.

In Figure 22-14, the four routers A, B, D, and E are a small four-router loop within the same topology. Because any other four-router loop that you can pick out can replace any other four-router loop in the network, this is a regular topology; any set of four routers could be moved anyplace else in the network without changing the

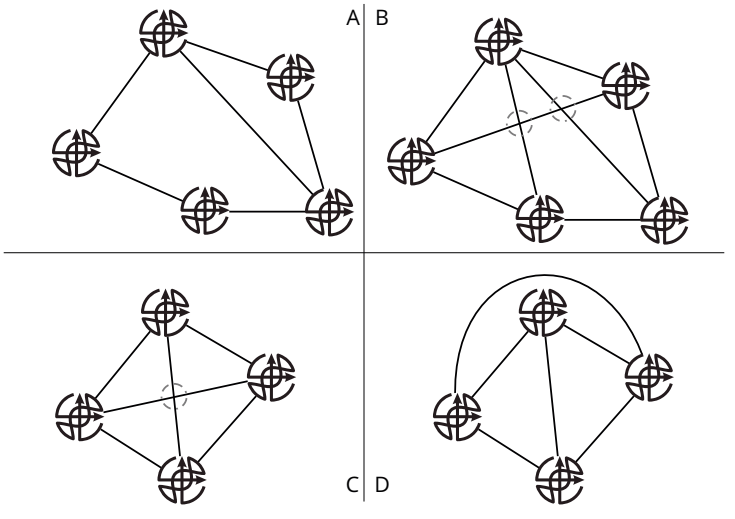


Figure 22-13 *Planar and nonplanar network topologies*

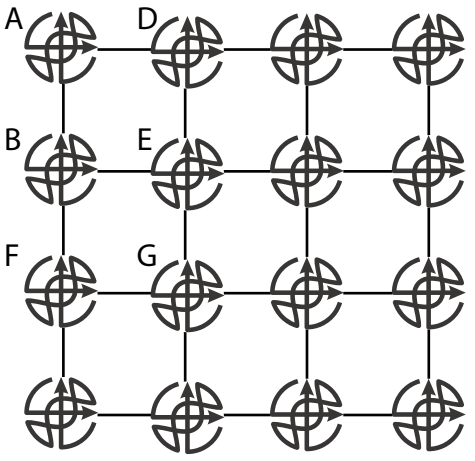


Figure 22-14 *A regular topology*

overall topology, and the network topology can be increased in size by simply replicating one piece of the topology and adding it back on.

Being able to pick out these kinds of topologies is helpful in understanding the way a particular network will converge, and what kinds of fast reroute and other options are available. It would take much more space than is available here in this chapter to draw these lessons out in detail, but being aware of these different design patterns is a good place to start.

Final Thoughts on Network Design Patterns

Network design is often treated the same way network security is—left until the last moment, done as quickly as possible, with as little thought and fuss as possible. Real design, beginning with business requirements rather than speeds and feeds, or ports and racks, is often ignored in the tyranny of the immediate. "This project needs to be done now, forget the design stuff, just get it working." This is the path to technical debt and—ultimately—crashed networks and failed businesses.

Proper network design needs to take a systemic view. This chapter, although a short overview, provides you with some of the basic mindsets and tools you need to start thinking through design problems and solutions. The next chapter will continue examining design topics by considering resilience and redundancy.

Further Reading

Oppenheimer, Priscilla. *Top-Down Network Design*. 3rd edition. Indianapolis, IN: Cisco Press, 2010.

White, Russ, and Denise Donohue. *The Art of Network Architecture: Business-Driven Design*. 1st edition. Indianapolis, IN: Cisco Press, 2014.

White, Russ, Alvaro Retana, and Don Slice. *Optimal Routing Design*. 1st edition. Indianapolis, IN: Cisco Press, 2005.

White, Russ, and Jeff Tantsura. *Navigating Network Complexity: Next-Generation Routing with SDN, Service Virtualization, and Service Chaining*. Indianapolis, IN: Addison-Wesley Professional, 2015.

Review Questions

1. Consider the objective of network design laid out in the text—to build the least expensive, most flexible way to provide transport. How does this goal set relate to the State/Optimization/Surface (SOS) model of managing network complexity?
2. Research an example of opportunity cost in the real world.
3. Explain why ossified systems appear to be well built and solid, but are often actually fragile.
4. Consider the disaggregated versus the vendor-independent model in terms of the State/Optimization/Surface tradeoff triad. Would either model increase state? Surfaces? In what way?
5. Give examples of when you might use a two-layer hierarchy versus a three-layer hierarchy.
6. Explain convergence in a ring topology.

This page intentionally left blank

Chapter 23

Redundant and Resilient

Learning Objectives

When you are finished reading this chapter, you should understand:

- What control plane failures and convergence look like to applications
- The different ways in which you can measure network availability
- Graceful restart and in-service software upgrades as tools to provide network resilience
- The interaction between modularization and resilience

Networks are designed to support applications, which in turn support specific business needs (or perhaps the application itself is the business). When the network is down, it obviously cannot support applications, but "down" is a rather ambiguous term. There are more kinds of "down" than "not forwarding packets at all." The question this chapter asks is

What does network resilience mean?

There are a number of tools network engineers can use to create a resilient network. Fast Reroute, Exponential Backoff, and other fast convergence technologies can make a large difference in the speed at which the network converges. Graceful restart is another set of tools engineers can use, but they are not covered in this book (see the "Further Reading" section at the end of this chapter for pointers to more

information on this topic). Redundancy, however, has always been one of the primary tools engineers of every kind have turned to, to build in resilience.

The first part of this chapter, then, will describe resilience; the second part will consider the use of redundancy to create resilience in a network.

The Problem Space: What Failures Look Like to Applications

Slow performance and complete failure are the two most common application problems associated with network failures of any kind. How does the operation of the network relate to application problems of these kinds? Figure 23-1 will be used to consider the answers to this question.

In Figure 23-1, assume there is a long-standing flow between A and F flowing across the [B,D] link. If the [B,D] link fails, the application driving the flow could see several results:

- **The flow could fail entirely.** If some form of packet or route filtering is configured at any of the four routers illustrated, a [B,D] link failure may result in a complete loss of connectivity between A and F. In this case, traffic between A and F will stop flowing, and the application will fail.
- **The end-to-end delay could change.** Once the routing protocol converges on the only other available path, [B,C,E,D], there will be two more queues, two more switches, and two more deserialization/serialization delays added to the path. The application will see this as a sudden change in the amount of delay across the network.
- **Jitter could increase.** The additional queues, serialization, and deserialization could also increase jitter through the network. Some packets will variably be delayed while the routing protocol converges, particularly if there is a microloop formed during the convergence process. The total impact will likely look like a short burst of high jitter, followed by a general increase in jitter across the path.

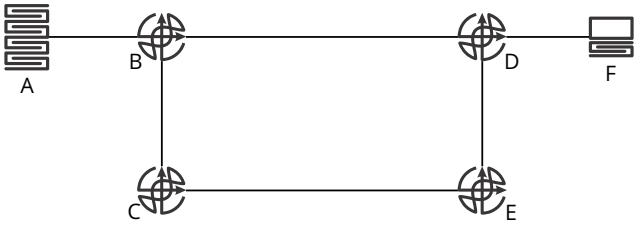


Figure 23-1 *Application Impacts of Network Failures*

- **Packets could be dropped.** Whatever packet is transmitted by B toward D just at the moment the link fails will likely be dropped.
- **Duplicate packets could be delivered.** It is possible, particularly if a microloop is formed during convergence, for one or two packets to be transmitted through the network twice. One simple example of this is if the retransmission timer at A is set very short before the failure, so packets are delayed longer than this timer during convergence, A could retransmit a packet while another copy of the same packet is already "in flight," resulting in two copies of the same packet being received at F.
- **Packets could be delivered out of order.** Consider what happens if a microloop forms between B and C during convergence. It is possible a packet is being forwarded from C toward B just at the moment the microloop resolves. If this happens, a packet transmitted by A and received by B, before the packet looping between B and C, will be forwarded by B before the packet caught in the microloop is. The earlier packet will be delivered after the later packet.

It is virtually impossible to resolve these problems in a packet switched network. In fact, while many networking technologies have been developed over the years seeking to prevent these failures from ever occurring, most of these technologies end up adding so much complexity to the network that they have an overall negative impact on network performance. Tradeoffs are the hard-and-fast rule in engineering of all kinds; network engineering is no exception.

Resilience Defined

Resilience is easy enough to understand: the network does not fail nor produce the kinds of effects in applications discussed in the previous section. Resilience needs to be measured, as well as understood, however. This section will consider several ways in which resilience is measured. Three specific measures will be considered in this section:

- The Mean Time Between Failures (MTBF)
- The Mean Time to Repair (MTTR)
- Availability

Figure 23-2 is used to illustrate these concepts.

Three different measures of resilience are shown in Figure 23–2.

MTBF is the amount of time between failures in a system. Just divide the number of failures in any slice of time into the total amount of time, and you have the MTBF for the system (during this time slice). In Figure 23-2, the MTBF is the amount of

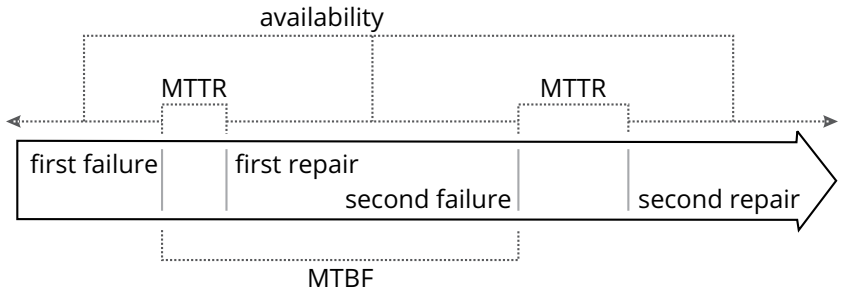


Figure 23-2 *Measuring Resilience*

time between the first and second failures. The longer the time slice (without changes in the system) you use, the more accurate the MTBF will be.

MTTR is the amount of time it takes to bring the system back up after it has failed. To find the MTTR, divide the total length of all outages by the total number of outages. To find the MTTR in Figure 23-2:

- Find the length of time between the first failure and the first repair.
- Find the length of time between the second failure and the second repair.
- Sum (or add) these two lengths of time.
- Divide this total by the number of outages—in this case, two.

Availability is the total time the system should have been operational (without counting outages) divided by the amount of time the system was not operational. To get to availability from MTBF and MTTR, you can take the MTBF as a single operational period and divide it by the MTTR, like this:

$$Availability = \frac{MTBF}{MTBF + MTTR}$$

Note

Most of the time, you'll see availability calculated by adding the uptime to the downtime, and then dividing the result by the downtime. This arrives at the same number, however, because the total uptime added to the total downtime should (in the case of networks) be the total amount of time the network should have been operational. You might also see this expressed using the idea of Mean Time to Failure (MTTF), which is just the MTBF minus the MTTR—so adding the MTTF and the MTTR should result in the same number as the MTBF.

Availability is often expressed as a number of nines; for instance, one network may have four 9s of availability, while another may have five 9s of availability. This is shorthand for the fraction of time the network is available:

- Four 9s of availability means the network is available 99.99% of the time, or is not operational for about an hour a year.
- Five 9s of availability means the network is available 99.999% of the time, or is not operational for about 5.2 minutes each year.

The concept of availability needs to be considered in light of the meaning of availability for a particular network. Does the entire network need to be down to be considered unavailable? Or perhaps service needs be unavailable to a particular set of applications of users? Or perhaps the network can be considered inoperable when some particular application (or set of applications, or set of users, etc.) suffers degraded performance. Answering these kinds of questions is very important in defining network availability.

Other "Measures"

There are other measures of network resilience for which no number can be produced, but are often just as important as the more commonly used measures. There is a good bit of humor in these measures, but the humor is backed by a good deal of serious experience.

The *Mean Time Between Mistakes* (MTBM) which measures how long, on average, it is between mistakes causing an application performance problem or network failure of any size. The MTBM is related to the complexity of configurations in the network, including how the configurations of widely dispersed forwarding devices interact. A widely used rule of thumb is called the 2 a.m. rule: if you cannot explain the configuration at 2 a.m. to a technical support engineer whose primary language is not the same as yours, it might be worth reconsidering the configuration.

The *Mean Time to Innocence* (MTTI) is the amount of time required to prove the network is not at fault for a particular application problem. Proving this often requires a lot of "before" and "after" network measurements to show none of the changes in the network could cause the observed problem. It is important to pay close attention to the various ways an application can "see" a failure in the network considered in the preceding section.

Redundancy as a Tool to Create Resilience

Perhaps the primary tool used by network engineers use to create resilience in a network design is adding redundancy. One of the primary concepts to understand when considering adding redundancy to increase resilience is the single point of failure. Figure 23-3 illustrates.

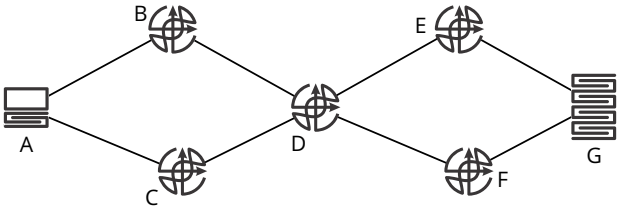


Figure 23-3 *A Single Point of Failure*

In Figure 23-3, from the perspective of A and G, the network has two redundant paths. But the redundancy is a deception; there is a single point of failure at D in the center of the network. If D itself fails, no traffic can flow between A and G. Figure 23-4 illustrates how adding a second (redundant) link would improve resilience.

In Figure 23-4, there are now two parallel paths through the network, one through [B,E], and a second through [C,F]. If both links fail on a fairly regular basis (they have similar availability), the odds are low that both links will fail at the same time. While there may be a small chance of both links failing at the same time, there is *some* chance this will happen. You can calculate the chance of both links failing at the same time, if you know the availability for each link, by calculating the combined availability of both links, using this formula:

$$a_t = \frac{1}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots}$$

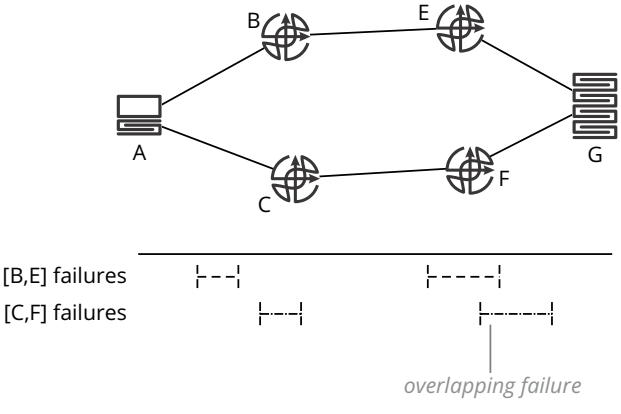


Figure 23-4 *Increasing Resilience Through Redundant Paths*

Substitute each parallel item in the network into a_1, a_2, etc., and you can calculate the availability of the entire system, a_t. This will tell you how often, over the course of a year, both links are likely to be down.

> **Note**
>
> This is called the availability calculation for parallel links of devices; it is also possible to calculate the total availability of devices or links connected in series, but this discussion is beyond the scope of this book.

There is a rule of thumb that works pretty well here without working through all of the math; it is called the halving rule. If you have two paths connected in parallel, each with a total downtime each year of 1 second, then the combined total downtime is likely to be half of the probable downtime of either link. The combined downtime of both links, then, should be around 500ms. Adding a third link will halve this number again to around 250ms. Although availability is essentially the opposite of downtime, increasing redundancy quickly produces very little increased availability. If you begin with a single link with four 9s of availability—so it would be unavailable, or down about 5 minutes in each year—adding a second link in parallel will mean the pair of links is now unavailable about 2.5 minutes each year. Adding a third link reduces the unavailable time to 1.25 minutes per year, and a fourth reduces the unavailable time to about 37 seconds.

Each link added in parallel also increases the complexity of the network from the perspective of the control plane; more adjacencies must be formed, there are more paths to calculate through, there are more sets of databases to synchronize, etc. Each of these will slow down the convergence of the control plane at least some small amount. There is no clear point where decreasing downtime by increasing parallel links will be completely offset by increasing convergence time. Experience shows two links is often optimal, three links is good in exceptional situations, and four needs a careful look at the math to ensure the additional links are not decreasing overall availability, rather than increasing it. There are some exceptions, of course, in the case of tuned protocol deployments in fabrics (such as in a data center).

Because of diminishing returns, *you simply cannot build a truly resilient system through redundancy alone*. Real resilience must be built into the entire network, and the entire stack, with each part of the system playing its own role, from applications to control planes to redundant links.

Shared Risk Link Groups

Links are one crucial place to look for single points of failure—and one place where redundancy is often introduced to increase resilience. Adding parallel links does not always increase resilience, however; Figure 23-5 illustrates.

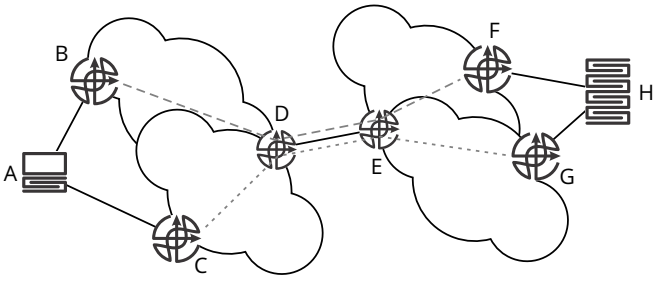


Figure 23-5 *Shared Risk Link Group (SRLG)*

In Figure 23-5, the network operator has purchased links from two different providers to provide connectivity between A and H. From the outside, these two links appear completely unrelated; they terminate in different locations, are managed by two different providers, etc. However, someplace in the transit path, both providers have leased virtual circuits over a single fiber, and both links pass through this single optical link. A backhoe pulling this single fiber out of the ground—a backhoe fade—will take both circuits down.

Any time virtual circuits are laid over a physical infrastructure, there are Shared Risk Link Groups (SRLGs). Further, SRLGs are surprisingly difficult to discover and plan around, particularly in dynamically routed packet switched networks. There are systems for calculating SRLGs within a single operator's network, which can be very useful for preventing SRLGs from causing a problem in data center fabrics or corporate networks, but they are outside the scope of this book.

Network engineers should be aware of SRLGs, and careful to plan around them where possible, particularly in relation to redundancy added to increase resilience.

In-Service Software Upgrade and Graceful Restart

Many times links are not the problem, but the rather complex network devices the links connect to. There are several solutions available to resolve problems with network devices, including

- Running multiple devices in parallel and allowing the control plane to route around failures. This essentially transfers any complexity from the device software into the network and applications running on top of the network—a valid design choice in many situations.
- Using Graceful Restart (see the "Further Reading" section at the end of the chapter for more information on graceful restart) to reduce the amount of time

required to reconverge the control plane in the case of a device reboot or some other short-lived failure. In the case of graceful restart, each device maintains its forwarding state, resynchronizing and recalculating the set of loop-free paths through the network once the control plane processes have restarted.

- Using In-Service Software Upgrades (ISSU), or hitless restarts to restart the control plane without impacting packet forwarding at all.

Graceful Restart (GR) and ISSU rely on the device being able to forward traffic in hardware while the control plane is restarting; the hardware must be able to hold a forwarding table, and forward packets, without the control plane feeding new routing information into the forwarding table. There is some amount of risk in forwarding without the control plane, as the network topology can change while the control plane is restarting, causing a loop until the control plane reconverges. This is another instance of a microloop occurring because of topology unsynchronized databases.

Each of these solutions has advantages and disadvantages—each one is applicable in some situations and not in others—but engineers should be aware of these tools and their application to the problem of building a resilient network.

Dual and Multiplanar Cores

While they are rarely used, dual plane and multiplanar cores are sometimes deployed to ensure the highest levels of availability. Figure 23-6 illustrates these two types of cores.

In Figure 23-6, each core is represented with a different kind of dashed line to make it easier to see both of the cores. In both of these core types, *everything* is different:

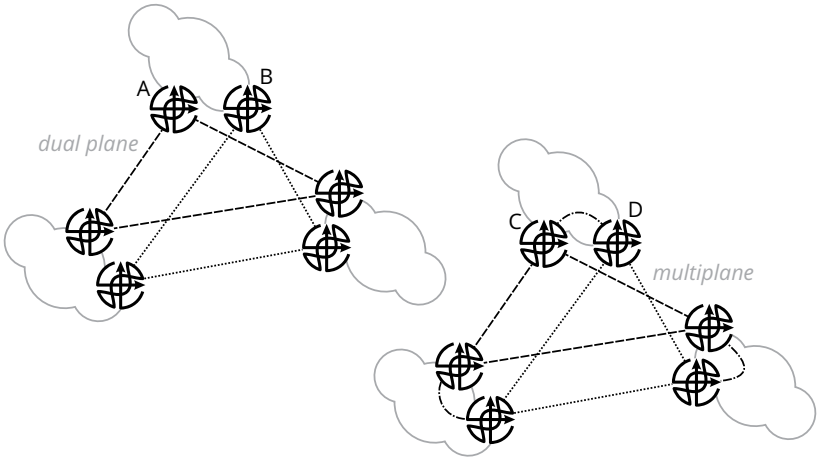


Figure 23-6 *Dual and Multiplanar Cores*

- Equipment from two different vendors, or at least two different hardware lines using two different protocol implementations, is used to prevent a single bug or kind of failure from impacting the entire network.
- Two different interior gateway protocols, such as Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS), are used, one for each core, so a problem in a single protocol cannot impact the entire network.
- Two different providers, one for each core, are contracted to supply the links between the sites.

By creating two completely separate cores, you can avoid the problems associated with monocultures, or a bug that allows a problem to propagate throughout the network.

The primary difference between these two core types is shunt links, which are represented as dash-dot in the multiplanar core illustration on the right in Figure 23-6, such as the curved link between C and D. These shunt links are set to a very high metric, so they are used to forward traffic only if there is no other path available. An exterior gateway protocol, such as the Border Gateway Protocol (BGP), is used to tie the entire network together; each site edge router, such as A, would have two routes to any given destination in the network. One would be learned through BGP over one core, and the second through BGP over the second core.

Modularity and Resilience

MTTR can be broken down into two pieces:

- The time it takes for the network to resume forwarding traffic between all reachable destinations
- The time it takes to restore the network to its original design and operation

The first definition relates to machine-level information overload; the less information there is in the control plane, the faster the network is going to converge. The second relates to operator information overload; the more consistent configurations are, and the easier it is to understand what the network should look like, the faster operators are going to be able to track down and find any network problems. The relationship between MTTR and modularization can be charted as shown in Figure 23-7.

Moving from a single flat failure domain into a more modularized design, the time it takes to find and repair problems in the network decreases rapidly, driving

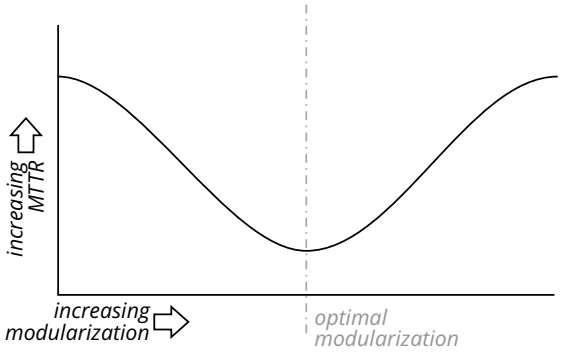


Figure 23-7 *Modularization Tradeoff with MTTR*

the MTTR down. However, there is a point at which additional modularity starts increasing MTTR, where breaking the network into smaller domains causes the network to become more complex. To understand this phenomenon, consider the case of a network where every network device, such as a router or switch, has become its own failure domain (think of a network configured completely with static routes and no dynamic routing protocol). It is easy to see there is no difference between this case and the case of a single large flat failure domain. How do you find the right point along the MTTR curve? The answer is always going to be, "it depends," but it is important to develop some general rules.

First and foremost, the right size for any given failure domain is never going to be the entire network (unless the network is really and truly very small). Almost any size network can, and should, be broken into more than one failure domain.

Second, the right size for a given failure domain is always going to depend on advances in control plane protocols, advances in processing power, and other factors. There were long and hard arguments over the optimal size of an OSPF area within the network world for years. How many LSAs could a single router handle? How fast would SPF run across a database of a given size? After years of work optimizing the way OSPF runs, and increases in processing power in the average router, this argument has generally been overcome by events.

Over time, as technology improves, the optimal size for a single failure domain will increase. Over time, as networks increase in size, the optimal number of failure domains within a single network will tend to remain constant. These two trends tend to offset one another, so most networks end up with about the same number of failure domains throughout their life, even as they grow and expand to meet the ever-increasing demands of the business.

So how big is too big? Start with the basic rules: building modules around policy requirements and separating complexity from complexity. After you get the lines

drawn around these two things, and you've added natural boundaries based on business units, geographic locations, and other factors, you have a solid starting point for determining where failure domain boundaries should go.

From this point, consider which services need to be more isolated than others, simply so they will have a better survivability rate, and look to measure the network's performance to determine if there are any failure domains that are too large.

Final Thoughts on Resilience

While redundancy is the "go-to tool" for engineers building resilience into a network, redundancy has as many negative aspects that must be managed as it does positive aspects. Resilience requires much more than redundant links and devices; it must include many other techniques that involve the entire network stack from the physical links, through the control plane, and into the application itself.

Resilience, as with security, must be built into the network, rather than bolted on at the very end.

Further Reading

Papadimitriou, Dimitri. "Inference of Shared Risk Link Groups." Internet-Draft. Internet Engineering Task Force, November 2001. https://datatracker.ietf.org/doc/html/draft-many-inference-srlg-02.

Pillay-Esnault, Padma, and John Moy. *Graceful OSPF Restart*. Request for Comments 3623. RFC Editor, 2003. https://rfc-editor.org/rfc/rfc3623.txt.

Rekhter, Yakov, and Rahul Aggarwal. *Graceful Restart Mechanism for BGP with MPLS*. Request for Comments 4781. RFC Editor, 2007. https://rfc-editor.org/rfc/rfc4781.txt.

Rekhter, Yakov, John Scudder, Srihari S. Ramachandra, Enke Chen, and Rex Fernando. *Graceful Restart Mechanism for BGP*. Request for Comments 4724. RFC Editor, 2007. https://rfc-editor.org/rfc/rfc4724.txt.

Torrell, Wendy, and Victor Avelar. "Mean Time Between Failure: Explanation and Standards." White Paper. APC. Accessed May 13, 2017. http://www.apc.com/salestools/VAVR-5WGTSB/VAVR-5WGTSB_R1_EN.pdf.

———. "Performing Effective MTBF Comparisons for Data Center Infrastructure." White Paper. APC. Accessed May 13, 2017. http://www.apc.com/salestools/ASTE-5ZYQF2/ASTE-5ZYQF2_R1_EN.pdf.

White, Russ, and Denise Donohue. *The Art of Network Architecture: Business-Driven Design*. 1st edition. Indianapolis, IN: Cisco Press, 2014.

Review Questions

1. Find or create a chart showing how much time per year three, four, and five 9s of availability translates to. Do you think these are realistic numbers? Is there anything interesting in the amount of downtime allowed at each point, or in how much the amount of allowable downtime changes?
2. Increased delay is not listed among the effects of a link failure in Figure 23-1. If you changed the network so the original link was a local circuit, and the backup path traveled over a much longer distance, would delay be something to look for in the case of the described failure?
3. Find the calculation for links or devices connected in series. What is the difference between this calculation and the calculation for devices or links in parallel?
4. Will running multiple devices in parallel, and allowing the control plane to route around failures, eliminate the need for graceful restart or ISSU in all networks? Why or why not?
5. Consider a multiplanar core with shunt links in light of the State/Optimization/Surface (SOS) model. What are the tradeoffs when deciding whether or not to include shunt links?

This page intentionally left blank

Chapter 24

Troubleshooting

Learning Objectives

When you are finished reading this chapter, you should understand:

- The concept of narrowing as a part of the troubleshooting process
- How to break a network down into components for troubleshooting
- The difference between how and what models for troubleshooting
- The importance of finding tools able to measure the problem
- The importance of models in troubleshooting
- The concept of half split and move
- The concept of technical debt

It's 2 a.m., the network is down, and the CEO is on the phone asking when it is going to be back up. The overnight job crucial to the business opening in the morning has failed, and the company stands to lose millions of dollars if the network is not fixed in the next hour or so. Almost every network engineer has faced this problem at least once in his career, often involving intense bouts of shouting (and/or screaming) intermixed with panicked attempts to find the root cause and fix it.

And yet troubleshooting is a skill that is hardly ever taught. There are a number of computer science programs that do include classes in troubleshooting, but these tend to be mostly focused on tools, rather than technique, or focused on practical skill application. While this chapter cannot be a complete course in troubleshooting, it will provide a basic overview of troubleshooting, including the problem set and some

tools you will find helpful in locating and fixing problems (more) quickly. The basic question this chapter will answer is

> **What is the most effective process for finding and fixing problems in a network?**

Each of the following sections will address one part of the answer to this question.

> **Note**
>
> In many cases, the points made in this chapter will be exemplified through stories told in the first person; these are true stories of troubleshooting success and failure supplied to help you understand the point being made.

What Is the Purpose?

Troubleshooting tends to be an exercise in narrowing—starting from a broad and imprecise description of the problem, moving to a more focused description, and finally finding one or more things to change in the network to resolve the problem. As with design, however, it is often easy to narrow too quickly and then to hop around rather than remembering to refocus on the overall purpose of the system if your first attempt at solving the problem does not turn out to be "the" problem.

In the middle of a long, exhausting troubleshooting session, it is easy to think of the system as the network path the application runs over and the application itself. To use an example from electronics, rather than networking:

> *One of the pieces of equipment on the flightline was a wind speed indicator. This is fancy name for a really simple device; there was a small "bird" attached to the top of a pole with a tail guiding the bird into facing the wind, and then at the nose of the bird an impeller attached to a Direct Current (DC) motor. The DC motor drove a simple DC voltmeter graduated with wind speeds, and the entire system was calibrated using a resistive bridge in the wind speed indicator box, and another in the wind bird itself. The power from the impeller was passed to the voltmeter, several miles away, through a 12-gauge cable. These cables were particularly troublesome, as they were buried, and had to be spliced using gel-coated connectors, with splices buried in gel-filled casing. This was all before the advent of nitrogen-filled conduit to keep water out.*

In one particular instance, a splice failed, requiring the cable to be dug up by hand, and the splice opened and repaired. A special team was called in to resplice the cable, but even with the new splice in place, the wind system could not be calibrated to work correctly. The cable team argued the cable had all the right voltage and resistance readings; we argued back that the equipment had been working correctly before the splice failed, and all tested on the bench okay, so the problem must still be in the splice. The argument lasted for days. From the view of the cable team, their "system" was working properly. From the perspective of the weather techs, the system was not, even though the testable components were. Who was right? It all came down to this: What does the "system" consist of, and what does "working properly" mean?

Eventually, by the way, the cable splice was fingered as the problem in a capacitive crosstalk test. The splice was redone, and the problem disappeared.

The purpose is ultimately what the system is supposed to *do,* not just what *you can measure.* It does not matter if the network, or some component of the network, appears to be working fine. What matters is whether or not the system is accomplishing its purpose.

Of course, this means you need to understand what the purpose of the system is. In the broadest view, this means *what the system is supposed to accomplish from a business perspective.* A network can be running just fine from the perspective of the engineers who built it, but if it is not solving a business problem the way the business problem needs to be solved, it is still broken.

On the other hand, it is important to remember business folks do not always understand precisely how the business and the network relate, or they may have unrealistic expectations of what the network is capable of, or what is possible. In these cases, resist the urge to ask, "How high?" when the business says, "Jump!" Rather, cultivate a conversation in which you, the network engineer, have the right to say, "No, this will add too much complexity," or "The tradeoff here is too high."

Moving from the business to the network itself, there is a different, but still large, context: the network components.

What Are the Components?

Saying "a network is made up of components" is like saying "a menagerie of handmade glass animals is made up of…glass"—it is not very useful. More specifically, what are the components of a network? In the network world, there are

- Hardware devices that process and forward traffic, such as routers, switches, and stateful packet filters
- The environmentals, such as the power and cooling
- The cabling, interfaces, and other hardware
- The software running on these devices (the operating system)
- The software applications providing the information needed to forward packets; the control plane
- The specifications to which the network was designed and needs to operate in order to fulfill *business requirements*
- The requirements placed on the network by the applications the network is supporting

A broader, and simpler, set of terms might be: requirements + network software + protocols + equipment. Again, this might be a little obvious, but it is easy to forget the entire picture at 2 a.m. when the fires are burning hot, and you are trying to put them out.

How well can you know each of these four systems? Can you know them in fine detail, down to the last packet transmitted and the last bit in each packet? Can you know the flow of every packet through the network, and every piece of information any particular application pushes into a packet, or the complete set of ever-changing business requirements?

Obviously, the answer to these questions is no.

As these four systems within a network interact (remember interaction surfaces from the first chapter?), they create a larger system that suffers from a combinatorial explosion. Figure 24-1 illustrates.

There are far too many combinations, and far too many possible states, for any one person to know all of them. How can you reduce the amount of information so you can reasonably understand the state of an entire system, and hence be able to troubleshoot it? By building abstract models of the system's components, the interaction points between those components, and, ultimately, of the system itself.

This is the first skill of effective troubleshooting: build a set of models describing the system.

All models will necessarily be *incomplete;* a model can represent only some aspects of an entire system or subsystem. Thus, models are a two-edged sword: they present a more readily understandable version of a system, but they also present a necessarily incomplete version of a system.

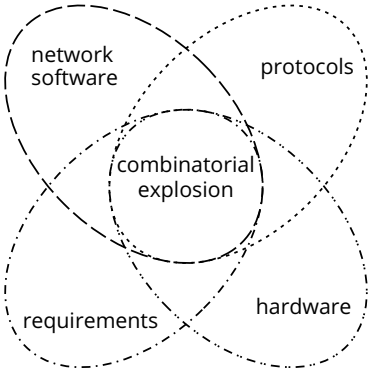


Figure 24-1 *Combinatorial Explosion of Systems in a Network*

Models and Troubleshooting

There is no single way, nor a single set of tools, you can use to build an effective model. This section will consider some common kinds of models—*how* models and *what* models—and discuss how to build them. The importance of accurate models and the ability to use models in the troubleshooting process effectively are also considered here.

> **Note**
>
> This has some interesting implications, of course. For instance, when a system is a "black box," which means you are not supposed to know how the system works, your ability to troubleshoot the system itself is nonexistent, and your ability to troubleshoot any larger system of which the black box is a component is severely hampered.

Build *How* Models

How models are built using problem/solution sets. This entire book, in fact, is an exercise in building *how* models, using a three-step process:

1. Determine the problem that needs to be solved.
2. Investigate the range of possible solutions for the particular problem.
3. Understand how this particular implementation uses a particular solution to solve a particular problem.

A potential fourth step beyond these three is integrating many solutions together into a complete system, considering the interaction between the solutions (such as where one solution reduces or increases the effectiveness of some other solution, etc.).

How models, of course, do not answer only one sort of question; for instance, when considering *how* Border Gateway Protocol (BGP) peers are formed:

1. How does BGP manage flow control, error control/correction, and data marshaling between two BGP speakers?
2. How does BGP manage the peering state between two BGP speakers?
3. How does an operator configure BGP to properly form peering relationships between two BGP speakers?

Each of these is a separate kind of *how* question. Where many engineers go wrong in building a solid foundation for troubleshooting is knowing the answers to the second and third questions, while never spending time on the first. Engineering tends to end up being focused on the question *how do I get this done?* rather than *how does this work?,* specifically *why does this work this way?* The result is a "hunt and peck" sort of troubleshooting style—combining small snippets of past problems with lots of knowledge about how things should be configured to make them work. This is generally a very inefficient way to not only design networks and protocols, but also to troubleshoot them.

You need to be able to build *how* models of all three types. There are several useful ways to build up your stock of *how* models, including

- Reading protocol theory and specifications, so you understand how and why a protocol operates (what problems are being solved and how they are being solved)
- Examining the designs of protocols and networks, and how they have performed in the real world
- Learning basic algorithms and heuristics, along with the problems they are designed to solve

Essentially, building *how* models is more about *theory* than *practice;* this is why engineers often skip learning *how* models—which prevents them from growing their engineering skills over the long term. *How* models are sometimes best expressed in a graphical format, such as Universal Modeling Language (UML) sketches, or flow charts.

Build *What* Models

What models are different from *how* models in describing the state of a particular network or application, or a common pattern found across many networks or applications. These kinds of models generally answer questions such as

- What is the normal path of this traffic flow (the signal path) through the system (such as a network, application, etc.)?
- What is the normal set of top talkers on this network?
- What is the normal distribution of load between these two paths in the network?
- What is the normal startup process between two BGP speakers?

The only real way to learn *what* is to observe and summarize many times over. For instance, observing the top talkers across a large number of networks, or even in a single network across a number of years, will give you a good sense of where to look for top talkers, and a good sense of when the top talkers in *this situation* do not make sense.

Manipulation in the observation process is important here, as well; Figure 24-2 illustrates.

In Figure 24-2, some value (representing, perhaps, an event or a property of an object) is assigned a variable, X. The question causation answers is: *does X somehow cause Y?* To answer this question, a range of possible interventions, or rather actions that will potentially modify X, are considered. In order to show X causes Y, all other potential interventions, Z_1 through Z_n, are held constant, while one potential intervention, Z_i, is manipulated. Manipulability is useful in building *how* models

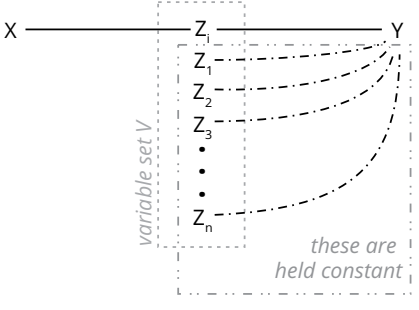


Figure 24-2 *Manipulation and Causation*

by helping you understand the relationship between the different parts of a system; if you manipulate the impact of X on Y by changing Z_i, then you can better understand the relationship between X and Y.

For instance, assume you want to understand how a specific application uses the network in terms of traffic flow, one possible way to go about discovering this information is by setting up a test instance of the application that passes through a router on which you can manipulate the Quality of Service (QoS) settings. By manipulating the QoS settings while watching the traffic flow, you can get a better sense of how the application works; you are literally building a *what* model of the application's operation.

It is important to build such models under as close to real-world conditions as possible, rather than in a "lab environment." A real-world example might be helpful here, told in the first person:

> *One time I was called out with another tech from my shop to work on the FPS-77 storm detection radar. There was some problem in the transmitter circuit; the transmitter just was not producing power. A resistor blew in the "right area" all the time, so we checked the resistor, and sure enough, it seemed like it was shorted. We ordered another resistor, shut things down, and went home for the morning (by the time we finished working on this, it was around 3 a.m.). The next day, the part came in and was installed by someone else. The resistor promptly showed a short again, and the radar system failed to come back up.*
>
> *What went wrong? I checked what was simple to check, what was a common problem, and walked away thinking I had found the problem, that is what. It took another day's worth of troubleshooting to pin the problem down, a component in parallel with the original resistor, but not on the same board, or even in the same area of the schematics, had failed. This second component was an inductor—essentially just a piece of wire wound tightly around a ceramic core. Inductors only show resistance when alternating current passes through them; they will always show a short when direct current is passed through them. Because the resistor and the inductor were in parallel, and the ohmmeter (a device which measures resistance) only uses direct current to measure resistance, the entire circuit appeared to be shorted.*
>
> *In reality, the inductor failed, but the ohmmeter, because it cannot produce alternating current at the right frequency and power level, simply could not "see" the correct failure, and I was too tired, and too convinced I had found the problem in the first try (because it was the "common" problem in this signal path), to check beyond the first discovery.*

There are several kinds of *what* models engineers should build, including

- A description of the normal state of each system in a production environment. This is often called the baseline, and should include traffic levels measured at different points in the network at different times of the day, different seasons, and during different kinds of regular events; the amount of time it takes for any particular process to run, such as the Shortest Path First (SPF) runtime in a network running a link state control plane; jitter and delay through the network on a per application basis, etc. These measures are important because you cannot know what "broken" looks like unless you know what "normal" looks like.
- A description of the normal configuration of each system in a production environment. Many networks will have a single source of truth that contains the proposed configuration for each device. Many automation systems are designed to ensure each device matches the proposed configuration contained in this single source of truth. These systems should also contain the *intent* behind each configuration, as a single intent can be expressed in many different ways.
- A description of the "normal" reaction of the network to different types of events.
- A description of the signal path of every application running on the network, including the origination of information from the application, the paths the traffic normally takes through the network, queuing, and other ways in which the traffic is processed.
- A description of the security boundaries in the network, including the boundaries of each security domain (logical or topological), why the security domain exists, and how the various security domains interact.

While some of these will necessarily be *complete,* in containing every available piece of information within a particular domain, it is also important to be able to summarize each one into a model. Knowing what to abstract is a skill that takes years to develop, and can never said to be perfected.

Build Accurate Models

In fact, choosing which information to abstract into a model is such a difficult problem that network engineers often live with poorly built models that simply do not represent the underlying system accurately—or simply do not provide the

information needed to ground good troubleshooting or design. Chapter 3, "Modeling Network Transport," discusses one such (controversial) example, the widely used Open Systems Interconnect (OSI) model. The OSI model is a good example of a useful model in a narrow range of contexts, but is often used far outside the domain in which it adds value. Figure 24-3 illustrates the OSI and the Recursive Internet Architecture (RINA) models for reference.

A simple question illustrates the differences between these two models of forwarding information through a network: *what functionality does this model describe?* The Open Systems Interconnect (OSI) model generally describes the kinds of information contained at each layer in the network, which is also useful in describing the kind of information carried between layers to carry information through the network. The RINA model, on the other hand, focuses on *functionality:* what problem is solved where in the network stack for this particular connection (whether hop by hop or end to end).

While the OSI model is often useful for *coding* a network stack, because it describes information and APIs, the RINA model is often more useful for *understanding* a network stack—knowing what is happening where and why. The RINA model, in other words, more closely aligns with the problem/solution mindset needed to troubleshoot a network problem.

Accuracy does not mean *perfection*, in the sense that every aspect of the system is represented, but rather *fit to purpose*. Different models may be required to understand different aspects of a particular system; one model may be useful for troubleshooting one sort of problem, and another model may be useful for troubleshooting another sort of problem (or another part of the same overall problem).

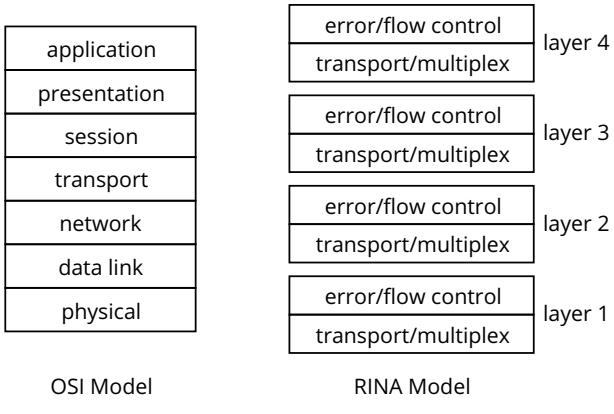


Figure 24-3 *The OSI and RINA Models*

Shifting between Models

Having a lot of models in your head to describe various aspects of the system is not helpful, however, if you do not know how to apply these models to the problem at hand. If you just take all these models and add them to the store of knowledge you already have about the system, you can make troubleshooting harder, rather than easier. The key lies in knowing how to apply models to the troubleshooting process.

The first step in applying models to troubleshooting is to learn how to shift between the various models as you need to, as you move between needing to understand a broader view of the system and a more detailed view of any piece of the system. Figure 24-4 illustrates.

In Figure 24-4, the overall system is depicted as a network path. Of course, there would be a larger context, such as business requirements, an application, or a set of applications, in real-life situations, but using the network path as a larger context will work for this example. Assume you are troubleshooting a problem with a specific flow through the network; the overall model you would keep in your head is the entire network path. As you encounter individual pieces of information about the problem, however, you might realize the problem appears to be something in the data plane, which might include QoS. Further information might indicate the problem is the application's reaction to jitter on the network, which should move you from the QoS model into a jitter model, which evokes a different model.

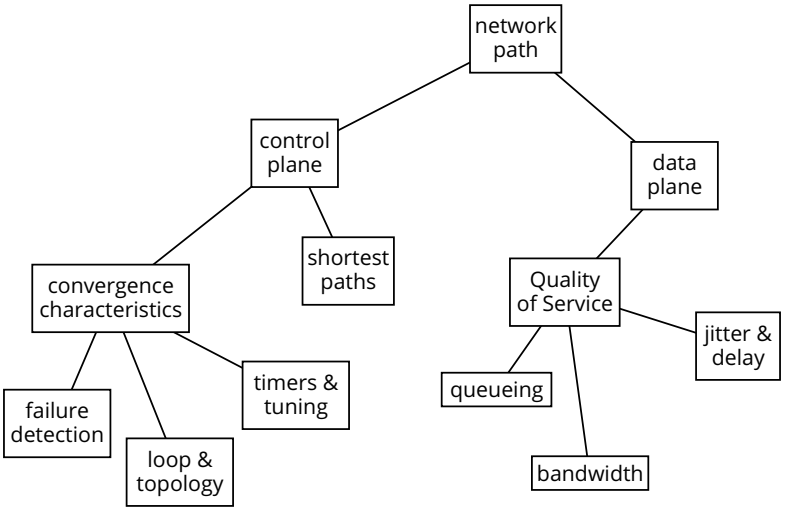


Figure 24-4 *Using Models in Troubleshooting*

Each model, as you move down the tree, is going to contain more detail within the specific area, but it is going to exclude more information from other areas of the network. Each model should accurately represent the problem and system at hand; using a model that "does not fit" at any point in this process can lead you down the wrong path in the "model tree."

Another problem engineers often face in troubleshooting is an unwillingness to mentally move back toward the top of the tree; in this case, it would be easy to focus on the jitter problem without considering how the convergence characteristics of the control plane might interact with jitter. Once an idea has been formed about what the problem is, it is important to start back at the top of the "model tree" and work back down toward the more specific models from the more abstract ones *taking the new information into account*. An example might be helpful here, told in the first person:

Two engineers were called into a major network failure at a rather important bank; for some reason, the routing protocol, the Enhanced Interior Gateway Routing Protocol (EIGRP), simply would not converge under some conditions. The Technical Assistance Center (TAC) had tried various troubleshooting techniques, reconfiguring EIGRP in various ways; finally the problem was escalated to the Global Escalation Team.

To begin, the escalation engineers started gathering all the information they could on the state of EIGRP during the outage; thus the primary model in use was around the operation of the EIGRP protocol and its convergence process. Over time, it became apparent the problem related to missed EIGRP packets; so the engineers began looking at the packet processing and the transport links between the routers. Thus the model in use for troubleshooting moved towards the transport and packet processing side. It appeared the packets were being transmitted, but not received, so the focus again shifted to packet processing on the impacted routers. As almost every router in the network appeared to be impacted, this was still a very wide scope, but it quickly dove into rather detailed considerations about how packets were received, queued, and forwarded on to the EIGRP process, and how information about what was going on in this processing could be gathered in a production network.

To discover this information, a server was set up to capture the input queue of several routers periodically. Each time there was a failure in an EIGRP neighbor state, the log file was pulled to see what, specifically, was in the queue at the moment in time to cause the EIGRP packets to not be delivered. The results were puzzling, to say the least; the packets in the queue were always from the same Internet Protocol (IP) address. No one could identify this IP address, so the

engineers began to suspect some form of denial of service attack. Ultimately, however, the offending server was found: it was a security server. The routers in the network had been configured to send a per command authorization request to this server to ensure the user currently logged in to the router had permission to run the specific command. The packets in the input queue each time the command completed and printed its output were the reply from the authentication server allowing the command to be executed. Needless to say, this rabbit trail did not help solve the problem any faster.

Eventually, the command level authentication was turned off, and the problem was found. A new backup software package had been installed on every host in the network—the server version had been installed on every host, rather than the client version. The server version, in order to find clients, attempted to contact every host on the network using subnet broadcasts across the entire IP address range, at a very high packet rate. These subnet broadcasts were being consumed by the actual routers, clogging the local process input queue, and hence causing EIGRP packets to be dropped.

The problem here ultimately required shifting between a number of different mental models, each one covering a different part of the network's operation, but it was important, when shifting between models, to refocus on the "larger context," so the problem at hand would not be forgotten. For instance, progress on the actual problem was completely left aside while the "rogue IP address" was being chased down.

Half Split and Move

While being able to shift between models is important, shifting between models randomly is generally not the most efficient way to go about troubleshooting a problem (although it is a common enough technique in the real world). What you need, once you have models built up and you have developed the ability to shift between models, is some way to guide how you move up and down the "model tree" while troubleshooting. Essentially, you need to know three things:

1. What question do you need to ask?
2. How do you ask this question?
3. Once this question has been answered, where should you move next in the system to continue troubleshooting?

One method stands out as a guide across many years of experience across many fields of study: the half split method. The steps for the half split method are as follows:

1. Map out the path of a signal through the system.
2. Split the path (roughly) in half.
3. Test the signal at the halfway point to determine if it is correct or incorrect.
4. If the signal is incorrect, move toward the source.
5. If the signal is correct, move toward the destination.

The connections between the half split method and the concept of having models should be fairly apparent:

- The path of a signal through a system can be described as a set of overlapping models, with "lower level," more detailed models being components of the larger context, more abstract models.
- The path of the signal is going to rely on the overall operation of standard components; each of these components either intersects or underlies one of the system models you encounter when tracing out the path of the signal.

The half split method can be used to guide your troubleshooting process through the subsystems within the system, using models to abstract enough information so you can "contain" the pieces you are trying to test in one step through the process. The half split method also helps you form the questions you need to ask of the system to know whether it is operating properly, such as

- What is the actual state of the system, and the result on the signal, at this point?
- What should a "normal" state look like?

The half split method can also force you to take your time and think through each step of the process. It is easy, when troubleshooting, to simply jump to where you think the problem might be and then dive into that small piece of the problem. Using the half split method will force you to look at what you are seeing, compare it to what should be there, and return to the larger context (move back up the "model tree") on a regular basis. These are all crucial to not getting lost in the weeds, troubleshooting something that is either not a problem or is a symptom instead of a problem.

When considering the concept of half splitting in a network, what is the *signal?* Essentially, it is *anything you can look at to verify the state of the system*. For instance:

- The state of a neighbor adjacency in a routing protocol
- The jitter or delay on a given set of packets within a flow
- The existence of a flow of packets at a particular point in the network
- The completeness of a flow of packets at a particular point in the network (primarily looking for dropped packets)

To find the signal, figure out what you would expect to be true of any particular information flow at this point (whether local, between peers, or end to end, host to host, etc.), and then determine what you think it should look like at the point you intend to test in the network. You might only need to shift between models during troubleshooting, but you might also need to shift between signals. For instance, in the example given in Figure 24-4, you need to shift from examining the jitter in a flow of packets through a network to the way in which the control plane converges, which may involve the *signal path* of distributing information about changes in the network topology. Both *getting stuck in a single signal path* and *hopping from signal to signal* are errors to be avoided in troubleshooting. These can be avoided using the two methods outlined in the following sections—using manipulability and simplification—but sometimes experience is the only and best guide.

Using Manipulability

Manipulation—and manipulability—is a key tool for testing theories and discovering the difference between correlation and causation. For reference, Figure 24-2 is repeated as Figure 24-5 here.

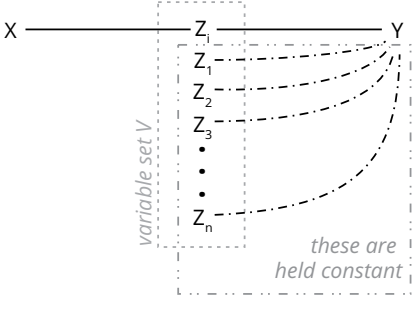


Figure 24-5 *Manipulation and Testing*

In troubleshooting, the key point is to find some Z you can use to modify the output of X in a way that impacts Y. In other words, if Y is the measured signal, you want to find some way to modify X to either show X is the cause of the current state of Y, or it is not. Figure 24-6 is used to explain this concept through an example.

Returning to the jitter example, assume there is some application passing traffic between A and H showing poor performance. After some work with the application, you determine the problem is with the jitter along the path between the host and the server. Examining the logs for the various devices, you notice the problem appears to correlate with SPF running on E. Using the half split method, you first trace the signal path, or, in this case the path of the flow, and find packets between these two devices follow the path [A,B,D,F,H]. You divide the circuit in half and decide to examine the signal at D.

How can you measure the jitter at D? The most obvious solution is to capture the packet flow on a packet trace device (or software loaded onto a standard host), then cause (or wait for) whatever problem to occur that is causing the event at E, to determine if there is jitter at the output of D when the event occurs. Assume you find the jitter is, in fact, present at the output of D; the next step is not to simply assume this is where the problem is. Rather, the next step is to move toward the source and determine if the jitter is also present at the *input* to D. In this case, it would be logical to examine the output at B during the event at E, to see if the jitter is also there. Skipping this half split step might seem like it would speed up your troubleshooting process—you know where the problem is, why not just move directly to finding out *why* this is happening? The reason is simple: by skipping the step moving toward the source and looking for the symptoms, you are failing to isolate the problem to a single point in the network. It is all too easy to spend a lot of time trying to understand *why* the problem is happening at D, only to discover the problem is not at D at all; it is someplace earlier in the network.

Assuming the signal is correct at the output of B, the next step is to find some way to manipulate the conditions at D to cause the problem; this will verify the problem

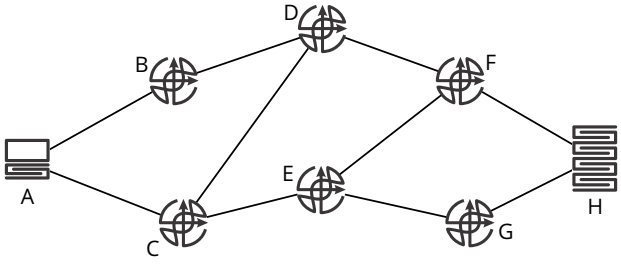


Figure 24-6 *An Example of Manipulation in Testing*

showing up at D and the event occurring (SPF at E) is not just correlation but is causally related in some way. The best way to do this is to examine any logs at E that can tell you *why* SPF is running at E and then *replicate those conditions while measuring the signal at D.*

Once the problem can be replicated, you can know, for certain, what the cause is and start thinking about how to solve the problem.

Note

Real life is messier than the example given here. In real life, there can be multiple interacting causes and no way to manipulate the network into causing the problem. Sometimes, then, correlation must be taken at face value, and you must *guess,* trying solutions until you find one that makes the problem *go away,* or just relying on your knowledge of the internal workings of the system to find a resolution without taking on the full process. The half split process, as described here, is an *ideal case;* you will likely need to modify it on a per case basis in the field. On the other hand, the closer you can come to the ideal, especially when starting out on simpler problems, the faster you will be able to develop the "troubleshooting sense" required to speed up the process. Further, when you are stumped, it is always best to stop relying on your "troubleshooting sense," and go back to the basics of half splitting, finding the signal, testing the signal, and finding a way to manipulate the signal.

Simplify before Testing

Returning to Figure 24-5 (and Figure 24-2), there are many different Z's (and probably many different X's). This raises an interesting question: how do you know which particular variable among the many available variables to concentrate on? Knowledge of the system, combined with a liberal dose of experience, will be your primary guides here, but there is one other thing you can do to make your life simpler: simplify the system.

For instance, in a network with a *lot* of parallel paths, you might make more headway in troubleshooting a problem if you begin by eliminating components until the problem goes away, or until the network is down to a bare minimum functioning set of links and devices. This might seem counterintuitive—why would you remove redundancy to troubleshoot, when the network is already having problems?—but it is sometimes the only way to narrow down where a problem is.

If the problem does, in fact, "go away" before you reach some minimal set, then you should suspect there is some form of positive feedback loop in the network causing the failure, there is some problem with the amount and/or speed of state being

carried in the control plane, or you have removed the problem in the process of reducing complexity (for instance, a flapping link, or device failing to forward traffic). In this case, you can add complexity back into the network until the problem reappears, which gives you a good manipulation test scenario. If the problem does *not* go away during the simplification process, then you now have a simpler signal path to troubleshoot, which will help you focus the half split process into a more confined space.

Fixing the Problem

Once the problem has been identified, you should fix it. However, the concept of *fixing it* isn't always so simple in the real world. There are normally two stages to *fixing it:*

1. Solving the immediate problem with a configuration change, hardware replacement, etc.—a *temporary fix*
2. Preventing the problem in the future through design or through replacement of equipment, etc.—a *permanent fix*

It is often very difficult to tell the difference between a temporary fix and a permanent one; a good rule of thumb might be

A temporary fix incurs technical debt; a permanent fix either reduces technical debt or leaves it constant.

Technical debt is very hard to explain, but essentially it means doing something that will either cause fixing a problem in the future to be more complex or will result in a similar failure mode happening in the future. Perhaps an example will be the most useful way to explain these concepts.

Assume you are in a situation where a number of different virtual networks suddenly stop carrying business-critical traffic at the same time. Investigating the problem, you find a broadcast storm is causing the problem; a particular network interface card (NIC) is pushing random broadcast packets onto the physical network in a way that prevents traffic from being carried across any of the virtual topologies (an example of fate sharing).

Unplugging the system with the faulty NIC is the obvious first solution: is this a temporary fix or a permanent one? The host is there for a reason, so this must be a temporary fix, correct? Yes…and no. Shutting down this host provides an immediate fix, but to determine if this should be the temporary or permanent fix, you need to

determine what the host is used for, and whether or not it is still needed. Leaving a host attached to a network if it is no longer needed, even once it is repaired, simply increases technical debt. Some other problem with this host in the future is going to cause a problem again, a problem that could have been avoided by simply removing the host entirely from the network.

Assume the host is required, replacing the NIC becomes the permanent fix, correct? Again, not necessarily. The host may be older and should simply be replaced entirely. Replacing the NIC in an older host may, again, simply increase technical debt, as the host may fail in some other way that causes a network failure at some point in the future.

Assume the host does need to be replaced. In that case, replacing the host should be the permanent fix, correct? Again, not necessarily. It could be time to reconsider the design of the network. If a single failed NIC should not be able to cause a system-wide failure, it may be worth considering a permanent fix that includes redesigning the network to reduce fate sharing or to reduce the scope of the failure domains.

The concepts of temporary and permanent fix are, then, flexible. Look to the business and the business drivers to think through where to stop when fixing a problem; don't assume replacing the hardware is the final fix, nor that every problem requires a complete network redesign.

Final Thoughts on Troubleshooting

The half split method, grounded in accurate models of the system, is an effective method to use when troubleshooting large-scale problems in any system. It is not perfect, of course; the real world is far too messy for a single process to be "perfect" at solving all problems, but long experience has shown the half split method to be the best general guide to finding problems quickly. To reiterate:

- Build accurate models, particularly the business, the applications, the protocols, and the equipment. This is probably where most failures to effectively troubleshoot problems occur, and the step that takes the longest to complete. In fact, it is probably a truism to say that no one ever completes this step, as there is always more to learn about every system, and more accurate ways to model any given system.
- Have a problem/solution mindset. This is probably the second most common failure point in the troubleshooting process.
- Half split, measure, and move.

Some final points to consider:

- Never assume the problem is a result of a configuration or design change; always remember equipment failures, new traffic patterns, and other situations can (and often do) cause failures. Some management systems focus on change control to the point of excluding other failure modes from view, which can discourage effective troubleshooting, and increase technical debt over time.
- Do not take shortcuts. Do not start with what can be easily tested. Do not assume you have found the problem on the first test. Always try to find a way to both prove, and attempt to disprove, your theory.
- If something does not look right, it probably is not.
- Many of the concepts used in troubleshooting can be applied to testing, as well—validation, etc.—before placing things into the network

Troubleshooting is an art grounded in technique, knowledge, and experience. Do not become frustrated if it proves difficult to learn this art; it often takes long hours of work with those who have more experience and have a better understanding of the system—and of what questions to ask—to have a strong set of troubleshooting skills. On the other hand, once you learn the art of troubleshooting, you will not likely forget it—and you will be able to apply it to many different areas of technology, not just network engineering.

Further Reading

Day, J. *Patterns in Network Architecture: A Return to Fundamentals*. Pearson Education, 2007. http://books.google.com/books?id=k9sFgIM-z6UC.

Fowler, Martin. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd edition. Boston, MA: Addison-Wesley Professional, 2003.

Huang, Peng, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. "Gray Failure: The Achilles' Heel of Cloud-Scale Systems." In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, 150–55. HotOS '17. New York, NY, USA: ACM, 2017. doi:10.1145/3102980.3103005.

Lieberman, Norman. *Troubleshooting Process Operations*. 4th edition. Tulsa, OK: PennWell Corp., 2009.

Mostia, William L. Jr. *Troubleshooting: A Technician's Guide*. 2nd edition. International Society of Automation, 2016.

Review Questions

1. Consider the Observe, Orient, Decide, Act (OODA) loop as described in the context of network security. How could the OODA loop be applied to troubleshooting?
2. Research the concept of a gray failure (look at the "Further Reading" section). How should gray failures change your troubleshooting process? What would you look for in troubleshooting gray failures?
3. Explain the difference between how and what models for network troubleshooting.
4. You are troubleshooting a problem where a small percentage of packets are dropped when being forwarded through a network. What does the percentage of packets dropped indicate about the tools required and the amount of information you will need to manage in order to troubleshoot this problem?
5. Describe different kinds of signals you might find in a network that can be used to trace out the operation of a particular system or application.

This page intentionally left blank

PART IV

Current Topics

Up to this point, this book has focused on problems and solutions. Part IV is a bit different, in that it primarily focuses on some of the newer trends in network engineering:

- What is the virtualization of functions, what does this accomplish in terms of business requirements and usage of networks, and how does virtualizing functions interact with network design and performance?
- What is the *Internet of Things,* and what impacts might this concept have on the design and future of networks?
- Cloud is moving from the new and exciting to the normal and operational; what is cloud computing, and how are clouds built?
- Networks are becoming so large that it is becoming difficult for administrators and engineers to actually manage each piece of equipment individually in near real time. How and where do automation and development operations play a role in solving these problems?

These chapters are simply overviews of each of these areas; there is not enough space in a book, even of this size, to cover each area in any sort of detail. It is important, then, to pay attention to the "Further Reading" sections at the end of each chapter to find more material to learn about each topic.

When reading these chapters, you should focus on understanding and analyzing these technologies and ideas in the framework presented throughout this book. Ultimately, there is nothing really new here in terms of problems solved or solutions

offered *at a technology level*. The constraints of the physical world, no matter how deeply buried in logical abstractions, will always impose reality checks on any solution set or design that must be deployed in the real world. Ultimately, you must look for the tradeoffs in design, security, privacy, cost, and fitness to the purpose of the network—complexity cannot be avoided; it can only be moved from one place to another in the network.

If you are reading this book after one of the trends in this section have "come and gone," you should still read these chapters for their ability to make you think about larger scale, hard-to-solve problems. The intent of this book is to be *timeless,* in that it will still be a useful learning guide and reference 20 years from now (when you are reading this, not when it is being written). While not every component of the future can be found in the past—there are always surprises in technology and ideas—the fundamental building blocks *can* always be found in the past.

The chapters in this part of the book will help you understand how the many pieces considered up to this point can be put together in different ways to make something new. The chapters in this section include:

- **Chapter 25: Disaggregation, Hyperconvergence, and the Changing Network**, which considers the application of disaggregation to building networks, and data center fabrics
- **Chapter 26: The Case for Network Automation**, which considers network automation and Development Operations
- **Chapter 27: Virtualized Network Functions**, which considers Network Function Virtualization, Service Chaining, and scale out service design
- **Chapter 28: Cloud Computing Concepts and Challenges**, which considers the business drivers, tradeoffs, and challenges in moving processing to public cloud services
- **Chapter 29: The Internet of Things**, which considers the widespread deployment of sensors and other "things" attached to the Internet, and the challenges and possible solutions resulting from this movement
- **Chapter 30: Looking Forward**, which considers the future of network engineering, including some further thoughts on network automation, blockchains, and named data networking

Chapter 25

Disaggregation, Hyperconvergence, and the Changing Network

Learning Objectives

After reading this chapter, you should understand:

- What disaggregation is and what advantages it brings to the business
- The concepts of converged, disaggregated, and hyperconverged architectures
- The basic design concepts of a network fabric
- The basic design concepts of a spine and leaf fabric
- The difference between a nonblocking and noncontending fabric
- The components necessary to build a disaggregated network

The network engineering world has, since the very beginning, been *appliance-based*; you buy a router, switch, or some other piece of networking gear, you rack it, cable it, power it on, and configure it to fulfill the functions you need. This is far different than the rest of Information Technology (IT), which has always had many more diverse models of software and hardware. This chapter will begin with a look at two specific movements within the broader IT world and then relate these movements to network engineering.

Changes in Compute Resources and Applications

In the distant past, computers were all built the same way. There was a case, a motherboard, memory, a hard drive, a keyboard, and a monitor. When companies started building networks, they began placing sets of servers into server rooms, including specially built furniture designed to hold 15–20 servers, and a Keyboard Video Mouse (KVM) switch so a single set of input and output devices could be used to manage all of the servers at once. The amount of space involved in such installations, along with the power and cooling problems, quickly led to the use of specially designed rack-mounted systems.

Each system, even in a rack-mounted case, was a single, standalone server of some type. One server might have file sharing, directory, and email services running (such as a Novell Netware, Banyan Vines, Lantastic, or IBM OS/2 server). Another server might have a database running on it, such as Oracle. As more resources were needed, the server would have additional memory installed, a bigger processor, more drive space, etc. This is called scaling up.

Over time, the processing and storage requirements simply became too large to build a single server able to handle the load, so applications were redesigned to run across multiple servers connected to the same segment. This is called scaling out.

Eventually, of course, through the work of Intel, VMware, and others, the applications, or processes, were disconnected from the physical compute resources—processor, storage, and memory—and placed into virtual machines (VMs), or later, containers. This virtualization process, however, had a side effect.

Converged, Disaggregated, Hyperconverged, and Composable

Once compute resources are virtualized, why should they be located on the same physical server? For instance, the hard drive does not need to be *in* the physical server, so long as it can be accessed as a virtual resource over the network connection. Thus, the compute resources themselves could be moved anyplace on the network, so long as they would be accessible, within specific performance requirements, to the applications that needed them.

The original physical format of these compute resources is called converged; all of the resources are converged in a single device. Only applications running on a physical processor can access resources such as disk, memory, and network interfaces, connected to the processor. Virtualizing access to these compute resources led to disaggregation. In a disaggregated system, the compute resources can live anywhere as long as they are accessible over the network. This brings the scale-out model to a new level. Rather than scaling out by crossing servers, you can scale out by actually pulling resources from various systems connected to the network as needed.

There is another side effect of the move to virtualize interfaces in this way. The virtual interface that applications use to connect to and use these resources essentially becomes a standardized Application Programming Interface (API), which means there is no reason to buy one brand of hardware over another, so long as the hardware meets the required performance metrics.

When you can buy hardware for its performance versus price profile, and use it with any other hardware (or software) you happen to have, the result is that the brand of equipment is deemphasized. This leads to the idea of a white box—buying hardware because of its components rather than the brand. Of course, *white box* is a somewhat loaded term; it somehow implies a couple of people sitting in a garage soldering boards together from whatever components they can come up with. This new "white box movement" might better be called disaggregation, as there is a wide range of hardware available, from basic features and functionality to fully supported branded devices.

The disaggregation movement, however, has a specific downside. Moving storage off the local system bus, connected directly to the processor, forces the processor and the applications running on the processor to access stored data through the network. The side effect is slower access to data. Although some databases are designed to allow the correct data set for specific operations to reside entirely in memory on a single node, carrying data to and from a disk over the network can still introduce serious limitations in a design.

The most obvious way to solve this problem is to move the data back onto the local system; however, then you lose the ability to build a set of compute resources dynamically. The solution to this problem is hyperconvergence. Here, the storage, memory, and network resources are still connected to individual processors, but they are virtualized in a way that allows all the processors attached to the network to access them. With good planning, storage, memory, and other resources can be allocated nearby, so network traffic is kept to a minimum, while still allowing VMs to be built out of a diverse set of resources.

Figure 25-1 illustrates the concepts of a converged, disaggregated, and hyperconverged architecture.

In Figure 25-1:

- In the *traditional* illustration, in the upper-left corner, each processor is attached to storage, memory, and network access through a local bus; applications running on the process have access to these resources.
- In the *converged* illustration, in the upper-right corner, a single process is attached to storage, memory, and network access through a bus. Multiple virtual machines are created using processor features; each of these virtual machines runs applications that can access the resources attached to the local processor bus.

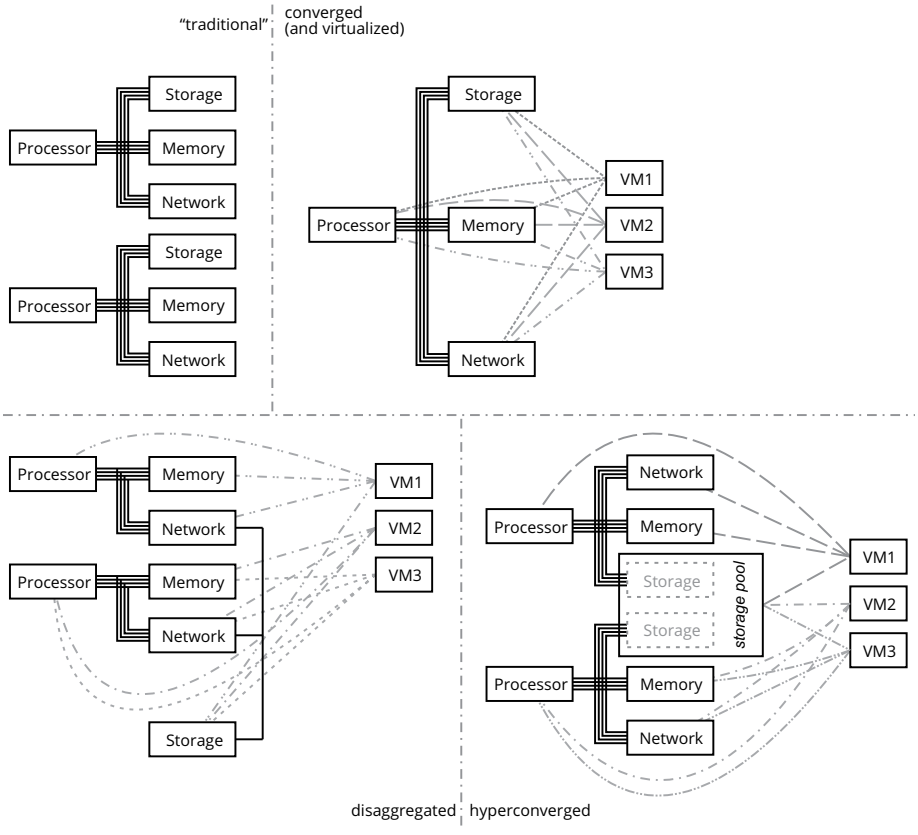


Figure 25-1 *Converged, disaggregated, and hyperconverged*

- In the *disaggregated* illustration, in the lower-left corner, the storage has been centralized onto a device reachable through the network. Virtual machines running on the various processors access local memory and network resources but connect to storage through the network, which is accessible through the local processor's bus.
- In the *hyperconverged* illustration, in the lower-right corner, each virtual machine runs on a particular processor, accessing memory and network resources connected to a processor through the local bus. An agent runs on each processor, as well, which redirects the locally attached storage to a network-based interface, and presents a single storage pool based on these resources. The storage manager will often attempt to locate data as close as possible to the processor using it.

> **Note**
>
> The term *processor* can be confusing in the context of virtualization. Many hosts contain one to four processors, with each processor containing one to eight cores. A single VM or container may run on a single core, consume all the processors and cores in a single host, or any combination of the above. To simplify the explanation here, however, the term *processor* was selected to represent any potential set of cores and/or processors a VM or container may run on.

You might note these illustrations focus specifically on the location of storage. This is because storage not only tends to be the easiest resource to move around the network, but it is also often one resource where you can save a lot of expense through some form of centralization. For instance:

- While data can be compressed on multiple devices, it is often better to run specialized hardware able to compress and decompress data to and from storage on the fly. Such specialized hardware can not only run compression much faster, but it can be tuned to compress more deeply and use less energy in the compress process than a general-purpose processor.
- The same holds true for encryption; most modern processors can certainly handle encrypting data while it is being written and deencrypting data while it is being read, but specialized processors are often so much more efficient, they are worth the investment if large amounts of storage are involved.
- Data deduplication can reduce the amount of storage used, also reducing costs. If, say, a company memo is sent with a 1MB attachment, and 1,000 people save it, the result will be 1GB of storage consumed. A data deduplication system can save one copy of the attachment, replacing each "copy" with a point to the single copy, saving 999 copies of the attachment. Data deduplication works for operating system files, applications, databases, and any other sort of information; it can dramatically decrease storage requirements in many cases.

In each of these solutions, applications are still limited to the physical memory and network resources attached to the local processor. In a composable system, even these resources can be shared among processors. Figure 25-2 illustrates one way to build a composable system.

In Figure 25-2, a processor bus has been extended so it has many different processors, network interfaces, memory banks, and storage devices that can be attached. A system manager *composes* sets of resources out of this large pool of resources for individual virtual machines to run on. Not all composable systems use an extended

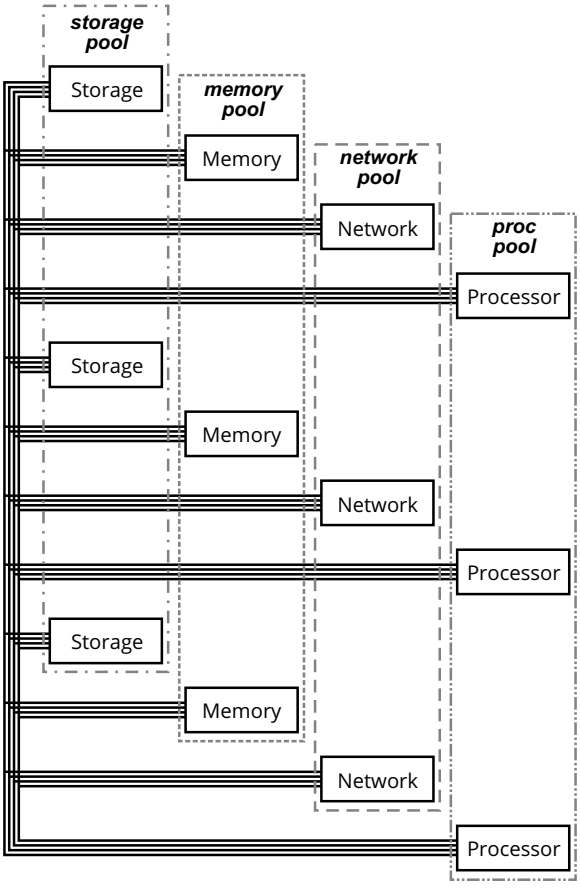


Figure 25-2 *A composable system*

processor bus in this way; some attach each individual device to an internal Ethernet network, using the network to transport information from the processor to the external network interface, or a storage device. This sort of configuration allows a system to scale out to very large sizes, while continuing to treat each resource as a white box; it does not matter who makes each device, so long as they can all present a uniform set of APIs to the composable system manager.

Applications Virtualized and Disaggregated

A second disaggregation movement happened as a result of virtualization at the server level—the disaggregation of applications. Most applications were designed

to run on a single device, with full access to an entire range of local hardware resources, and to complete a task from start to finish. For instance, an application tracking customer orders might hold customer information, product information, current orders, past orders, inventory, etc., all in a single set of databases. Over time, such applications were broken into a database back end and a business logic front end, but these two pieces were still somewhat unified, identifiable applications.

With the rise of virtualization, it started to make more sense to break up an application into many pieces, with each piece running on a set of virtual servers. In this way, any piece of the application could be scaled up to meet demand or scaled back to use less resources when demand was low—another version of scale out, but in terms of applications.

Breaking an application into smaller pieces, each of which represents a single service within the larger application, and then interconnecting those applications eventually leads to microservices, a form of computing where each individual module of an application is broken out into a smaller app, each of which does one thing very well. The apps are connected over the network so the application actually runs on the entire network.

Not only do such systems tend to scale out well, but they can also manage change and failure in more graceful ways. If a single host or router in the data center network fails, it will likely represent just some small part of the processing the entire application does; such failures can more easily be dealt with than a single host that runs an entire processing system failing.

The Impact on Network Design

These three trends—the disaggregation of server hardware, hyperconvergence, and the trend toward virtualized services rather than applications in the traditional sense—have had a marked impact on the design parameters for data center networks. This section will consider two of these changes specifically: the rise of east/west traffic and the rise of jitter and delay sensitivity in the network.

The Rise of East/West Traffic

In converged and virtualized converged systems, the network is primarily used for carrying traffic to and from hosts, whether the host is virtualized or not. The server is, in effect, a black box to the network; traffic of various sorts enters the device from the outside world, and traffic is transmitted from the device to the outside world.

Traffic being carried to and from servers from outside the data center is called north/south traffic, as it is traveling between the top and bottom of the network diagram as "traditionally drawn." Figure 25-3 illustrates.

In Figure 25-3, the entire server H appears to be one black box; moving traffic between storage, memory, processor, and the network interface is handled through the processor bus, which is, in effect, a small internal network. The primary traffic flows in this network will be from A to H and back again, which is along the north/south axis of the network.

Figure 25-4 illustrates what happens when the storage is centralized through disaggregation.

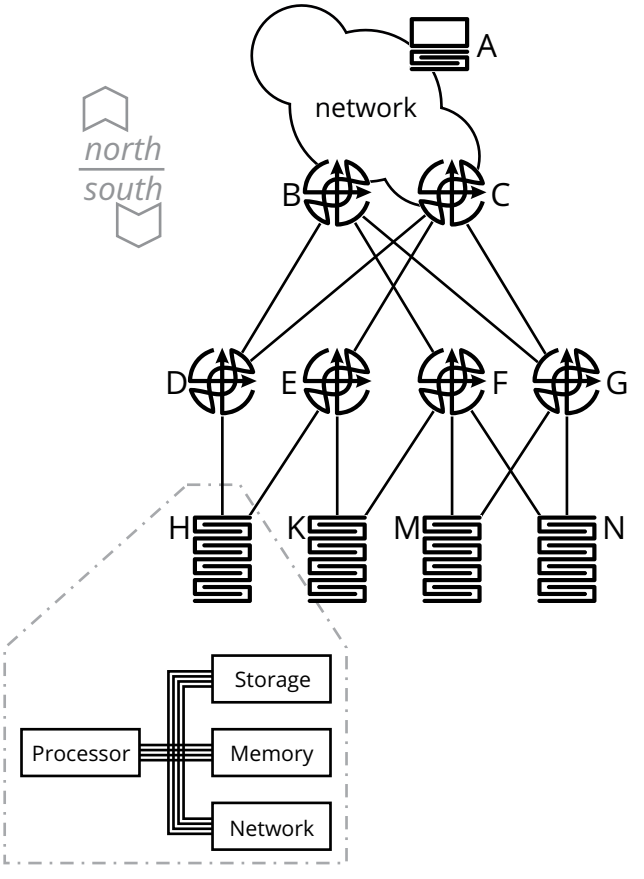


Figure 25-3 *North/south traffic flow in a network with converged compute resources*

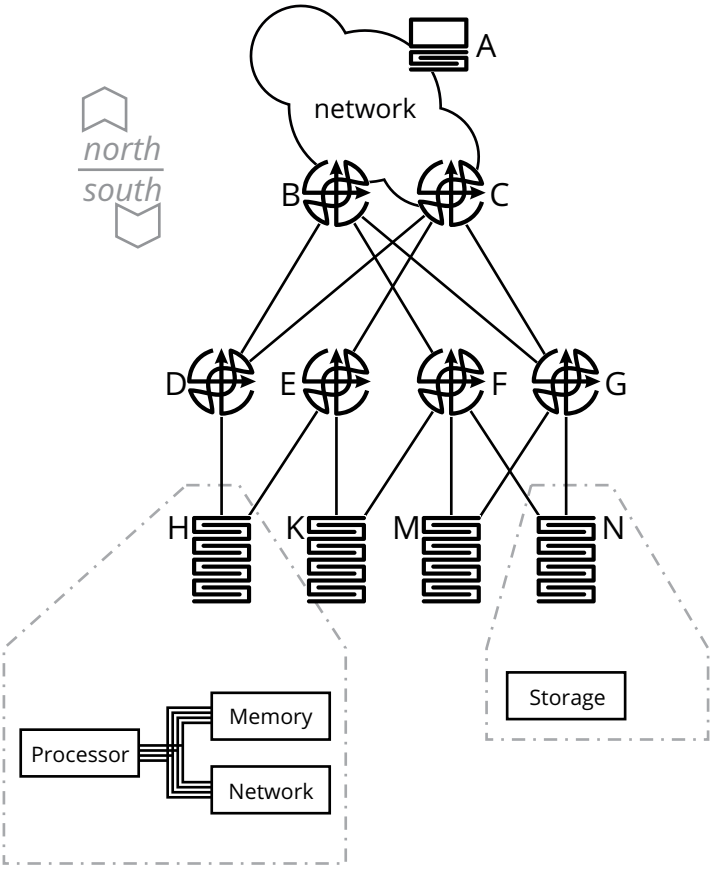


Figure 25-4 *The impact of disaggregation (centralized storage)*

In Figure 25-4, *any time* the processor needs to copy information from storage into memory, the data must travel across the network. This data is called east/west traffic, as it is flowing from one device connected to the data center network to another. The disaggregation of applications into services, and potentially microservices, has the same effect as the disaggregation of hardware resources. Combining these two realities, a single request from a host, such as A, will represent a small amount of north/south traffic but will drive a lot of east/west traffic.

How much more? Most web and hyperscale network operators report about a 10-to-1 ratio—for each bit of north/south traffic, there will be about 10 bits of east/west traffic. It is not unusual for web scale networks to carry multiple terabits of data a day in response to several hundred gigabits of actual user requests.

The Rise of Jitter and Delay

The disaggregation of applications and compute resources has caused jitter and delay through the network to become a very big problem. Specifically:

- Once you separate the storage from the rest of the compute resources, the performance of the application and the performance of the network are intrinsically linked. If the network is congested, for instance, taking even some fraction of a second to transfer data from a storage device to a processor, the impact on the performance of the application can be devastating.
- Once you break up the application into services and move toward microservices, the performance of any one service will impact the performance of the entire application.

A convenient way to think about this is: The processor bus itself has been extended over the data center network. The application, as a whole, is now running *on the network* in the same way it once ran *on a single host or within a single device*. The network, as a whole, is now a system and must be treated as a system.

Any delay or jitter in the network can cascade through the system, causing the entire application to perform poorly. When your revenue depends on user engagement, and user engagement depends on the speed at which your application loads, any problem in the data center network shows up directly as a loss of revenue.

Packet Switched Fabrics

How can network architectures be adapted to meet the requirements of an application running on the network itself, treating the network as a system? To solve this problem, network engineers returned to some old ideas about the best way to build circuit switched networks, merging them with packet switching principles to create the packet switched fabric. This section will consider some aspects of fabric design.

The Special Properties of a Fabric

How is a *fabric* a special case of a *network*? To begin, it is best to discard various *marketing* uses of the term *fabric*, such as

- Any network with an overlaid virtual topology, including a “core fabric” and a “campus fabric”

- Any high-performance network, with high performance meaning high bandwidth
- Any network with a lot of equal cost multipath (ECMP) availability
- Any network where the *entire network* is treated as a single "thing," rather than as a set of separate components

These uses of the concept of a fabric almost always come down to marketing; engineers and managers have become comfortable with a *fabric* being some sort of special network and hence more *desirable* than a "plain old-fashioned network." This is much like the marketing craze in the mid-1990s around calling a router a "layer 3 switch" because it performed a header rewrite in hardware. The last definition in the preceding list—any network treated as a single "thing"—is very clever, because it implies you cannot build a fabric out of individual components. Rather, in this definition, a fabric is something you must buy as a unit from a vendor as "one thing."

Leaving aside these sorts of marketing definitions, what makes a network a fabric? There are three specific characteristics of a *fabric*:

- The regularity of the topology
- The way in which the topology scales in bandwidth and connectivity
- The specific performance goals the topology is designed to fulfill in terms of forwarding

Each of these deserves a closer look.

Topological regularity means the topology of the network is *well defined* and *repeating*. To say a topology is *repeating* is to say the topology consists of a large number of identical pieces repeated to create the scale required; Figure 25-5 illustrates.

The difference between the regular and irregular topologies should be apparent:

- If you "pick up" [A1,A2,B1,B2] as a unit, and move A1 to the same position as B2, the two pieces of the topology are identical. In fact, [A1,A2,B1,B2]; [B1,B2,C1,C2]; [A2,A3,B2,B3]; and [B2,B3,C2,C3] are identical "subtopologies" of the larger topology. Each of these subtopologies is interchangeable within the larger topology.
- The same is true of [D1,D2,E1,E2] in the second network topology illustrated; these four routers can be moved to any other position in the network without any modifications to the overall topology.

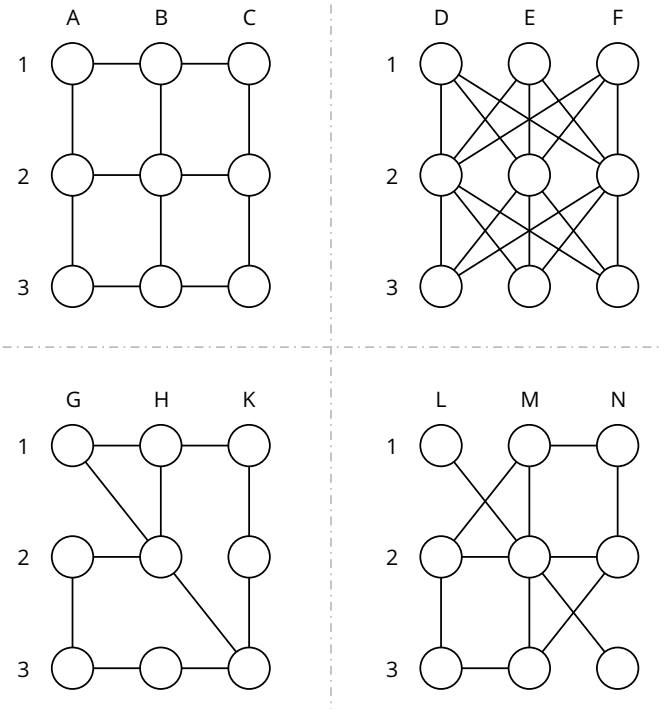


Figure 25-5 *Regular and irregular topologies*

- [G1,G2,H1,H2], however, is unique within the third network, at the lower-left corner of the illustration. There is no other place in the network with the same topology. This is an *irregular* topology.
- While [L1,L2,M1,M2] has the same topology as [M2,M3,N2,N3], neither of these sets of four routers has the same topology as [L2,L3,M2,M3]. Again, this is an *irregular* topology.

Why is this an important point when deciding if network is a fabric? First, because fabrics are generally designed to use completely replicable hardware, software, and configurations at the subtopology level. You can think of this as a form of *micro-modularization*, perhaps, with each piece of the network designed to be fully replicable in very small pieces primarily for ease of configuration and management, rather than for breaking up failure domains. In fabric designs at scale, the physical layout is separated from the logical layout of the network as much as possible.

The scaling characteristics of the network topology are the second marker of a fabric. Specifically, fabrics tend to scale out instead of scale up. These two concepts

have already been discussed in relation to servers and applications. How do they apply to a network? Figure 25-6 illustrates.

The upper network in the illustration is configured as a fabric, while the lower one is configured in a hierarchical topology. The problem at hand is, *how do you add enough bandwidth to connect a new pod of equipment*? In the lower network, the hierarchical design, you can add a new aggregation router at the edge of the network, and connect the new equipment there. However, adding this new router and new equipment may also mean the bandwidth in the core of the network needs to be increased to support the additional load. Generally, this means adding more links or perhaps adding parallel links and either running ECMP or bonding the links in some way. In either case, this means larger ports or more ports, higher-speed links, etc. The older equipment must either be replaced or augmented to add capabilities.

In the upper network, adding a single new pod requires adding three new routers to the network and the links associated with the new routers. However, *the total bandwidth*

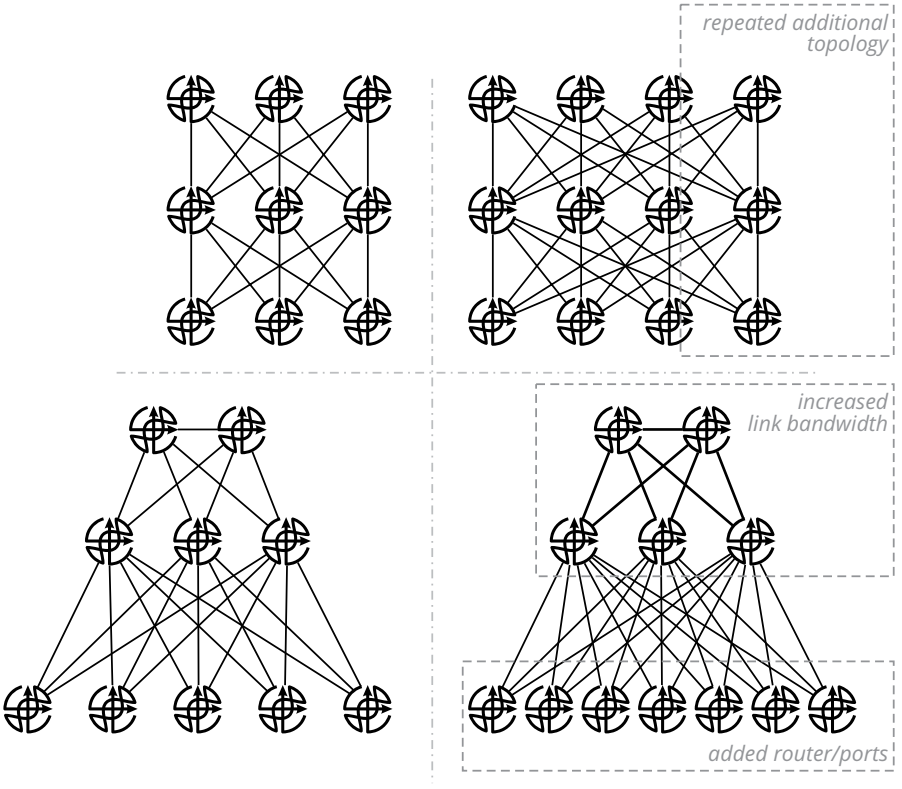


Figure 25-6 *Network scale up versus scale out*

of the network increases as the new connection point is added. Hence, the network scales by adding more equipment of the same kind, rather than by modifying the existing equipment. The difference between adding more modules and replacing or augmenting existing equipment is the key differential between scaling up and scaling out.

The general rule of thumb is this: fabrics scale out, rather than up; hierarchical designs scale up, rather than out. This is not always true, of course; fabrics do have a scaling limit based on the number of ports connected to each device, and other designs can be built so they have some measure of "scale out" before the hardware must be augmented or replaced, but the general rule holds in most cases.

Performance goals are the third differentiator between a network and a fabric. Networks typically have performance goals centering around Quality of Service (QoS) handling and uptime. Fabrics have similar, but sometimes slightly different, sorts of performance goals. For instance:

- Failure rates are often measured in terms of the pods and/or other components of the fabric, rather than the "entire network," or a particular application. Most applications designed to run on hyper- or web-scale fabrics are designed to tolerate being moved between racks of servers, so a single rack, pod, or link failing can be countered by moving the application to a different rack or pod attached to the fabric.
- The movement of workloads to different places in the fabric places an often difficult-to-manage mobility requirement on fabrics. Mobility is not often a factor in other network topologies. Workload moves on a fabric must be dealt with very quickly; application users do not often wait for the network to converge around a failed rack or pod.
- Fabric design is often focused on the fabric's oversubscription, which means the amount of bandwidth available in the network core compared to the amount of bandwidth available at the edge ports. For instance, if an edge switch or router (called a Top of Rack [ToR] or leaf) offers 320Gb of bandwidth down to servers, but has just 180Gb of fabric connections, it is described as being 2:1 oversubscribed. Another way to describe oversubscription is in terms of how much bandwidth is available from any port to any other port on the fabric. If every port on a ToR can send traffic at a full rate to some other set of ports attached to the fabric, then the fabric is said to have 1:1 (or no) oversubscription.
- Many network designs focus on reducing delay using traffic engineering and Quality of Service techniques. Fabrics, on the other hand, try to reduce jitter as well as delay, and mostly try to reduce end-to-end queueing, rather than implementing any sort of complex QoS. Many fabrics do, however, use some form of traffic engineering.

Spine and Leaf

One of the most commonly used topologies to build fabrics is the spine and leaf, which is not really a single design, but rather a family of designs based on the same basic building block. Figure 25-7 illustrates a basic spine and leaf design.

The bottom and top *stages* are called either Top of Rack (ToR) or leaf nodes; these are where hosts and other devices are connected to the network. The remaining stages are generally called spines of some sort; there are two rules for spines in a standard spine and leaf:

- There are no connections between spine routers.
- No devices of any sort are connected to spine routers; all connectivity into the fabric is carried through a leaf node.

An alternate form of numbering is shown on the right side of the fabric. Fabrics can be drawn *folded*, but the stage count is given based on the total distance through the fabric; the fabric shown in the illustration is a *five-stage* or *ary* fabric. The number of stages can be confusing in some configurations of spine and leaf topologies.

Note

Some spine and leaf designs *do* have connections between spine routers; this can solve some problems when you are aggregating routes on the fabric, but it also can add a lot of complexity into the network design and control plane convergence.

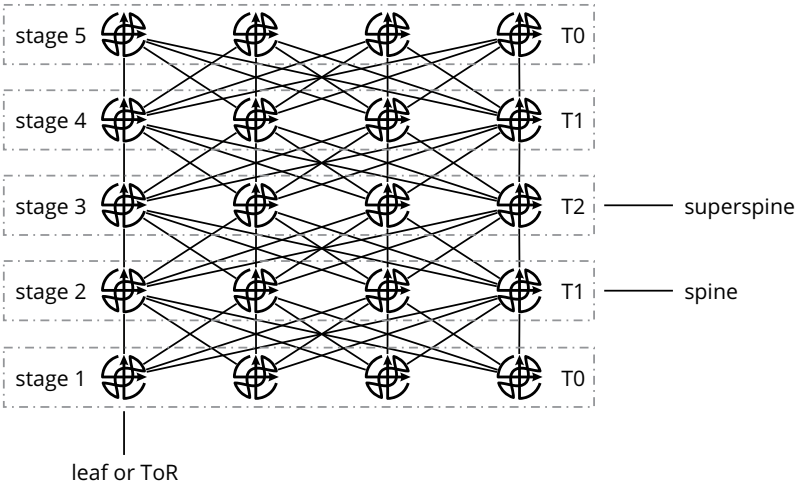


Figure 25-7 *A five-stage spine and leaf*

In the standard configuration, as shown in Figure 25-7, adding stages does not really add more ports; instead you would scale out this kind of fabric. The scaling limit is the number of ports available on each device in the fabric, and the oversubscription rate is the difference between the amount of bandwidth offered by the ToR routers and the amount of bandwidth available from the ToR routers into the fabric.

One of the key points about spine and leaf fabrics is *they do not need a complex control plane to forward traffic correctly*. While most hyperscale networks do use a complex control plane, it is normally used to compensate for cross links, to provide information for an overlay virtual network, or to provide for some form of traffic engineering.

Nonblocking versus Noncontending

What is the difference between *nonblocking* and *noncontending* networks? In a nonblocking network, there is no way for any packet to be blocked while traversing the network. Packet switched networks *cannot* be nonblocking. Consider, for instance, the network illustrated in Figure 25-8.

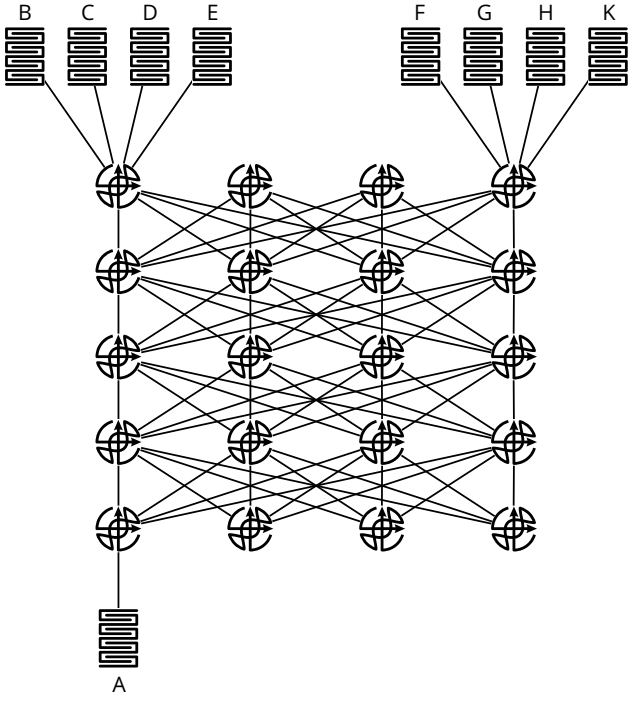


Figure 25-8 *Nonblocking versus noncontending*

If every link in this network is the same bandwidth, then it is possible for B, C, D, and E to each send a stream the same size as their connected links to F, G, H, and K. Because the number of edge ports offered to servers, 4, is the same as the number of fabric side ports on each ToR, it is possible for each device connected to the fabric to fill its connection link into the fabric without anything being dropped. On the other hand, if B, C, D, and E each send a large stream toward A, the link from A to its connected ToR will need to switch four times more traffic than it has in available bandwidth. At this point, the fabric itself does not block any traffic, just the edge port facing the attached server. However, if F sends a full rate stream with A as the destination, there are now five full rate streams that the ToR at A needs to receive, and just four links on which to receive them. In this situation, one of the spine switches is going to need to block traffic (or, in this situation, throw enough traffic away to reduce the amount of traffic on the fabric to a level the available links can support).

In a circuit switched network, this problem would be solved on the inbound side through traffic scheduling, so the fabric itself does not block any traffic. In a packet switched network, however, it is still possible for multiple attached devices to send an overwhelming amount of traffic toward a single destination, causing the fabric itself to block traffic. There are two ways to resolve this kind of problem.

First, QoS controls can be placed on the network to control which traffic is forwarded and which is either dropped or at least queued for some amount of time. Second, the application can be designed to sense this sort of problem and slow down the rate at which it is transmitting. If these first two mechanisms sound familiar, it is because they are the same kinds of techniques used in any network to deal with congestion.

Two other helpful concepts in the fabric world are the type of tree. Skinny tree fabrics have the same link between every stage within the fabric (this does not include the ports provisioned for servers, however). Fat tree fabrics use a smaller number of higher-speed links in the center stage of the fabric, generally between the super spine and the spines, and lower-speed links between the spines and the ToR devices. In either case, the same oversubscription concepts apply; the primary difference between the two is optical, just in the amount of cabling and how the ports are configured on the various devices in the fabric.

Traffic Engineering on a Spine and Leaf

Why would you ever need to deploy traffic engineering on a fabric designed with no oversubscription? Figure 25-9 is used to illustrate.

In Figure 25-9, assume A has some large flow destined to C, consuming just about all of A's local link into the fabric, and is persistent; it will last for more than two or three seconds, potentially into the realm of days or months. These large, persistent flows are called elephant flows in the context of a data center fabric. Generally, elephant flows relate to large data transfers (such as those involved in moving a Hadoop job around on the fabric, or a database replication), and are not sensitive to jitter. Assume this flow is placed on the path [V1, W2, X3, Y2, Z1]. At some point during the duration of this elephant flow, B starts a short session with low bandwidth use requirements, in support of an delay or jitter sensitive application. Assume this flow is placed on the path [V2, W2, X3, Y2, Z3].

Both of these flows are going to share the [W2, X3] and [X3, Y2] links. Given the nature of the two flows, the smaller flow, sometimes called a mouse flow, will not meet its jitter requirements even if there is plenty of bandwidth available on other

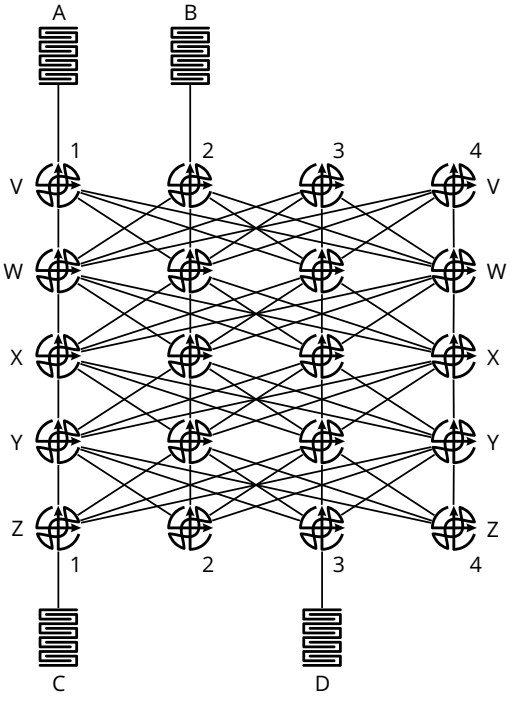


Figure 25-9 *Traffic engineering in a fabric*

paths on the fabric. To resolve this, the elephant flow needs to be pinned to a single path, and the path taken out of consideration for use by other flows passing through the network. This is the primary use case for traffic engineering in a noncontending data center fabric.

A Larger-Scale Spine and Leaf

Many large (web- or hyper-) scale networks use a butterfly fabric, which is a variant of a Benes, and also a type of spine and leaf fabric. Figure 25-10 shows a small example.

In Figure 25-10, there are two fabrics, each of which might also be called a core, and a set of ToR devices. Each fabric is a full spine and leaf; each ToR connects to one point in each fabric. Depending on your perspective, this can be considered a five-stage fabric, ToR to ToR, or it can be considered a three-stage fabric with an additional set of access devices (though you would still never connect any devices or external access to the leaf nodes of the two fabrics in this network—all connectivity would be through one of the ToR routers or switches).

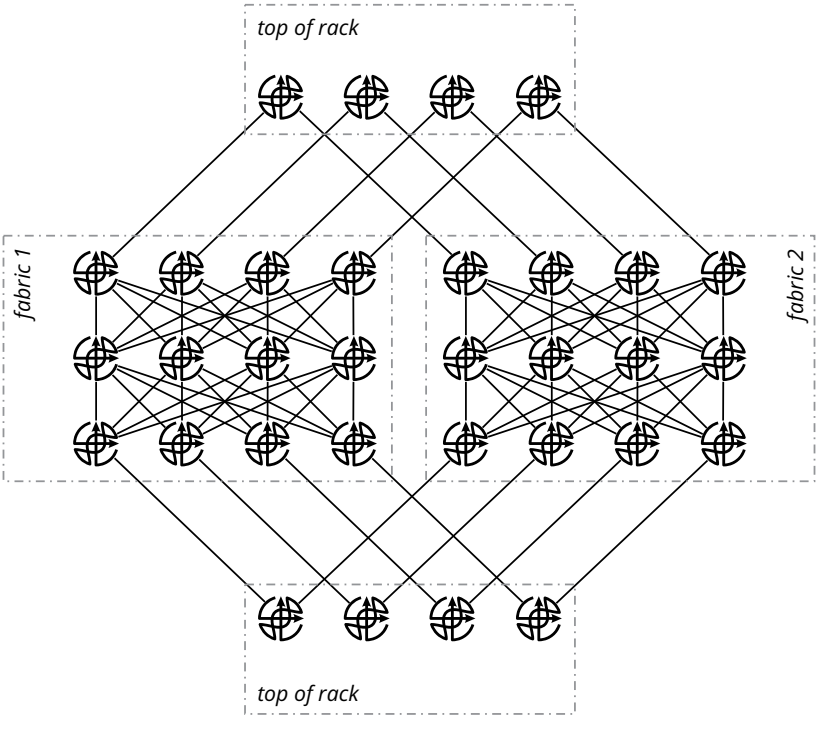


Figure 25-10 *A butterfly or Benes fabric*

The primary advantage of such a design is the oversubscription rate and scaling can be adjusted within the limits of the fabric side ports of the ToR devices. To decrease the oversubscription rate, increase the number of cores. To increase the scale, increase the number of cores and ToR devices in parallel.

Disaggregation in Networks

Disaggregation has caused a revolution in the way compute resources are built and used. Can these same concepts be applied to the network?

Networks have, "since forever," been built out of appliances. A device is purchased from a vendor, racked, cabled, powered on, and configured through some sort of semiproprietary interface. Each device has a fairly unique feature set; in fact, the feature set of the software and hardware combined is the primary selling point, because the wide range of features (and nerd knobs) allows a single piece of gear to be used in a wide variety of networks, under a wide variety of conditions. This ability makes the appliance "future proof," in the sense that no matter what problem you throw at the appliance, it is likely to have some feature that can be enabled to "solve" the problem (for some value of "solve").

The result is an engineering world that

- Chases features whether or not they are needed to solve a particular problem right now, leading to overengineering in many cases.
- Chases service and support, because the devices themselves are so complex, and the networks built from them tend to use a combination of features found nowhere else in the world; hence each network is the same and yet each network is completely unique.
- Splits the work of design and architecture between the vendor, who shapes architecture by building products for the widest possible audience, and the operator, who tries to use as many square pegs as possible, because this is what vendors offer, regardless of the shape of the problem.

Looking over the history of compute resources, *these are precisely the same problem set that disaggregation was designed to solve*. Perhaps, then, disaggregation in the network can help solve these same problems. There is one more lesson from the compute and applications to consider before looking at disaggregation in the network, however.

Disaggregation does not look the same in applications and compute resources; this is primarily due to the physical limitations of each kind of system, and where

there are points at which efficiency can be improved. Given this experience, disaggregation will probably look different in the network, while still driving the same sorts of efficiency and operational gains.

The key points in disaggregation in applications and compute resources have been

- Decoupling hardware from software
- Commoditizing the hardware so it is usable across a wider range of functionality
- Specializing the software (such as services- and microservices-based application development)
- Pooling resources as needed to solve specific problems using the principle of scale-out design

How can these be applied to the network? The first step is to consider where software and hardware can be decoupled, which drives the remaining steps. Returning to a sketch of how a router is built can be helpful here; Figure 25-11 illustrates.

Note

The kind of diagram shown in Figure 25-11 is notoriously difficult to draw, simply because there are so many different ways of building software. What is shown here is one possible representation just to illustrate the various pieces required to build a router (or other network device).

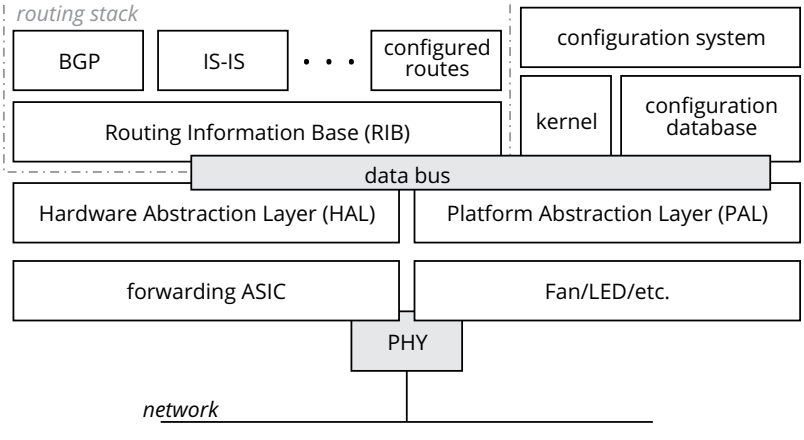


Figure 25-11 *Router components*

In Figure 25-11:

- The forwarding Application-Specific Integrated Circuit (ASIC), fans/LEDs/etc., and PHY (physical network interface chipset) are the only hardware devices shown; the remainder are software components.
- The software components are assumed to run on some local processor, memory, and storage resources; these are not normally shown when considering the architecture of a network device.
- The routing stack consists of two components: the actual routing protocol (or other control plane) applications and the Routing Information Base (RIB).
- The kernel primarily manages processes, including memory and processor usage; the kernel may also provide a communication channel between some pairs of components.
- There may be no, one, or two data busses in the system. If this component exists, it is responsible for providing a standard way of carrying information between the other components in the system, and potentially acting as a data store for system state. The data bus can be implemented as a database or a publish/subscribe system.
- There may or may not be a configuration database. If it exists, the configuration database is responsible for holding configuration state for all the other systems on the device.
- The configuration system provides some way to read and write the configuration of the device. Generally, this will include both a machine-readable (an API) and human-readable interface (a command-line interface [CLI]). The machine-readable interface will be considered more fully in Chapter 26, "The Case for Network Automation."

There is a single term, the *Network Operating System* (NOS), that is often used to describe either

- All of the software components
- The kernel, data bus, and (sometimes) other components, such as the HAL and PAL

Because the meaning of NOS is variable, you need to make certain you understand precisely which components are being included when the term is used, and which are not.

Given this set of components, the interesting disaggregation question becomes: which components can, or should, be split off and developed, owned, etc., by different people? There are several different logical places to place such a divide:

- Between the routing protocols and the RIB
- Between the HAL and the rest of the components
- Between the PAL and the rest of the components
- Between the hardware and the software
- Between the RIB and the data bus
- Between the configuration system and the configuration database
- Between the configuration database and the data bus

Traditionally, all of these components are purchased as a single item—an appliance. To disaggregate the network, you want to be able to break this appliance apart into multiple pieces. It is possible, in a disaggregated model, for an operator to

- Purchase the hardware, HAL, and PAL from one vendor, build his own control plane to run on top of an open source or vendor-provided RIB, and purchase the remainder of the system, which might be called the NOS, from another vendor
- Purchase the hardware from one vendor and the software from another (where the entire software piece may be called the NOS)
- Purchase everything except the system configuration system from a single vendor and build his own configuration system

The key point is the operator must choose which pieces of the network he wants to own, which he wants to purchase, and which disaggregation model makes the most sense for his business. The specific model chosen is going to depend on the business drivers and requirements in a particular environment. Some specific cost advantages that can be realized by disaggregating hardware from software in the network include

- Commoditizing hardware by separating it from the software. If a suite of software can be used across multiple hardware platforms, the hardware capabilities and cost become driving factors, rather than the brand on the outside, and the software bundled with the hardware. This is the primary goal of the white box movement among network operators.

- Providing operational stability through many generations of hardware. If the software and hardware can be replaced or modified separately, then the hardware can be replaced with newer, more capable devices without modifying operational processes and cadence. At the same time, the software can be modified over time without replacing the hardware, allowing for the network to grow and mature without resorting to using a forklift to replace all the equipment at once.

Like disaggregation on the compute and applications front, disaggregation in the network space goes far beyond cost savings. Decoupling the software from the hardware allows the software to be built specifically around the application architecture—remember that disaggregated applications treat the entire network as a single "thing." Just like building a high-performance computer requires tuning and adjusting the hardware to support the specific computing task at hand, building a high-performance distributed application often requires building a network as a platform, tuned to the application to increase performance and focus where the operator spends time into areas with higher returns on investment.

Note

Network engineers do not tend to think about *removing* features, rather than adding them, as a method of tuning for optimal performance. When you build a race car, you do not start by adding a bigger engine; you start by removing the weight of *unnecessary* things. This simplifies the problem set, reduces the number of components needed, and generally makes replacing or refitting the remaining parts a lot simpler, as well as making the car itself simpler to maintain. It is critical for network engineers to get into the habit of thinking about what can be removed, as well as what can be added.

The result is a two-front gain. On one side, hardware is commoditized, driving the cost down. On the other side, the software is customized, providing greater value, and allowing the software to move at the pace of the business. Even in a fully supported disaggregated environment (which *are* available at the time of this writing), it is possible to disconnect the software life cycle from the hardware life cycle. This allows for hardware replacement on a much faster schedule to gain new speeds and feeds, and new switching features, while keeping software in place for a longer cycle, allowing business processes to adjust and work around the software.

Final Thoughts on Disaggregation

The network world is changing rapidly, and disaggregation has played a major role in driving these changes. Will the network, itself, eventually be disaggregated in the same way and for the same reasons? The final chapter, Chapter 30, "Looking Forward," will take a look at the future of networking and try to answer these questions.

Disaggregation in the compute and application space have driven many more changes in the world of IT and in network management. For instance, another form of disaggregation in the network is to divide the services offered by the network itself from the network appliance.

Further Reading

Churchill, Elizabeth F. "Patchwork Living, Rubber Duck Debugging, and the Chaos Monkey." *Interactions* 22, no. 3 (April 2015): 22–23. doi:10.1145/2752126.

"Engineered Elephant Flows for Boosting Application Performance in Large-Scale CLOS Networks." Irvine, CA: Broadcom, 2014. https://docs.broadcom.com/docs/1211168569445?eula=true.

Gill, Phillipa, Navendu Jain, and Nachi Nagappan. *Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications*. ACM, 2011. https://www.microsoft.com/en-us/research/publication/understanding-network-failures-data-centers-measurement-analysis-implications/.

Lapukhov, Petr. "Routing Design for Large Scale Data Centers." British Columbia, Canada, June 3, 2012. https://www.nanog.org/meetings/nanog55/presentations/Monday/Lapukhov.pdf.

Lapukhov, Petr, Ariff Premji, and Jon Mitchell. *Use of BGP for Routing in Large-Scale Data Centers*. Request for Comments 7938. RFC Editor, 2016. doi:10.17487/RFC7938.

Martin Casado, and Justin Pettit. "Of Mice and Elephants." *Network Heresy*, November 1, 2013. https://networkheresy.com/2013/11/01/of-mice-and-elephants/.

Pepelnjak, Ivan. *Data Center Design Case Studies*. ipspace.net, 2014. http://www.ipspace.net/Data_Center_Design_Case_Studies.

Roy, Arjun, Hongyi Zeng, Jasmeet Bagga, and Alex C. Snoeren. "Passive Realtime Datacenter Fault Detection and Localization." In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 595–612. Boston,

MA: USENIX Association, 2017. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/roy.

Singh, Arjun, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, et al. "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network." In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 183–197. SIGCOMM '15. New York, NY: ACM, 2015. doi:10.1145/2785956.2787508.

White, Russ. "The State of Open Source Routers." Presented at the North American Network Operators Group, Bellevue, WA, June 7, 2017. https://www.youtube.com/watch?v=JTQqmnVRToI.

White, Russ, and Denise Donohue. *The Art of Network Architecture: Business-Driven Design*. 1st edition. Indianapolis, IN: Cisco Press, 2014.

Review Questions

1. Research the difference between a virtual machine and a container. Provide a short list of three or four differences between the two.
2. Research the toroid fabric design. How does it differ from the more widely used spine and leaf design? What would be the impact of a toroid design on oversubscription?
3. Explain the differences between a "normal" network and a fabric.
4. The problem of blocking is not truly removed in nonblocking designs but rather moved. Where is it moved to?
5. Explain the difference between elephant and mouse flows.
6. Find two hardware abstraction layers available for network operating systems. Note four differences between these two abstraction layers.
7. Find two open source routing protocol stacks. What protocols are supported? How much support does each of these projects appear to receive?

Chapter 26

The Case for Network Automation

Learning Objectives

After reading this chapter, you should understand:

- The main objectives of network automation
- The basic network management requirements
- The basic purpose and operation of NETCONF
- The basic purpose and operation of RESTCONF
- Some of the languages used in network automation and their attributes
- On-box automation and network controllers

A typical network consists of a collection of distributed nodes, each running an operating system, each configured with protocols and feature sets. Networkwide features—for example a routing protocol—require synced configurations to enable nodes to work together. The number of nodes in addition to the distributed configurations leads to complexity. New nodes or new features increase the complexities of the system, and increased complexity increases the opportunity of failure, increases operational cost, and retards the network's ability to change. These monetary and nonmonetary costs often restrict network engineers from adopting new features, using the network to solve business problems, or understanding network failures.

Network automation can lead to better deployment, operation, and troubleshooting of the network. It can reduce the complexity of network deployment, configurations,

and operations; increase agility, or the ability of the operator to reshape the network to new requirements more quickly; and reduce cost and risk by removing the human element and using automation tools to interact with individual network devices.

Network Automation and Complexity

There is a general assumption in the network engineering field that automation reduces complexity. While automation can provide a simple set of tools to perform repetitive tasks, and it reduces human interaction with individual network devices, it does not reduce complexity in a general sense. Within the State/Optimization/Surface (SOS) model, automation reduces the interaction surface between humans and individual network devices, but it adds more state, in the form of an additional database of configurations and new protocol interactions, and adds an additional interaction surface, in the form of a new protocol to the management plane—the Application Programming Interface (API). Someone must, ultimately, maintain the new tools and APIs created and used in the process of automating.

This does not mean automation is a bad thing; it is a requirement at almost anything beyond trivial scale in network operation. However, engineers should always consider the tradeoffs and prepare for where and how increased complexity may impact network operation. Rather than assuming complexity will be reduced, assume complexity will be moved someplace else, and consider how the movement needs to be managed.

Automation can also act as a form of abstraction, replacing a large configuration with a few simple commands. This is both a good and a bad thing: good, because it ensures the same configuration is done the same way consistently; bad, because abstractions often remove state in one place, while reducing optimization someplace else. Again, this does not mean automation should not be deployed; in many cases, automation is the only path forward. On the other hand, engineers should always look for where optimization might be affected through abstraction, and understand the potential impact on the network. All nontrivial abstractions also leak in some way; it is important to look for and understand—where possible—these leaks.

If you have not found the tradeoff, you have not looked hard enough.

One particular place to consider the tradeoffs is in the complexity and capability of the API and the tools used to access the API versus the amount of state and the breadth and depth of the interaction surface. For instance, to interact with an interface designed explicitly for human interaction generally means a chain of interactions from human to tool, from tool to human-centered interface, then from human-centered interface to the device configuration. The

movement through the human-centered interface adds a lot of complexity and makes the interaction surface very difficult to maintain. Moving from an automation tool through some sort of API designed specifically for machine-to-machine interaction, then to the device configuration, is much simpler, and hence adds much less complexity in the automation system. In fact, the latter case, tool to API to configuration, can *reduce* overall complexity at a system level.

Network automation can be as simple as automatically provisioning new switches in the data center to changing configurations through dynamic software development to automating response to Syslog events. More robust implementations enable network teams to stop thinking about the network as a collection of individual network devices and start thinking about the network as a system.

Network automation has spawned a new title within network teams: the automation engineer. Automation engineers, usually part of the operations team, require advanced network, protocol, and troubleshooting skills; proficiency in a scripting language such as Python or Bash; and the ability to manipulate text, such as using regular expressions. Network automation teams are usually very small.

Note

Regular expressions are a way to match on text strings within a larger text file (such as a network device configuration, or even a book), either to find those strings, or to use a text processor to replace one string with another. More information about the formatting and uses of regular expressions can be found in the "Further Reading" section at the end of the chapter.

Automation Concepts

To automate a network device, a network automation tool requires some method to connect, authenticate, and interact with the management plane of a network node. Traditionally, most if not all network devices feature a command-line interface (CLI). The CLI provides access to the management plane over Telnet or a Secure Shell (SSH), creating what's well known as a human-to-machine system. While CLI interfaces are optimized for humans, they can be automated using tools such as Expect, Puppet, Ansible, Chef, Salt, and CFEngine (see the "Further Reading" section at the end of the chapter for links to information about these tools).

Taking one of these tools as an example: Expect is a scripting language to automate configurations through interactive interfaces—for example, CLI interaction running over SSH. Expect creates a machine-to-human-to-machine system, normally using the CLI as an API (so Expect can be leveraged for any system with a CLI). Expect scripts run a set of commands after the SSH session returns some text. For example, an Expect script to log in may entail the following:

```
expect "Username: "
send "Groot"
expect "Password: "
send "cisco123"
```

The Expect processor examines the text stream in real time, looking for the "Username:" prompt (generally by using some form of regular expression matching engine on the incoming text stream). When this prompt is encountered, the processor sends a text string in reply containing "Groot" based on a previously written script. The same pattern is followed for the password. In the case of a Cisco CLI, Expect may just respond to a command prompt at a particular level, such as the *enable* prompt. Expect is very extensible and can automate any CLI-based feature on a single device or an entire network. In most cases a network automation administrator will use Expect with text parsing and manipulation tools (such as regular expressions) to automate the configuration of a large number of devices, hence managing the entire network with a small number of scripts.

While CLI scripting tools have proven to be very successful and are still in use in modern networks, they are very difficult to build, maintain, and troubleshoot. A common issue with Expect is dealing with what happens when something unexpected happens. If a vendor changes the output of a particular command or prompt, or the order in which commands need to be entered, the change will need to be discovered and the affected scripts modified for the new input/output pattern. For example, a vendor might change "Username" to "username," or even ask for the password first. In these cases, the script will simply not run or throw an error. Additionally, because each vendor has slight variations of CLI, scripting work must be duplicated in multivendor or multinetwork operating system environments. Finally, Expect does not have any implicit understanding of configuration state; thus this logic must be written in the script.

One of the first tools to emerge to better manage and automate networks was the Simple Network Management Protocol (SNMP). SNMP enables network operators to securely connect to a device and use a common (standardized) or vendor-specific Management Information Base (MIB) to interact with it. SNMP was originally designed for both monitoring and configuration management; however, using SNMP

for configuration management has extremely low adoption because it is very difficult to use and usually does not reflect all the capabilities of a node.

SNMP, in dictionary terms, stores the metadata in an MIB specification, while the corresponding state is stored in the information retrieved from the device. In order to retrieve a particular piece of information, the information requested must be specified according to the dictionary rules; for example, a request might look like

```
snmpget -m ../../mibs/RFC1213-MIB localhost .iso.org.dod.
internet.mgmt.mib-2.system.sysDescr.0
```

In order to retrieve information about an entire subsystem, the entire MIB table must be "walked," item to item, with each item being returned separately. The separate items must then be reassembled into their proper form and then interpreted based on the MIB definition to understand what the actual state of the device is. A new method to automate networks was required.

As networks became bigger and more important to application delivery, network operators sought better ways to manage and automate those networks. In 2002 a small group of network engineers held an Internet Engineering Task Force (IETF) workshop to discuss the future of network management and build a high-level architecture for future protocol developments. The results of this workshop are documented in RFC3535, *Overview of the 2002 IAB Network Management Workshop.*[1] RFC3535 discusses current network management technologies, including SNMP and CLI, and more importantly describes 14 requirements for network management and automation protocols. These requirements are

1. Ease of use is a key requirement for any network management technology from the operator's point of view.
2. It is necessary to make a clear distinction between configuration data and data describing operational state and statistics. Some devices make it very hard to determine which parameters were administratively configured and which were obtained via other mechanisms such as routing protocols.
3. It is required to be able to fetch separately configuration data, operational state data, and statistics from devices, and to be able to compare these between devices.
4. It is necessary to enable operators to concentrate on the configuration of the network as a whole rather than individual devices.

1. Schönwälder, Overview of the 2002 IAB Network Management Workshop.

5. Support for configuration transactions across a number of devices would significantly simplify network configuration management.
6. Given configuration A and configuration B, it should be possible to generate the operations necessary to get from A to B with minimal state changes and effects on network and systems. It is important to minimize the impact caused by configuration changes.
7. A mechanism to dump and restore configurations is a primitive operation needed by operators. Standards for pulling and pushing configurations from and to devices are desirable.
8. It must be easy to do consistency checks of configurations over time and between the ends of a link in order to determine the changes between two configurations and whether those configurations are consistent.
9. Networkwide configurations are typically stored in central master databases and transformed into formats that can be pushed to devices, either by generating sequences of CLI commands or by pushing complete configuration files to devices. There is no common database schema for network configuration, although the models used by various operators are probably very similar. It is desirable to extract, document, and standardize the common parts of these networkwide configuration database schemas.
10. It is highly desirable for text processing tools such as diff and version management tools such as RCS or CVS to able to be used to process configurations, which implies devices should not arbitrarily reorder data such as access control lists.
11. The granularity of access control needed on management interfaces needs to match operational needs. Typical requirements are a role-based access control model and the principle of least privilege, where a user can be given the minimum access necessary to perform a required task.
12. It must be possible to do consistency checks of access control lists across devices.
13. It is important to distinguish between the distribution of configurations and the activation of a certain configuration. Devices should be able to hold multiple configurations.
14. SNMP access control is data oriented, while CLI access control is usually command (task) oriented. Depending on the management function, sometimes data-oriented or task-oriented access control makes more sense. As such, it is a requirement to support both data-oriented and task-oriented access control.

Modern Automation Methods

In response to RFC3535, modern automation protocols were developed. These protocols enable better multivendor network management and automation by enabling machine-to-machine interfaces through open standards and methods.

NETCONF

NETCONF was developed by the IETF in response to RFC3535; it is an open standard protocol enabling device configuration and monitoring in either single or multivendor networks. NETCONF works in a client/server model where the server is the network node, and the client is a standalone network management station. The management station provides holistic management of the network and supports networkwide automation, allowing network administrators to address the network as a single entity.

NETCONF features multiple configuration data stores to closely mirror the operational state of a network device, as shown in Table 26-1.

Table 26-1 *NETCONF Data Stores*

Data Store	Purpose
<candidate>	Working copy of the configuration for validation and testing
<running>	The configuration the device is currently using
<startup>	The configuration the device will run when booted

Table 26-1 shows three data stores (or tables), which represent a standard process for updating configurations on network devices:

- The <running> data store is the configuration currently being used, or run on the device.
- The <startup> configuration is what will be run the next time the device boots.
- Candidate configurations are stored in the <candidate> data store and can be manipulated without impacting the running configuration. When a network operator is finished with a candidate configuration, the configuration can be validated for proper syntax, against a set of rules to ensure no configuration items have been missed, or sent through a dry run. For instance, a validator might check to make certain an interface configuration always includes IPv4 and IPv6 addresses, or a routing protocol is configured for both IPv4 and IPv6, or just IPv6. This kind of validation can catch and prevent simple mistakes.

If the configuration is acceptable, it is then committed, or pushed into running configuration. Candidate configurations are often leveraged to schedule commits during outage or change windows. This allows changes to be written and tested before the change window. Because some devices do not support a candidate configuration, NETCONF features a capability exchange with initial HELLO messages. NETCONF configurations are atomic: if any part of the configuration fails or has unexpected results, the entire configuration can be rolled back.

If the running configuration proves to be correct, then it can be committed to the <startup> data store, so the device will boot with this configuration the next time it is restarted. It is also possible for the device to boot with a very simple configuration, which is then modified by a network management station through the <candidate> and <running> data stores.

A key component of NETCONF is the management station. The management station provides a networkwide viewpoint for network management and automation. It will have a graphical user interface (GUI) or CLI interface, enabling network administrators to focus on the deployment automation of an entire networkwide service. A sample service may be provisioning a new VPN customer or changing an SNMP password. Network administrators can use a management station to manage the entire lifecycle of a service. By default, and because of NETCONF, network management stations support multivendor networks. To provide additional extensibility, some network management stations feature other southbound configuration methods or protocols—for example, SNMP or CLI—and robust northbound API for integration into other systems.

NETCONF is a modular protocol, organized in layers, as shown in Figure 26-1.

These layers allow for other tools or protocols to be inserted to extend functionality.

The bottom layer is concerned with the transport of messages between devices. NETCONF supports many different transport protocols; however, SSH is commonly used because it is well known and provides authentication, integrity, and

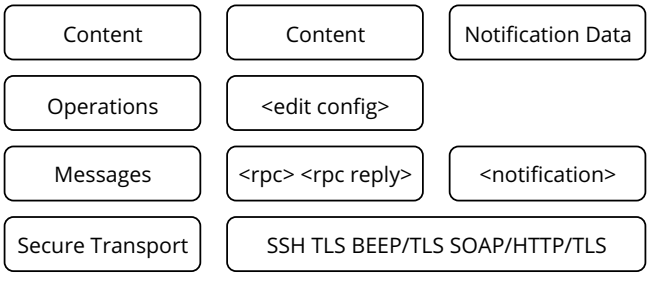


Figure 26-1 *NETCONF Layers*

confidentiality. Because SSH runs over the Transmission Control Protocol (TCP), it also provides reliable transport.

The message layer frames and encodes remote procedure calls (RPCs). An RPC appears to be a local function (or procedure) call to the calling application, but is actually executed on a remote device (see the "Further Reading" section at the end of the chapter for more information on RPCs). NETCONF's use of RPC enables NETCONF to instruct the remote device what to do with the command—for example, apply the configuration detailed in the operations layer. NETCONF RPC messages are encoded in the eXtensible Markup Language (XML) and must contain a message-id element allowing NETCONF to track messages. Finally, the messages layer supports notifications, where devices notify the management station of a configuration change.

The operations layer defines the actions for NETCONF clients and servers. NETCONF operations are a set of create, read, update, and delete (CRUD) actions used on the data stores. Common operations include get-config, edit-config, and delete config.

The base protocol includes the following operations shown in Table 26-2.

Table 26-2 *NETCONF Operations*

NETCONF Operations	Description
get	Retrieve running configuration and device state information
get-config	Retrieve all or part of a specified configuration datastore
edit-config	Load all or part of a specified configuration to the specified target configuration datastore
copy-config	Create or replace an entire configuration datastore with the contents of another complete configuration datastore
delete-config	Delete a configuration datastore
lock	Lock the entire configuration datastore system of a device
unlock	Release a configuration lock
close-session	Gracefully terminate the NETCONF session
kill-session	Force the termination of a NETCONF session

The content layer contains the formatted data, either configuration or notification, sent to or from the network node. The NETCONF specification does not define how the data should be formatted; it does, however, suggest the use of the YANG data modeling language. Data modeling ensures compatibility between systems.

YANG is a data modeling language defined in RFC6020 and updated in RFC7950; it is used to format configuration, notification, and state data in the operations and content layers of NETCONF. Data modeling is a definition of both the syntax and the semantics or schema of the data and is critical when working between remote systems. The data model ensures the requests of the NETCONF management station are faithfully carried out on the NETCONF server (network device).

There are two sets of widely deployed and supported YANG models:

- The IETF standardizes a data model in YANG for each protocol, for basic routing functionality, and for common equipment management requirements.
- The OpenConfig group maintains another set of data models, largely overlapping, and often coordinated with the IETF data models.

Beyond these models, each vendor also supports a vendor- and equipment-specific model set that can (often) be downloaded from their support sites.

The underlying syntax of YANG is XML in a keyed hierarchical model, much like a Type Length Value (TLV) format. The hierarchy of the model enables multiple organized levels of parent/child relationships of key/value paired data. The keys in YANG must be unique within a layer with single values or lists of data. YANG data is typed—for example, integer, string, etc.—and is enforced by the server and client implementations. Because YANG data is in XML, the data is generally human readable. The following code snippet is an example of YANG data showing "show interface brief":

```
01 <?xml version="1.0"?>
02 <nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns="http://www.cisco.com/nxos:7.0.3.I6.1.:if_manager"
     message-id="1">
03 <nf:get>
04 <nf:filter type="subtree">
05 <show>
06  <interface>
07   <brief/>
08  </interface>
09  </show>
10   </nf:filter>
11   </nf:get>
12   </nf:rpc>
13 ]]>]]>
```

In the code snippet, line 1 declares the document as XML. Line 2 is the RPC call from the standard NETCONF library. Line 3 is the NETCONF operation. Lines 5 through 7 are the NETCONF content and the command to show the interface information.

> **Note**
>
> RFC7951 defines JavaScript Object Notation (JSON) encoding of data modeled with YANG. Both XML and JSON are discussed later in this chapter.

Together NETCONF and YANG provide an open-standards-based, easy-to-use toolset to automate a network. Commercial tools (for example, Cisco's Tail-F) leverage NETCONF/YANG and enable network administrators to manage the network in terms of the service the network provides—for example, Quality of Service (QoS) or Virtual Private Networks (VPNs).

RESTCONF

RESTCONF is an emerging extension of NETCONF that leverages the widely deployed Hypertext Transfer Protocol (HTTP) over the Secure Sockets Layer (SSL, combined with HTTPS) to interact with network devices. RESTCONF uses YANG as a data modeling language and has the same basic functionality as NETCONF; however, it uses HTTP methods such as POST, PUT, and DELETE to implement the equivalent of NETCONF operations. The RESTful methods available in RESTCONF enable basic create, read, update, and delete (CRUD) operations on a hierarchy of data stores via HTTP.

The "REST" is in RESTCONF because it provides a RESTful interface; the concept of Representational State Transfer (REST) and RESTful interfaces is discussed further in the next section.

Automation with Programmatic Interfaces

Some modern network operating systems, for example, Cisco NX-OS, offer additional vendor-specific automation programmatic interfaces. These interfaces, known as Application Programming Interfaces, or APIs, are exposed on standard TCP ports and require a higher-level language such as Python to interact with a device. The higher-level program contains the specific interaction—for example, a configuration or troubleshooting routine—and leverages the API to connect to the device. Most

network operating systems require the API to be enabled via CLI and require authentication.

Network APIs are not created or maintained under the guidance of a standards body, and each vendor or even different platforms from the same vendor will have unique methods and procedures to access and use the APIs. Documentation for a network API is found on the vendor's website.

Network automation with APIs offers higher-level applications the capability to interact with a network device. Higher-level programs introduce logic such as IF…THEN…ELSE or on-demand configuration changes that tie the network into business-level systems or procedures. Custom applications with APIs are extremely flexible; however, they often come with a cost in complexity and supportability. Each time a vendor changes a custom API, some amount of work must be performed to adapt applications using those APIs.

A popular way to use network APIs is to connect the network to higher-level automation tools in private cloud environments. Private cloud environments enable automatic provisioning and teardown of the network, storage, and compute to reduce costs and increase the agility of data center infrastructure. The combination of automated network, storage, and compute is commonly referred to as infrastructure as a service and is the starting point for other cloud services, for example, platform as a service (PaaS) and software as a service (SaaS)

Most modern APIs provide RESTful interfaces (or APIs), or rather adhere to REST. Beyond the transport specifications, the most interesting aspect of REST as an API is *no maintenance of state is required at the server.* This means a RESTful operation must complete in a single call and return from the network device's perspective. For instance, say a RESTCONF client needs to configure three static routes on a network device:

1. 2001:db8:3e8:100::1 via 2001:db8:3e8:110::1
2. 2001:db8:3e8:110::1 via 2001:db8:3e8:120::1
3. 2001:db8:3e8:120::1 via 2001:db8:3e8:130::1

And if it installs all three routes at once, the client must handle any dependencies, such as the failure to install one 2001:db8:3e8:120::1, on which the route to 110::1 depends. The server simply does not have any state about the interaction between commands executed, and hence has no way to roll back or otherwise modify one command transmitted serially through a RESTful interface.

REST was originally designed as a paradigm for the HTTP protocol, and hence RESTful APIs are most often implemented over HTTP using common HTTP

verbs, such as GET, PUT, and DELETE. A RESTful client connects to the interface, sends formatted data—for example, a configuration or show command—and the device responds with an HTTP code and optionally formatted data. The returned HTTP code is a standard response; for example, 200 = OK, 401 = unauthorized, etc., informs the client if the command was successful. Figure 26-2 illustrates.

The data sent and received in a RESTful connection requires some structured formatting. Common data formats include XML, JSON, and YAML (originally standing for *Yet Another Markup Language,* but later changed to *YAML Ain't Markup Language,* a recursive acronym, like GNU, which means *GNU's Not UNIX).*

The following snippets illustrate three markup systems often used to format data in a RESTful interface. XML is the first example:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
 <Beatles>
  <Revolver>
   <Songs>
    <element>Taxman</element>
    <element>Eleanor Rigby</element>
    <element>I'm Only Sleeping</element>
    <element>Love You Madeline</element>
    <element>Here, There and Everywhere</element>
    <element>Ellie Said She Said</element>
    <element>Good Day Sunshine</element>
    <element>And Your Bird Can Sing</element>
    <element>For No One</element>
    <element>Doctor Alex</element>
    <element>I Want to Tell You</element>
```

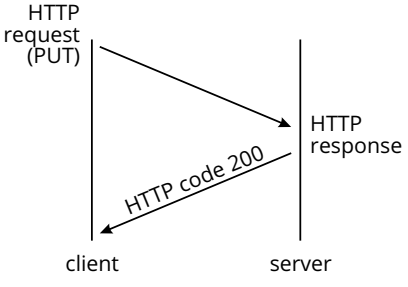


Figure 26-2 *A RESTful Operation*

```
    <element>Got to Get You Into My Life</element>
    <element>Tomorrow Never Knows</element>
   </Songs>
  </Revolver>
 </Beatles>
</root>
```

XML is a markup language that encodes information between descriptive tags (XML is a superset of the Hypertext Markup Language, or HTML, which was originally designed to describe the formatting of web pages served by servers through HTTP). The encoded information is defined within user-defined schema that enable any data to be transmitted between systems. In the case of network automation, XML-encoded data may be a single command or an entire configuration. The entire XML document is stored as text, making it both machine and human readable.

YAML is the second example:

```
Beatles:
 Revolver:
  Songs:
   - Taxman
   - Eleanor Rigby
   - I'm Only Sleeping
   - Love You Madeline
   - Here, There and Everywhere
   - Ellie Said She Said
   - Good Day Sunshine
   - And Your Bird Can Sing
   - For No One
   - Doctor Alex
   - I Want to Tell You
   - Got to Get You Into My Life
   - Tomorrow Never Knows
```

YAML, considered a subset of JSON, is designed to be very human readable. Similar to JSON, YAML is structured in key|value pairs and allows for user-defined white space. The extra white space enables the readability of YAML documents but can be resource intensive to parse.

JSON is the third and final example:

```
{
 "Beatles": {
  "Revolver": {
```

```
   "Songs": [
    "Taxman",
    "Eleanor Rigby",
    "I'm Only Sleeping",
    "Love You Madeline",
    "Here, There and Everywhere",
    "Ellie Said She Said",
    "Good Day Sunshine",
    "And Your Bird Can Sing",
    "For No One",
    "Doctor Alex",
    "I Want to Tell You",
    "Got to Get You Into My Life",
    "Tomorrow Never Knows"
   ]
  }
 }
}
```

A more recent alternative to XML is JSON. JSON is defined in RFC4627, and encodes information in structured key|value pairs. The keys in a JSON document are predefined tags that are understood between systems; each such tag has a single associated value. For example, a key of "Command" may have a value of "show running configuration." In typical cases, a key is a list of values, represented with open and close brackets—for example, "[data1, data2, data3]." Similar to XML, JSON is stored as a text file and is both human and machine readable; however, JSON is much easier for humans to interact with. The main advantage of JSON is that it is straightforward to parse because a key can reference values.

REST, XML, JSON, and YAML are all supported in a variety of different programming languages, including C, Java, and Python. Python, mainly because of its ease of use, is the unofficial standard for network programmability and automation projects. The Python language is easy to write, hard to mess up, and is supported on most operating systems. Python supports thousands of libraries that extend the language to support a wide variety of technologies.

APIs, Python, REST, and JSON come together to automate a network or network device via programmability. Most modern network operating systems require the API to be enabled on a particular TCP port and configure an authentication method. Then a different computer invokes a Python program to interact with the node. Figure 26-3 illustrates.

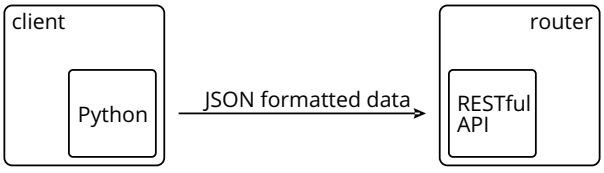


Figure 26-3 *Programmability*

On-box Automation

Many network devices also support on-box automation. On-box automation is a script or procedure running on the management plane of a network device, enabling network operators to automate configurations or events that are local to the network device. Because the on-box automation scripts are distributed with the network device, they are better at handling link failure or isolation-type events. On-box automation tools consist of vendor-specific offerings—for example, Cisco Embedded Event Manager (EEM), Python, or Linux Bash scripts.

Cisco EEM is a popular on-box automation tool for Cisco devices. Cisco EEM features event detectors—for example, an environmental issue or routing protocol adjacency change and the ability to tie an action to an event. A common example of EEM is to, in the event of a downed interface, automate a response of "shut, no-shut" the interface, or collect information about processes and memory usage when the processor utilization rises above a specific percentage. EEM actions support CLI-based responses or more complex actions with Python or TCL scripts.

Some network devices such as Cisco Nexus support on-box automation with Python or BASH scripts. Python or BASH scripting is normally available on network devices running Linux as the underlying OS; it allows the network administrators to automate network or device functions with the flexibility of Python. A sample on-box script may perform an action or generate an alert on bootup or after someone has logged in with privileged access. On-box Python scripts can simulate or replace features that are not available on a deployed platform.

Network Automation with Infrastructure Automation Tools

Infrastructure automation tools are designed to manage and automate operating systems, network devices, or resources. Infrastructure automation tools can be used for network automation; however, these tools are more common in agile software

development tool chains, such as DevOps. Infrastructure automation tools will connect and authenticate with a network device and use either the CLI or an API to make changes. They will have a playbook or manifest detailing how to interact with a specific vendor device for a specific feature. Infrastructure automation tools enable the network to be represented as code, known as Infrastructure as Code (IaC). IaC enables agile network configurations because a DevOps team deploys or changes network resources as part of a software rollout. To date, a number of infrastructure automation tools are available, but the open source tool Puppet is most popular.

Note

DevOps is *development operations* contracted into a single word. The general idea of DevOps is to use development processes to manage the operational tasks of running a network, such as managing configurations and versioning.

The Puppet software package, developed by Puppet Labs, is an open source automation toolset for managing servers and other resources by enforcing device states, such as configuration settings.

Puppet components include a puppet agent that runs on the managed device (node) and a puppet master (server) that typically runs on a separate dedicated server and serves multiple devices. The operation of the puppet agent involves periodically connecting to the puppet master, which in turn compiles and sends a configuration manifest to the agent; the agent reconciles this manifest with the current state of the node and updates state based on differences.

A puppet manifest is a collection of property definitions for setting the state on the device. The details for checking and setting these property states are abstracted, so a manifest can be used for more than one operating system or platform. Manifests are commonly used for defining configuration settings, but they can also be used to install software packages, copy files, and start services.

Network Controllers and Automation

A relatively new component in networking is a network controller. Network controllers provide holistic management of a distributed network and a single interface for network automation and programmability. Controllers build an abstraction layer to simplify network management, making network automation easier. An abstracted configuration for a network enables networkwide configurations—for example, setting a new NTP server on a number of devices. In this case, the network operator

would simply set the configuration in the controller, and the controller would deal with connecting, authenticating, and ensuring the configuration is set on every device, as illustrated in Figure 26-4.

Some network controllers feature out-of-the-box automation to deploy and manage networks. For example, the Cisco APIC data center controller automates the deployment of VXLAN as well as many other technologies. Additionally, network controllers simplify deployment of network features by automating complexity and providing guided GUI-based configurations.

Network Automation for Deployment

Deployment automation, also known as zero-touch deployment, automates the deployment of new network nodes. Automated deployments ensure new additions to the network infrastructure, either from initial deployment or replacement from failure. Deployment automation reduces the time, risk, and expense of deploying new nodes.

Deployment automation technologies require a device to request deployment automation from a deployment server. A device may have a configurable flag to request a configuration at next boot or the request can be as simple as lack of a configuration. The network device will find an automation server using information discovered through DHCP or with broadcast technologies. The node will then ask the automation tool for a configuration server. The configuration server must respond to the request using a templated configuration that may be customized by the automation tool. A final step for a deployment automation tool is to notify the network administrator a new device has been added.

Deployment automation tools are available for data center, campus, and wide area network (WAN) environments. To date, there is no standard for deployment

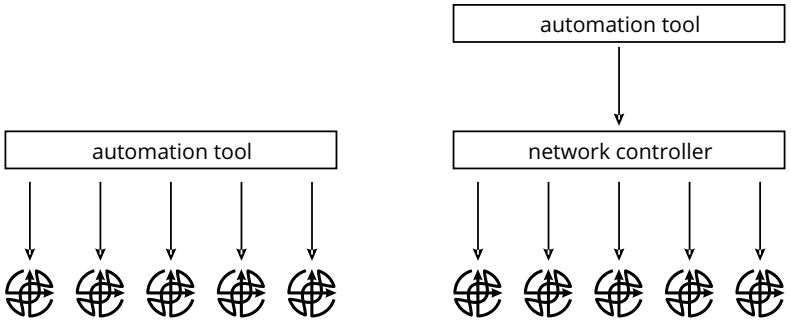


Figure 26-4 *Box-by-Box versus Controller-Based Automation*

automation solutions, and each vendor brings proprietary solutions to market. These solutions are normally a component of larger network management tools, such as Cisco Prime for WAN/campus environments or Cisco Data Center Network Manager (DCNM) for data centers.

Final Thoughts on the Future of Network Automation: Automation to Automatic

An emerging and very popular world of data analytics and machine learning will power the next generation of network automation. *Data analytics* is a general term for a series of tools allowing the collection of data and transforming the data into organized and insightful information. Much of the data that network devices generate today is discarded. This discarded data could give network operators better insight into the status and health (logs) of network devices (configurations), network traffic, or the health of applications traversing the network.

Machine learning enables computers to predict events from the data. For example, machine learning may predict a security issue or an expected traffic load. Machine learning systems can then change network configuration based on its predictions without human intervention. The combination of data analytics, machine learning, and network automation will enable self-provisioning, self-healing networks and the transition to automatic networks.

Further Reading

Other resources on network automation and programmability:

Ansible by Red Hat. "Ansible Is Simple IT Automation." Accessed September 3, 2017. https://www.ansible.com.

Bjorklund, Martin. *The YANG 1.1 Data Modeling Language*. Request for Comments 7950. RFC Editor, 2016. doi:10.17487/RFC7950.

———. *YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. Request for Comments 6020. RFC Editor, 2010. doi:10.17487/RFC6020.

"CFEngine—Automate Large-Scale, Complex and Mission Critical IT Infrastructure with CFEngine." *CFEngine*. Accessed September 3, 2017. https://cfengine.com/.

"Chef: Automate Infrastructure and Applications." *Chef*. Accessed September 3, 2017. https://www.chef.io/.

"Expect—Expect—Home Page." Accessed September 3, 2017. http://expect.sourceforge.net/.

"Extensible Markup Language (XML)." Accessed September 3, 2017. https://www.w3.org/XML/.

Goyvaerts, Jan. "Regular Expressions: The Complete Tutorial," July 2007. https://www.princeton.edu/~mlovett/reference/Regular-Expressions.pdf.

"Grpc / Grpc.io." Accessed September 3, 2017. https://grpc.io/.

Harrington, David, Bert Wijnen, and Randy Presuhn. *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*. Request for Comments 3411. RFC Editor, 2002. doi:10.17487/RFC3411.

Marshall, A. D. "Remote Procedure Calls." In *Programming in C: UNIX System Calls and Subroutines Using C*, 2005. https://users.cs.cf.ac.uk/Dave.Marshall/C/node33.html.

Meyer, Paul, David B. Levi, and Bob Stewart. *Simple Network Management Protocol (SNMP) Applications*. Request for Comments 3413. RFC Editor, 2002. doi:10.17487/RFC3413.

"Netconf Central." Accessed September 3, 2017. http://www.netconfcentral.org/.

Petrusha, Ron. "Regular Expression Language—Quick Reference." Documentation, March 2017. https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference.

Presuhn, Randy. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. Request for Comments 3416. RFC Editor, 2002. doi:10.17487/RFC3416.

"Puppet—The Shortest Path to Better Software." *Puppet*. Accessed September 3, 2017. https://puppet.com/.

Schönwälder, Jürgen. *Overview of the 2002 IAB Network Management Workshop*. Request for Comments 3535. RFC Editor, 2003. doi:10.17487/RFC3535.

Thurlow, Robert. *RPC: Remote Procedure Call Protocol Specification Version 2*. Request for Comments 5531. RFC Editor, 2009. doi:10.17487/RFC5531.

Tischer, Ryan, and Jason Gooley. *Programming and Automating Cisco Networks: A Guide to Network Programmability and Automation in the Data Center, Campus, and WAN*. 1st edition. Indianapolis, IN: Cisco Press, 2016.

White, James E. *High-Level Framework for Network-Based Resource Sharing*. Request for Comments 707. RFC Editor, 1975. doi:10.17487/RFC0707.

"XML Tutorial." Accessed September 3, 2017. https://www.w3schools.com/xml/.

Review Questions

1. What is the primary objective for automating network device configurations?
2. What are the advantages and disadvantages of SNMP?
3. Explain the relationship between NETCONF, YANG, and a YANG model.
4. What is the difference between NETCONF and RESTCONF?
5. Research the concept of an ATOMIC operation. How is this similar to, or different from, a RESTful interface?
6. What is the difference between development operations, or the automation of network configurations, and Software-Defined Networks (SDNs)?

This page intentionally left blank

Chapter 27

Virtualized Network Functions

Learning Objectives

After reading this chapter, you should understand:

- The relationship between network function virtualization and network design flexibility
- The relationship between NFV, scale up, and scale out
- The concept of service chaining, and how it can be used in NFV
- The concept of intent-based networking
- The tradeoffs in computation power between specialized and general-purpose processors
- Some of the tradeoffs involved in NFV

Sue had a problem. The infrastructure team had just stood up several new racks of servers, all of which were going to be running workloads needing firewall and load-balancing services. Sue had firewalls and load balancers, but they were not in a place where she could easily provide access to them from the location of the new racks.

The task was not impossible, of course. She created new virtual networks from each of the new racks, and added them to the tagged VLAN interface of the load balancers and firewalls. From there, she created new subinterfaces, added appropriate route statements, and made it work. Traffic was backhauled from the new racks, back through the core, shipped to the load-balanced segment, passed through the firewall, and back. The job was done.

However, the load-balancer configuration was beyond unwieldy, supporting thousands of virtual servers, pool members, and health checks in a chaotic mess overwhelming the management interface. The cluster was also a bottleneck at certain times of the day, processor-bound on the one hand and experiencing congested network links on the other. Already a substantial cluster of nodes to handle the volume of traffic and requests, the cluster capabilities were never quite able to stay in front of the demand for new load-balancing services.

The data center firewall cluster was in much the same state. Containing a massive security policy choked by multiple thousands of rules, the firewall cluster was becoming an intractable bottleneck. The policy was an administrative nightmare, filled with rules authored by a myriad of administrators who had come and gone over the years. Updates to the security policy were the bane of Sue's existence. She could never seem to find quite the right place to install fresh rules. At the same time, she was afraid to simplify the policy by deleting existing rules for fear of breaking a critical business service.

Like the load-balancer cluster, the firewall cluster was also becoming a performance bottleneck. As compliance requirements demanded both stateful and deep packet inspections for much of the company's traffic, the architecture team directed ever increasing amounts of traffic through the firewall cluster. The cluster could no longer keep up. Some days, she thought she could feel the heat from their processors right in her cubicle, watching sustained 60%, then 70%, then 80% steady utilization during peak business hours in recent months.

Sue explored growing the clusters even further, but this solution would address the capacity problem, and even then temporarily. She also needed a way to move both load-balancing and firewall services closer to the new racks rolling in month after month as their customer base steadily rose.

She also wanted to reduce the administrative nightmare these clusters had become. She had to admit scrolling through massive configuration paragraphs while tired and under time pressure was eventually going to result in an outage—probably a big one. Endless complexity was not the sort of challenge humans were designed to deal with effectively. One of these days, she was going to make a mistake—a "resume-generating event," as the operations team always joked about in the cafeteria. Sue wanted to turn over the rote configuration tasks to an automated system but was struggling to sort out exactly how.

As time went on, Sue researched how to address these challenges, which she compartmentalized as follows:

1. **Backhauling traffic to specific network locations was too limiting.** Sue wanted to be able to move the services to where they were needed, and not move the traffic to where the services were.

2. **Network services needed to be able to scale easily.** Adding new cluster members was too hard with too much operational overhead. And besides, the load problem was not a problem 24×7. Only during peak hours was there a need for more capacity. Sue wanted to be able to shrink network services when she did not need them as well as grow them on demand.
3. **Provisioning of new network services needed to be done quickly and with limited chance for error.** Thus, the administrative domains had to become more manageable. Multiple thousands of rules or virtual servers or routing configuration stanzas needed to be broken up into smaller, easier-to-automate-and-understand chunks.

The solution Sue found was Network Function Virtualization (NFV). Much like the compute world has turned bare-metal machines into virtual ones, the networking world has turned bare-metal routers, switches, firewall, and load balancers into virtual ones.

Using VNFs, Sue moved network services close to the new racks. Rather than backhauling services to some central location, Sue stood up virtual network services in the same racks where the workloads were running. With this approach, she gained flexibility.

Sue was also able to gain scalability using the NFV approach. Rather than add new cluster members, Sue would stand up additional virtual network service instances when load required.

Sue found many orchestration systems able to handle the spin-up and spin-down of virtual network functions for her. She was even able to integrate many of these automated tasks into the larger compute stack orchestration scheme. When operations would stand up a new workload, the networking services required would come up right along with it, all handed by the orchestrator.

Sue moved her role from one of endless error-prone provisioning to one of orchestration and automation system operator.

Network Design Flexibility

Network functions virtualization (NFV) takes network functions once run on dedicated network hardware and repackages them so they can run on generic x86 hardware. “Generic x86 hardware” means a general-purpose hardware platform running an x86 instruction set. Servers and PCs running Linux and Windows operating systems and hypervisors such as Xen Server or VMware ESXi fall into this category.

A network function that has been virtualized in this manner is called, cleverly enough, a Virtualized Network Function (VNF). You heard this right: NFV is made up of VNFs. The critical word to focus on is virtual. Sue can find the answers to many of her architectural challenges through virtualization.

First, consider one of Sue's initial challenges: backhauling traffic. In Sue's scenario, she needed to move traffic between the hosts and a service's clusters. Consider Figure 27-1.

Traffic is funneled from several hosts requiring, in this example, load-balancing servers from hosts uplinked to top-of-rack switches (ToRs), passing into core switches, and eventually making it to a load-balancing cluster. These appliance clusters can be scaled up by increasing the amount of processing power in order to handle larger numbers of transactions. This creates a natural bottleneck. Host traffic must funnel from a collection of ToRs creating a wide mouth down a comparatively narrow-mouthed location where the load-balancing services were offered in a physical form factor via several clustered hosts. The wider the mouth becomes, the more egregiously the bottleneck is felt.

Contrast this traffic pattern with the flow of traffic in Figure 27-2.

In this scenario, a series of virtual load balancers have been created that can scale horizontally, or scale out. Rather than traffic being forced through a funneled

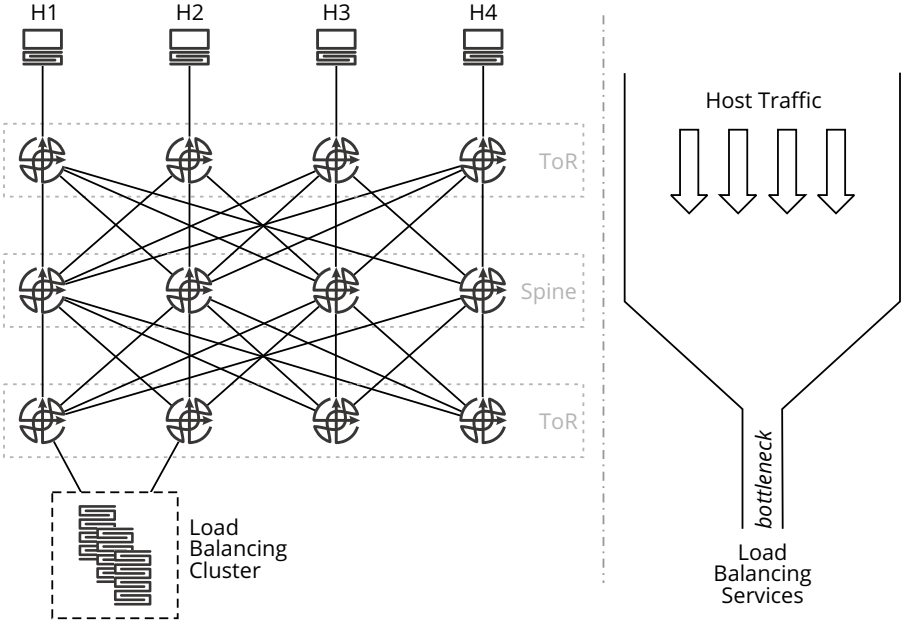


Figure 27-1 *Bottleneck into Fixed (Appliance-based) Services*

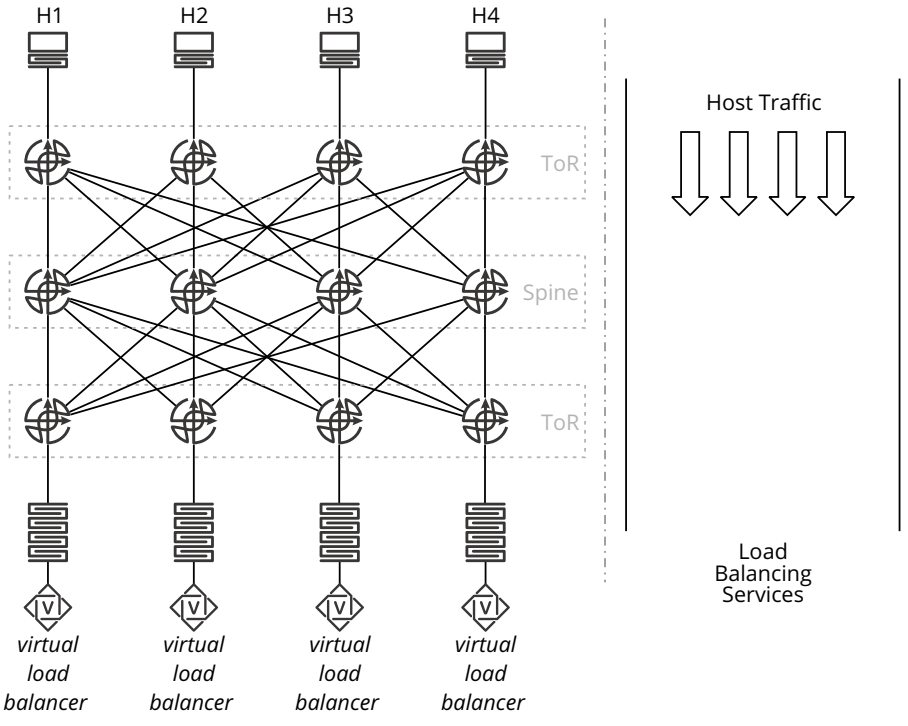


Figure 27-2 *Virtualizing Functions to Resolve the Bottleneck*

bottleneck, a wide load-balancing mouth matching the host traffic is created. The funnel is eliminated.

Service Chaining

A single type of traffic flowing into and out of a single type of service is somewhat simple to solve, but in real-world data centers, traffic flows will more likely need to flow through several different VNFs. For instance, a traffic flow might be part of a load-balancing scheme, but might also require packet filtering and deep packet inspection. Traffic must be routed to each of these services in the correct order. While some flows may need to be routed through every service available in the network for processing, others may need to flow through just a subset of the available services. This is a much more difficult problem to solve.

In a traditional network model, traffic would flow through required services because the network was plumbed to make it happen; physical appliances are physically wired into the network so traffic can only be routed through the correct set of services. For instance, in between a client and a server, a firewall would be installed. An inline load balancer, too, would be placed in the path. Traffic would naturally

flow through the required services because of the routing architecture created by a network engineer, as illustrated in Figure 27-3.

What happens if some particular traffic flow between the client and the server needs a slightly different set of rules applied in the firewall, or does not need to be managed by the load balancer? Each appliance along the path must be configured with some way to detect which flows they must manage, and with specific instructions on how to manage each one. Over time, in a network with hundreds of thousands of flows, the amount of configuration—and the amount of work required to manage those configurations—increases to become unmanageable. Of course, the configuration process can be automated, but this does not *remove* the complexity involved in the configurations, but rather *moves* the complexity someplace else in the network. Instead of humans managing these complex configurations, operators are managing configuration management systems—and these systems, themselves, tend to increase in complexity over time as new requirements are overlaid onto the network.

VNF not only allows network operators to eliminate the physical appliance bottleneck, but it also allows individual virtual appliances to be created for each type of traffic flow in the network. Each virtual appliance can have a much simpler configuration, because it can be inserted into the path of a small subset of the flows passing through the network.

But these two possibilities—virtualizing functions to avoid the topological (or physical) bottlenecks imposed by installing physical appliances in a network to provide services, and virtualizing functions to narrow the focus of any particular function instance to reduce complexity—require a new way of thinking about how to direct traffic through the network. In a VNF scenario, traffic is not naturally going to flow through necessary services when passing between client and server. Since the services required have been virtualized, they no long sit on the wire with physical plumbing and a routing architecture conveniently herding traffic through the services required. Rather, VNFs are virtual, residing out of the physical path of the network functions required.

One way to address this concern is through service chaining. Service chaining steers traffic between network functions before allowing it take its natural path to its destination. Consider Figure 27-4.

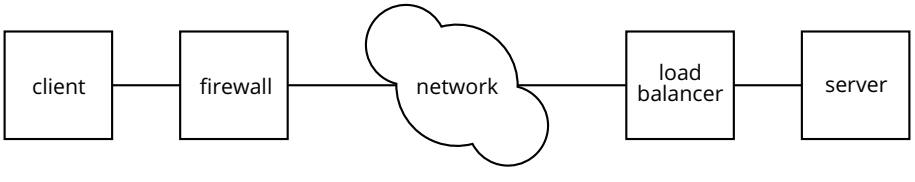


Figure 27-3 *Connecting Services in an Appliance-based Network*

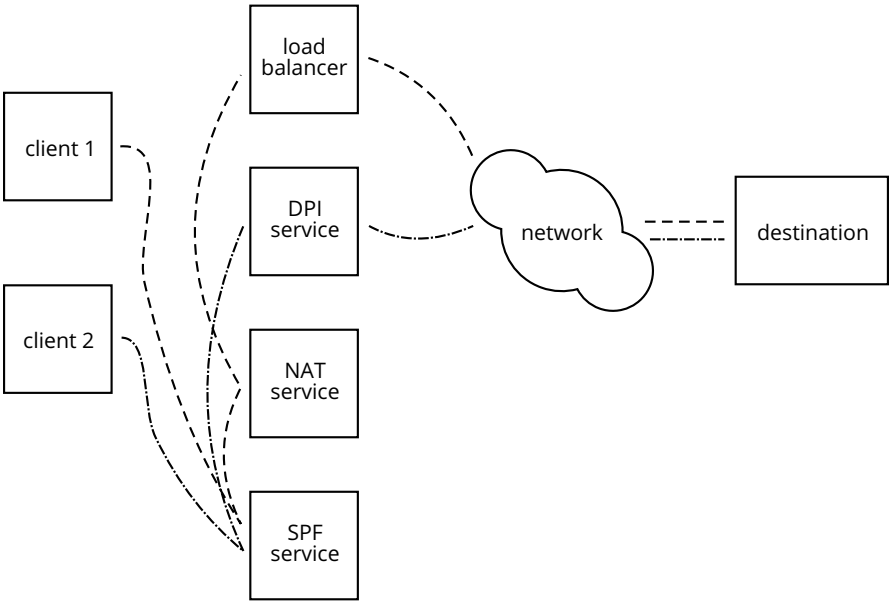


Figure 27-4 *Service Chains*

Two different paths are represented through the virtualized functions in Figure 27-4:

- Traffic from client 1 is passed through a Stateful Packet Filter (SPF) service, then to a Network Address Translation (NAT) service, then to a load balancer, and then finally passed out to the network toward its final destination.
- Traffic from client 2 is passed through an SPF service, then through a Deep Packet Inspection (DPI) service, and then through the network toward its final destination.

Each flow can now pass through just the set of services required, based on specific flow-based requirements. Bypassed services do not need to be configured to ignore flows that they do not need to touch, nor to switch packets related to ignored flows, which both simplifies configuration and reduces unnecessary load on the service.

But how can traffic be chained through services in this way? Service chaining is a nascent technology in networking. At the time of this writing, several industry standard bodies are actively working to standardize the approach.

Note

Two organizations, the Internet Engineering Task Force (IETF) and the European Telecommunications Standards Institute (ETSI), are active in building standards for network function virtualization. Documents in these areas can be found at https://datatracker.ietf.org/wg/sfc/documents/ and http://www.etsi.org/technologies-clusters/technologies/nfv; readers are also referred to the "Further Reading" section at the end of the chapter for specific documents useful for developing a deeper understanding of the technologies and architectures involved in NFV.

Historically, manually installed policy-based routing (PBR) has accomplished service chaining by making a forwarding decision based on the characteristics of a specific traffic flow. For example, traffic from a host with a specific Internet Protocol (IP) address might be routed to the interface of the firewall. PBR has been used by network engineers for exception routing. When traffic needs to go some way other than the standard way indicated by the Forwarding Information Base (FIB), a routing policy is installed to override the FIB.

Therefore, in a limited sense, PBR might function as a service chaining tool, but it is better suited for legacy network topologies featuring physical appliances, rather than VNF scenarios. PBR is notoriously difficult to manage and is only locally significant. For a PBR scheme to be effective, a PBR policy must be installed every hop along the way traffic steering might need to occur. Otherwise, traffic will cease to be chained and will end up being forwarded in accordance with the FIB.

In addition, PBR introduces the same sort of inflexibility that physical appliances do. The entire traffic steering system becomes dependent on predictability. The physical appliances must be in a predictable place. The network architecture must be predictable. But in NFV, the primary goal is for flexibility—a dynamically changing network design and VNFs that come and go as the situation demands.

One way to achieve this flexibility is through Service Function Chaining (SFC), which is evolving as a standard way to route traffic flows through an architecture of VNFs. In SFC, a flow is assigned a Network Service Header (NSH), which contains a service path identifier defining both the services and the order the services are to be traversed by the traffic flow. A companion service index assists with path validation and loop prevention. Moving the traffic flow along the chain from service to service requires Ethernet or IP source and destination addresses, the same as it ever has. The service plane created by NSH maps the service path identifier and service index to an overlay; the flow's packets will be encapsulated to route them across each link of the service chain.

NSH might map to a number of encapsulations, including VXLAN, GRE, and plain old Ethernet. NSH's service-plane means service chaining is topology independent, a crucial feature for services deployed as VNFs.

As with many other things in networking technologies, there are a number of ways to move traffic along a service chain in a network. The chapter describes using an NSH, which is a separate header included in a packet, but there are other ways to direct traffic along a specific path in a network, as well. For instance:

- By building a label-switched path through the network, where each network device reads the outer label, swapping labels to direct each packet to the required hosts connected to the network; this is similar to MPLS Traffic Engineering (TE).
- By building a stack of labels, with each label in the stack representing a hop in the network or a service (virtual device) the packet needs to visit to complete its service chain; this can be done using Segment Routing (SR), for instance.

Each of these solutions has various positive and negative attributes, but the solution deployed in any particular network will mostly depend on hardware support, who owns the applications (how easily the applications can be modified to support service chaining natively), and whether there are requirements for an overlay to solve other problems in the network.

Figure 27-5 is used to illustrate a service chain through a network.

In Figure 27-5:

1. A policy is set through an automated process, manual configuration, orchestration system, etc., that specifies a particular flow originating at H1 needs to pass through DPI and SPF services before being sent to a particular service for processing. There are several instances of the destination service, so the flow must also pass through a load balancer. This policy is injected into the network either at the originating process or host (if either has the ability to impose service chain headers of some type onto transmitted packets), or configured as a filter with an imposed service chain header at the first hop router—in this case, a ToR device on a data center fabric.
2. The traffic is forwarded to the first service indicated on the service chain; if an IPv6 NSH is being used, the network devices will forward the packet based on the first, or "top," service in the service chain, rather than based on the destination IP address. If some form of label swapping or stacking is being used, the packet will be forwarded based on the outermost label in the stack. When the traffic reaches the virtual DPI service, the contents of the packet are inspected for malware, etc.
3. The first segment in the service chain is removed from the packet header and the packet transmitted back onto the data center fabric toward the second service. Again, the network devices need to forward the packets in this flow

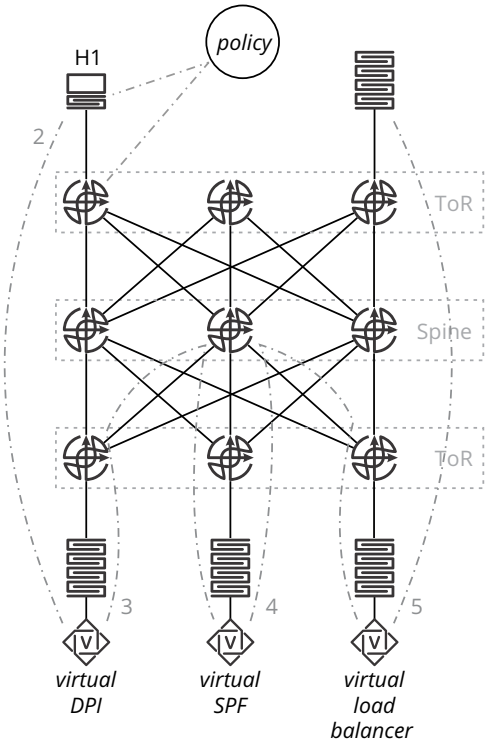


Figure 27-5 *A Service Chain through a Network*

based on the "top" service on the service chain, rather than the destination IP address.

4. When the packet arrives at the second virtual service, it is matched against local state in the stateful packet filter to ensure H1 is allowed to access the destination service, there is an existing flow, etc. The top service is again stripped off the service chain (or labels removed/swapped as needed), and the packet is forwarded back onto the data center fabric.

5. When the packet arrives at the virtual load balancer, the load balancer will check to see if it is part of an existing flow, and modify the label, NSH header, or other information to ensure the packet is forwarded to the correct destination server out of the group of servers providing the destination service. At this point, the IPv6 NSH and/or flow labels may be removed, and the packet forwarded using native IPv6 lookups through the data center packet to the final destination server. The packet is then forwarded one more time onto the data center fabric for delivery to its final destination.

This process may appear to be complex, but it is much less complex than wiring the entire network so that the traffic in this flow would pass through three separate appliances, and managing the configurations on each device on a per-flow basis.

Scaling Out

Network design flexibility means organizations can create VNFs when they need them. Rather than force all traffic through a single massive load-balancer instance or gargantuan appliance-based firewall, services can be spread across many smaller instances. With service chaining, there is no longer a dependency on network placement for where those VNFs are created. Traffic can be directed by policy and service chain to wherever the servicing VNF is located.

This scale-out strategy might sound like simply adding members to a physical cluster to increase capacity. However, in a sense, clustering is nothing more than scaling up, and not out. A cluster with increased capacity still functions as a unit rather than discrete units. The result of adding cluster members to a network service function is an increase in processing capacity but does not come with the advantages that true scale-out architecture offers.

For example, all clusters must remain in full contact with either one another or a cluster controller. When this contact is broken, the cluster is said to be partitioned. While partitioned, each partition will function as its own cluster, a condition known as split-brain. When the partition is healed, the cluster must reconcile, sorting out the inevitable differences resulting from the partitioned clusters acting independently.

Clusters are also subject to major system failures, where the entire system might be taken offline due to an operating system fault or coordinated attack.

Truly scaling network functions out using VNFs breaks service functions down into discrete, independently functioning units. While VNFs are highly likely to be managed centrally, they do not operate as a single device. Therefore, they are not subject to the foibles of partitioned clusters or attacks.

When a VNF fails, the blast radius is limited to the traffic flowing through one VNF. Other VNFs performing identical functions are not affected by the failed VNF. A single small instance failing hurts a production computing environment much less than a massive cluster failing.

By way of example, consider an imaginary payment processing organization called NetBuckPay. NetBuckPay offers several payment gateways to its customers across several different networks they connect to. One of the gateways is an XML service. Another uses JSON. Another uses a proprietary format.

If NetBuckPay was using the legacy network services model, it might use a large firewall cluster for security, an intrusion detection cluster for deep packet inspection, and a load-balancing cluster to spray transactions across pools of gateway servers.

What happens if…

- The load-balancing cluster fails?
- The firewall cluster fails?
- The intrusion detection cluster fails?
- The network path between any of these clusters fail?
- Any of the clusters becomes overwhelmed with traffic?
- Any of the clusters is attacked?

The observant reader will argue a well-designed cluster can tolerate the outage of a member and a well-designed network would tolerate a failure in the network path. The observant and experienced reader will also know systems tend to fail in complex and unexpected ways. Savvy information technology architects are always looking for ways to reduce the potential blast radius of a failed system.

In a worst-case scenario where a redundant system fails in a spectacular and unexpected way, what is the blast radius? NetBuckPay would lose all three payment gateways it offers to its customers until the failure recovery was complete.

If NetBuckPay was using a VNF model, it would be possible to dedicate VNFs for load-balancing, stateful packet filtering, and intrusion detection services to each gateway. Assuming competent design, this would reduce the blast radius of a failure to a single payment gateway. Rather than all payment gateway customers being offline due a commonly shared resource failure, select customers would be impacted as the result of a contained, discrete failure.

Decreased Time to Service through Automation

Reflecting on Sue's challenges, one of them was difficulty in provisioning. As network services become increasingly utilized, their configurations become increasingly complex. Network services perform a central function, but the way in which the function is performed along with unique handling for specific situations results in lengthy command-line stanzas describing how the device is to behave. For devices driven by a graphical user interface (GUI), pages upon pages of screens with configuration information populate the interface.

For network operators, configuration management is a critical part of their roles in an IT organization. Managing configurations effectively and accurately is an essential part of bringing an application to life.

But as Sue recounted, configuration management is also the most error-fraught. With large configurations come opportunities for a human to get lost in the configuration. The more configuration objects there are to collide with, the more difficult it is to make additions that do not disturb the existing configuration functionality. Conversely, deleting what seems to be stale configuration data is risky, as proving what configuration elements are or are not in use is challenging.

Centralized Policy Management

VNFs help networkers with the configuration problem. Assuming manual configuration is being done, a single VNF dedicated to a specific purpose will have a much smaller set of configuration data that a human being must work through. This reduces the opportunity for error as well as the time required to simply sort out what the appropriate configuration might be.

However, VNFs are often managed in an automated way, where a central policy manager stands up and tears down services. The human interacts with the central policy manager. The policy manager handles the VNFs and their configuration.

As VNFs have grown in popularity, the techniques used to manage their policies have evolved. One example to consider is stateful packet filter policy management. Traditionally, stateful packet filters have been managed by rules permitting or denying traffic flows at a very detailed level, potentially including the source IP address, the destination IP address, the source port number, the destination port number, whether there is an existing session, and even various Transmission Control Protocol (TCP) flags. In other words, granular flow information is used to describe each rule.

Applications often use several different ports to communicate. Hosts might use several different IP addresses to communicate. Therefore, building a stateful packet inspection policy out of granular rules is enormously challenging. The challenge grows as the number of applications that need be permitted through the stateful packet filter grows and as the number of hosts involved in serving the applications grows. Over time, traditional stateful packet filtering policy management fails.

Traditional stateful packet filter policy management is not a realistic option when considering many small packet filters deployed as VNFs. To handle VNF packet filter management, central policy management is used. A single policy leveraging metadata is written.

In this context, metadata refers to less granular ways to group objects. For example, users might be grouped by an object in Microsoft's Active Directory. Applications might be grouped by name. Hosts might be grouped by DNS suffix. Leveraging metadata, humans can write policies that say, "Hosts containing 'web' as part of their name can perform SQL queries against hosts containing 'dbase' as part of their name."

The central policy manager software analyzes the policy, the metadata, and the hosts filtered by VNF packet filters. The policy manager then compiles and deploys the correct packet filter rules for each VNF stateful packet filter. Figure 27-6 illustrates.

This approach removes the burden of granular management from the network operator, shifting it to software. Policy management software makes managing dynamic VNFs possible.

Intent-Based Networking

An emerging technique being used to handle complex configuration is intent-based networking. Intent-based networking allows for plain language to be used to express a configuration desire. While traditional configuration describes exactly how the network is to behave, intent-based networking describes the hoped-for outcome, but not how the outcome will be achieved.

Intent-based networking is interesting in the context of VNFs because it introduces a layer of abstraction between network state and configuration specifics. The intent engine interprets the generic intent expressed by a human or possibly software, and then sends the specific configuration (or individuated policies) required to convert the intent into network state. Figure 27-7 illustrates.

Intent is also a useful tool in enforcing a desired network state. If intent is used to describe the desired network state, and the intent engine can interpret those directives into configuration specifics, then the intent engine can also be leveraged to determine when the network state no longer matches the expressed intent.

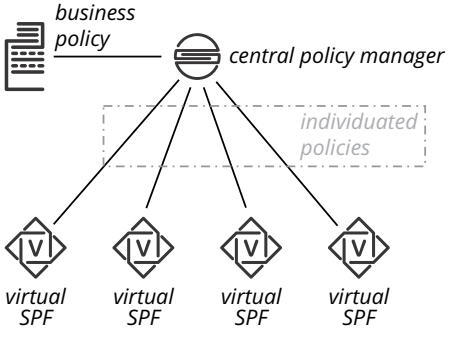


Figure 27-6 *A Centralized Policy Manager*

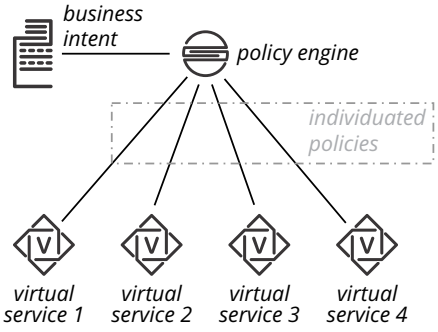


Figure 27-7 *Intent-based Networking*

Intent is still very new, and proving difficult to implement. Wide variations in network hardware and network operating system software introduce many variables that make implementing intent-based networking a complex programming challenge. Nonetheless, specific intent-based networking implementations have found their way into open source projects such as Open Network Operating System (ONOS) and OpenDaylight. In addition, several commercial variants have been introduced to the market, with others expected to find their way to market soon.

Benefit

The chief benefit of VNF automation is a decreased time to service. Bringing applications to market quickly is a critical benefit of VNFs, as they allow for a service to be composed and instantiated with a minimum of risk and without human configuration.

Centralized policy management based on metadata and intent-based networking are examples of tooling that enable VNF spin-up time to be reduced.

Compute Advantages and Architecture

Physical network devices such as switches and routers run on built-for-purpose silicon. Other network devices such as firewalls and load balancers might run on general-purpose CPUs while offloading specific functions to dedicated hardware. Encryption is an example of this, where a load balancer might run most functions on a general-purpose CPU, while mathematically intensive packet encryption is offloaded to dedicated silicon to maintain high throughput levels.

Built-for-purpose silicon chips are called Application-Specific Integrated Circuits (ASICs). ASICs are designed by networking vendors to perform a small number of networking functions and do them quickly. ASICs are limited in function—they are

"application specific." Thus, network devices perform tasks using ASICs to do what they do very well but cannot perform anything beyond what the ASIC was designed for.

In contrast to ASICs, general-purpose processors are found in servers and desktop computers. General-purpose processors are designed to run a wide variety of software. Intel x86 processors are the most widely known general-purpose processor; network functions that have been virtualized are said to be running on x86.

While ASICs do a small number of things extremely well, x86 processors do a large number of things merely adequately. This is critical to understanding VNFs. When a network function is virtualized to run on x86, performance might be reduced when compared to the same function running on an ASIC.

Discussions about VNFs and performance are commonplace due to two major concerns.

1. VNFs need to function quickly enough to fill the network pipe of the host they run on. Ethernet speeds of 10, 25, 40, and even higher are all interesting to network operators leveraging VNFs.
2. VNFs use the general-purpose x86 CPU in hosts to provide their services. These hosts are also going to be running other workloads the data center requires—web server, database engines, and so on. CPU cycles used for VNFs are not available for those other workloads.

Improving VNF Throughput

There are two significant means VNF software architects use to gain sufficient throughput from their network services. The first means is software optimization. The second is hardware offload.

Software optimization is related to the peculiarities of the operating system that many virtualized network functions run in, Linux. Linux processes network functions in the Linux kernel. However, the VNF software is going to be running as a user. When the user space program needs access to the network interface hardware to send or receive packets, it will perform a system call to the kernel. A hardware interrupt is performed, and data is copied between kernel and user space. All of this takes time, reducing the maximum amount of throughput the VNF might otherwise achieve.

Software optimization of VNFs seeks to eliminate the back-and-forth between kernel space and user space. Many networking software stacks run completely in Linux user space. To gain access to the network hardware, these user space stacks leverage an open source project called Data Plane Development Kit (DPDK). DPDK

provides a means for a networking stack to directly access the networking interface inside a host without having to perform system calls to the kernel. This reduces latency, subsequently increasing throughput.

Hardware offload consists of network interface cards (NICs) with customized silicon designed to offload the x86 CPU from some VNF tasks. NICs with customized silicon are costly compared to less-capable NICs, as well as being specialized for specific environments. These custom NICs run with special drivers, handing off functions from specific VNFs to hardware to accelerating them.

Considering Tradeoffs

If you have not found the tradeoffs, you have not looked hard enough.

This is true in every area of engineering (and life!), including NFV. This chapter has largely considered the case for NFV. What are the tradeoffs? The State/Optimization/Surface (SOS) triad will be useful in evaluating these tradeoffs.

State

Rather than putting an appliance, or a cluster of appliances, into the path of packets flowing through the network, NFV brings the packets to the services. These services, in turn, could be scattered throughout the network, including being located anyplace on a data center fabric. If you consider the movement of traffic through the network as an optimization problem, NFV requires more granular information about where services are located in order. In essence, the service becomes the destination, rather than a logical subnet. NFV, then, will require the control plane to carry greater amounts of state.

At the same time, virtualized services are likely to move more often than services in a physical cluster or appliance; it is more difficult to unbolt, unrack, rack, and bolt an appliance than it is to respin a service on a new server. This means services move around more quickly both "because they can" (lowered cost often leads to less discipline) and because, once the network is perceived as "free," the service "wants" to move to the highest-quality, lowest-cost compute resources possible.

There is another aspect of state to consider, as well: the distribution of policy in the network. It is simple enough to say, "smaller chunks of state spread throughout the network are easier to manage than one large configuration or state store." Each individual piece of state is smaller and more tuned to the local need (a return to the principle of subsidiarity). On the other hand, understanding how widely distributed state interacts can be a lot more difficult; the interaction between two pieces of state in a single configuration file can be difficult to understand, and the interaction

between two pieces of state configured on two different devices, widely separated in the network, and with different configuration parameters and/or styles can easily move into the "impossible" territory.

All of the additional state used to configure and manage a larger number of devices must also be carried on the network, which means a different order of magnitude of state must be carried and managed on the network. This not only eats network resources, but it also increases resilience demands on the network.

Optimization

There are several tradeoffs to consider in the realm of optimization. First, NFV often treats the network as a "free resource." Any time you make a resource appear to be cheap or free, you are saying you would prefer to use more of the free resource and less of more expensive resources. If the cost of the network is free, the marginal utility on network resources drops to the point where network resources are not considered when deciding where to run a particular service and why. Spreading services out across the network drives more traffic onto the network, which uses more network resources than by gathering services into clusters.

Network utilization is not just about the amount of bandwidth being carried over the network versus the amount of work being done. There is also the efficiency of troubleshooting, which directly impacts the Mean Time to Repair (MTTR), and therefore the network uptime (or measured resilience).

Surface

Finally, there are interaction surfaces to consider. It often sounds good to automate everything and then turn the automation system over to an intent-based controller to manage the interaction between the applications running on the network, the control plane, and device configuration. Each of these new interactions, however, also represents a new interaction surface, with the implied complexity of abstraction, leaky abstractions, and other issues with interaction surfaces. Each of these interaction surfaces will require an Application Programming Interface (API), which introduces the complexity of managing these APIs over time.

Other Tradeoffs to Consider

There are other tradeoffs to consider, as well, such as whether outsourcing as much complexity as possible to a vendor in the process of moving to an intent-based network is really a "good thing." Internal skill sets are bound to atrophy when complex problems are outsourced, leaving the business without any local resources to call on when problems happen. Moving configuration, policy, and intent to a single device

can mean a single mistake impacts a lot more devices. You are trading the centralized configuration of a clustered appliance for the centralized configuration of an orchestration system. Does this make sense? As with all things, it is important to consider the tradeoffs.

Final Thoughts

NFV and intent-based networking are two attempts to define a simpler network for the future. The question, as always, is: "Simpler in what way—and more complex in what other ways?" NFV, combined with service chaining, the disaggregation of network services out of appliances and into standardized compute resources, the concept of scale-out services, and the movement toward automation and intent are all interesting trends in the network engineering world that will ultimately make contributions to the way networks are designed and operated.

The next chapter will discuss another trend likely to make a large impact on the way networks are designed and operated: the Internet of Things.

Further Reading

Boucadair, Mohamed. "Service Function Chaining (SFC) Control Plane Components & Requirements." Internet-Draft. Internet Engineering Task Force, October 2016. https://datatracker.ietf.org/doc/html/draft-ietf-sfc-control-plane-08.

Dolson, David, Shunsuke Homma, Diego Lopez, Mohamed Boucadair, Dapeng Liu, Ting Ao, and Vu Anh Vu. "Hierarchical Service Function Chaining (hSFC)." Internet-Draft. Internet Engineering Task Force, January 2017. https://datatracker.ietf.org/doc/html/draft-ietf-sfc-hierarchical-02.

Filsfils, Clarence, Stefano Previdi, Bruno Decraene, Stephane Litkowski, and Rob Shakir. "Segment Routing Architecture." Internet-Draft. Internet Engineering Task Force, February 2017. https://datatracker.ietf.org/doc/html/draft-ietf-spring-segment-routing-11.

Halpern, Joel M., and Carlos Pignataro. *Service Function Chaining (SFC) Architecture*. Request for Comments 7665. RFC Editor, 2015. https://rfc-editor.org/rfc/rfc7665.txt.

Hudson, Jon, Lawrence Kreeger, Dr. Thomas Narten, Marc Lasserre, and David L. Black. *An Architecture for Data-Center Network Virtualization over Layer 3 (NVO3)*. Request for Comments 8014. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc8014.txt.

K., Thomas. "User Space Networking Fuels NFV Performance." *Intel Developer Zone*, June 12, 2015. https://software.intel.com/en-us/blogs/2015/06/12/user-space-networking-fuels-nfv-performance.

Kumar, Surendra, Mudassir Tufail, Sumandra Majee, Claudiu Captari, and Shunsuke Homma. "Service Function Chaining Use Cases in Data Centers." Internet-Draft. Internet Engineering Task Force, February 2017. https://datatracker.ietf.org/doc/html/draft-ietf-sfc-dc-use-cases-06.

"L4-L7 Service Function Chaining Solution Architecture." Open Networking Foundation, June 14, 2015. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/L4-L7_Service_Function_Chaining_Solution_Architecture.pdf.

Lasserre, Marc, Florin Balus, Thomas Morin, Dr. Nabil N. Bitar, and Yakov Rekhter. *Framework for Data Center (DC) Network Virtualization*. Request for Comments 7365. RFC Editor, 2014. https://rfc-editor.org/rfc/rfc7365.txt.

Nadeau, Thomas, and Paul Quinn. *Problem Statement for Service Function Chaining*. Request for Comments 7498. RFC Editor, 2015. https://rfc-editor.org/rfc/rfc7498.txt.

Narten, Dr. Thomas, Luyuan Fang, Eric Gray, Lawrence Kreeger, Maria Napierala, and David L. Black. *Problem Statement: Overlays for Network Virtualization*. Request for Comments 7364. RFC Editor, 2014. https://rfc-editor.org/rfc/rfc7364.txt.

"Network Functions Virtualisation (NFV); Continuous Development and Integration; Report on Use Cases and Recommendations for VNF Snapshot." European Telecommunications Standards Institute, March 2017. http://www.etsi.org/deliver/etsi_gr/NFV-TST/001_099/005/03.01.01_60/gr_NFV-TST005v030101p.pdf.

"Network Functions Virtualisation (NFV) Release 3; NFV Evolution and Ecosystem; Hardware Interoperability Requirements Specification." European Telecommunications Standards Institute, March 2017. http://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/007/03.01.02_60/gs_NFV-EVE007v030102p.pdf.

"Network Functions Virtualisation (NFV) Release 3; Security; Security Management and Monitoring Specification." European Telecommunications Standards Institute, February 2017. http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/013/03.01.01_60/gs_NFV-SEC013v030101p.pdf.

"Network Functions Virtualisation (NFV); Reliability; Report on Models and Features for End-to-End Reliability." European Telecommunications Standards Institute, April 2016. http://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/003/01.01.01_60/gs_NFV-REL003v010101p.pdf.

Quinn, Paul, and Uri Elzur. "Network Service Header." Internet-Draft. Internet Engineering Task Force, February 2017. https://datatracker.ietf.org/doc/html/draft-ietf-sfc-nsh-12.

Quinn, Paul, and Jim Guichard. "Service Function Chaining: Creating a Service Plane Using Network Service Header (NSH)." IEEE. Accessed April 21, 2017. https://www.opennetworking.org/images/stories/downloads/sdn-resources/IEEE-papers/service-function-chaining.pdf.

Yizhou, Li, Lucy Yong, Lawrence Kreeger, Dr. Thomas Narten, and David L. Black. "Split-NVE Control Plane Requirements." Internet-Draft. Internet Engineering Task Force, February 2017. https://datatracker.ietf.org/doc/html/draft-ietf-nvo3-hpvr2nve-cp-req-06.

Yong, Lucy, Aldrin Isaac, Linda Dunbar, Mehmet Toy, and Vishwas Manral. "Use Cases for Data Center Network Virtualization Overlay Networks." Internet-Draft. Internet Engineering Task Force, February 2017. https://datatracker.ietf.org/doc/html/draft-ietf-nvo3-use-case-17.

Review Questions

1. In what ways are humans ill-suited to performing configuration management?
2. Explain the constraints placed upon a network infrastructure by physical network functions.
3. Why is service function chaining necessary?
4. Explain the purpose of a network service header.
5. How do VNFs help reduce the "blast radius" of a network outage?
6. What is the purpose of metadata in centralized policy management?
7. How is intent-based networking distinct from traditional configuration?
8. What is the chief benefit of VNF automation?
9. What is the most significant impact of moving a network function from an ASIC to a general-purpose CPU?
10. In what way does the Linux kernel impose a bottleneck to VNF performance?

This page intentionally left blank

Chapter 28

Cloud Computing Concepts and Challenges

Learning Objectives

After reading this chapter, you should understand:

- Some common cloud service offering models, such as software and platform as a service
- Some common reasons businesses choose to move their processing to public cloud services
- Some common tradeoffs to moving processing to public cloud services
- Some common technical challenges involved in moving processing to cloud services
- The concept of data gravity
- Common elements of cloud security

While the term *cloud* has come to mean many different things, it will be defined here as *virtualized products or infrastructure consumed via self-service*. In the cloud model, the consumer uses a portal or Application Programming Interface (API) to requisition a service, platform, or server, specifying requirements during the request. The request is fulfilled by software automatically, so the consumer can use the requested service immediately. Common examples of cloud services include

- **Software as a Service (SaaS).** SaaS is application software for which the consumer must neither provide hardware for nor buy and install shrinkwrapped software. Rather, the service is leased via a subscription and consumed via the Internet through a web browser, mail client, or other custom software supplied by the SaaS provider. SalesForce.com, Microsoft's Office365, and Google Apps for Business are all examples of SaaS.
- **Platform as a Service (PaaS).** PaaS offerings are software building blocks used in the creation of a full software package. PaaS offers a blank canvas software developers use for their own projects as opposed to the complete software products offered as SaaS and aimed at end users. PaaS building blocks vary by vendor but include features such as programming languages, development testing environments, databases, security, load-balancing, workload orchestration, and data analytics.
- **Serverless or Functions as a Service (FaaS).** FaaS are on-demand software routines hosted in a cloud environment that, upon receiving a data input, perform processing and return an output. In fact, FaaS runs on servers just like any other computing function but is not bogged down with maintaining a heavy software environment. FaaS was first popularized using the term *serverless*, and serverless is still seen in most technical literature describing the service.
- **Infrastructure as a Service (IaaS).** IaaS is virtualized servers, storage, networking, and security. IaaS consumers request a virtual machine with a specific number of virtual CPUs, RAM, storage, etc. The consumer can also request an operating system to be installed, or even a service such as switching, routing, load balancing, etc. From there, the IaaS compute is available as a virtual machine to run any software the consumer can install on it. IaaS offers a consumer maximum flexibility of software without running physical hardware. IaaS providers could be thought of as virtual data centers.

The lines separating SaaS, PaaS, serverless, and IaaS are not well defined; some cloud products could conceivably be lumped into more than one "as-a-service" category, and the market changes rapidly. Because of this, it is not useful to become overly fixated on the distinctions, especially for this chapter, which will primarily focus on the technical problems faced by network engineers when working with cloud-based services.

Computing clouds are most often thought of as public clouds. The physical infrastructure making up the cloud is not owned by the organization consuming it. Rather, public clouds are created and maintained by third parties. The most well-known

public clouds are Amazon Web Services (AWS), Microsoft Azure (Azure), and Google Cloud Platform (GCP).

However, there is no technical reason an organization cannot build a computing cloud system hosted on its own physical infrastructure. A cloud of this type is called a private cloud. Private clouds are built by organizations to move beyond traditional computing infrastructure, but constrained by cost, security, or privacy concerns ruling out public cloud adoption. OpenStack, CloudStack, and other orchestration engines can be used to create private clouds.

Most cloud-consuming organizations are neither exclusively public nor private cloud users. Instead, they are hybrid cloud users and operators. Some of their compute workloads are in the private cloud, while others are in the public cloud. Typically, there is some sort of physical and virtual network interconnecting the different domains making up the hybrid cloud environment.

Most hybrid cloud organizations consuming public cloud also use more than one public cloud. For example, an organization might use both AWS and Azure. These organizations are said to be multicloud.

One critical consideration for network engineers studying cloud computing is Application Programming Interfaces (APIs). The configuration of network equipment traditionally involves using a command-line interface (CLI) to create configurations describing how a device is supposed to behave. Commands are entered via the CLI, and responses to those commands are posted by the network operating system. The CLI is intended primarily as a human-friendly interface. The output in particular is geared toward a human being reading the information on a screen and making sense of it.

In contrast with the CLI, APIs are intended for programs. APIs accept specific input and provide structured output. A program using APIs to configure a device will provide a specific input, perhaps about an interface or routing protocol, and make a call to an appropriate API class and method using the input. The API will accept the input and act on it appropriately, configuring the device. The result of the operation is returned to the calling program as structured data. The structured data conforms to a predictable format and can be stored by the program for later reference.

APIs are used for configuration as well as status inquiry. For instance, the appropriate API calls can return structured output describing the state of all Border Gateway Protocol (BGP) neighbors, interface counters, and so on.

Cloud resources are often consumed via APIs. For businesses automating the lifecycle of their applications, API consumption is a given, as programs are behind the automation process. For network engineers, this means familiarity with programming, APIs, and automation techniques; and integration of networking services into a larger provisioning task becomes a useful, and even critical, skill.

Note

For a deeper understanding of APIs and automation, see Chapter 26, "The Case for Network Automation."

Public Cloud Business Drivers

When a business moves from using internal resources to building either a cloud or a more traditional information technology infrastructure, it is outsourcing infrastructure and operations. Why would a business outsource its infrastructure and operations to another company? The reasons are not much different from many other decisions to outsource; they are financial and operational.

Shifting from Capital to Operational Expenditure

There are two basic kinds of costs in business:

- Capital expenses (CAPEX) are what a company buys in order to operate the services it sells. This includes desks, buildings, and information technology, such as routers and switches.
- Operational expenses (OPEX) are what a company pays for on an as-needed basis, such as people, services, and consumable goods.

There is some amount of tradeoff between these two; sometimes you can buy equipment that will reduce OPEX. Other times, of course, equipment purchases add to OPEX. Purchasing cloud services moves the cost for processing power from being a mix of CAPEX and OPEX to being entirely OPEX; by outsourcing, the business does not need to worry about buying any sort of network or server gear other than to support connectivity within campuses or to the cloud service. Moving from CAPEX to OPEX is helpful to business operations because it results in smoother, more predictable cash flow.

Businesses often assume, as well, that large-scale cloud providers, because they have access to bulk buying and deeply staffed design and hardware teams, can build and manage computing resources more cheaply than a smaller company can.

Specifically, large-scale network operators can take advantage of white box hardware and open source software to provide services at a lower cost than a company not specializing in providing computing resources. This may, or may not, be true in any particular situation, depending on the maturity of a given ecosystem, the actual requirements placed on the network, and the willingness of the business to look outside the box for solutions.

The amount of OPEX varies with how much of the cloud operator's resources are consumed, measured in a variety of metrics including CPU cycles and network bits transferred. OPEX might further vary due to changes in staffing or consulting requirements. It is possible that outsourcing infrastructure building to public cloud operators reduces the need for certain kinds of in-house expertise. This will be more true for SaaS offerings than PaaS or IaaS.

Some businesses might find public cloud consumption is more expensive than operating their own private infrastructure. In fact, some organizations have shifted workloads back from public cloud to private infrastructure to lower OPEX costs. Serverless is one response by public cloud operators in response to this problem. Applications leveraging FaaS instead of sitting on full-blown virtual machines or long-lived containers see as much as an 80% cost reduction.

Time-to-Market and Business Agility

Public cloud gives businesses computing infrastructure with the swipe of credit card, reducing procurement times from weeks or months to minutes. This sort of immediate access to infrastructure and information technology can enable a business to bring products and solutions to market very quickly. Another use for this immediate access to information technology is the ability to shift load during a peak in the business cycle, so the business does not lose opportunity because of an inability to scale. The business versus infrastructure spending chart originally used in Chapter 1, "Fundamental Concepts," provides a good example of where cloud computing can be useful for business agility; Figure 28-1 illustrates.

In order to avoid the times marked in dark gray as *lost business opportunity,* many businesses will consistently overbuild their infrastructure. Cyclical businesses are in the worse position of trying to cope with consumer behavior while also managing this kind of growth curve. Rather than buying enough network and compute resources to stay consistently above demand, businesses can use public cloud computing platforms as a resource on which to temporarily burst (such as a retail operation in the few weeks before the winter holidays), leveling out their information processing purchases over time.

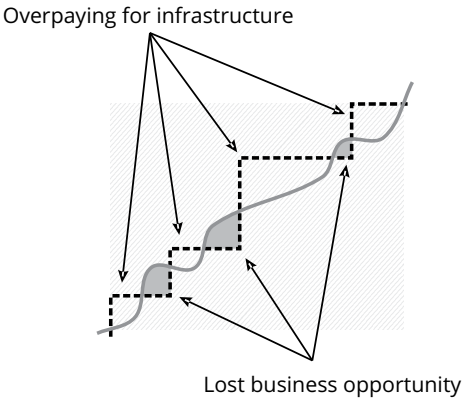


Figure 28-1 *Business Agility and Cloud Computing*

Nontechnical Public Cloud Tradeoffs

While there are advantages to using public cloud services, covered in the previous section, there are also tradeoffs—those little things the account team selling the services is either not going to tell you about or is going to minimize as challenges. It is important, however, to really consider these tradeoffs when making the decision to move processing to a public cloud. The benefit of public cloud is neither obvious nor a foregone conclusion, as it varies with the business situation in question. This section will consider some of the various tradeoffs that businesses and engineers need to consider when considering moving their processing to public cloud services.

Operational Tradeoffs

A common reason for moving to cloud is to reduce internal operational costs by reducing the number of network and infrastructure engineering resources required to build and deploy applications. Because infrastructure resources can be consumed via APIs in a public cloud environment, there is no longer any need for an operations team. Developers building an application can use their programming skills to build the application and to provision the infrastructure the application runs on. This might appeal to businesses trying to conserve operational expenses. So not only are all the costs of providing computing shifted from OPEX to CAPEX, the total amount of OPEX is at least held steady, if not reduced, as well. If application developers can do the job of infrastructure engineers, this seems to present an attractive cost savings. However, this "NoOps" view of infrastructure is short-sighted for several reasons as described in the sections that follow.

Moving to Cloud Computing Does Not Remove the Need for Infrastructure Design

Infrastructure engineers are competent in infrastructure provisioning, to be sure. However, provisioning infrastructure in a way that meets business objectives is complex. Business requirements drive specific infrastructure provisioning decisions.

For example, a business might have specific service level agreements (SLAs) it has made with its customers. To meet those SLAs, the organization will have matching resiliency requirements for their application. The application might need to be available despite a catastrophic failure in a specific region. The application might have to respond to user requests in a specific amount of time.

These sorts of requirements require a keen understanding of infrastructure design. To maintain availability, any number of infrastructure decisions must be made:

- The network engineer might need to apply specific security policies to a traffic flow.
- Network capacity must be sized to meet demand.
- Connections between multiple clouds might be required, depending on what resources an application calls on and where those calls are going to and from.
- Data replication to a disaster recovery site will require connectivity and capacity, as well as routing and possibly address translation services to ensure application availability during a primary site outage.

In short, light does not come from a light switch. While flipping a light switch turns on the lights, being able to operate a light switch implies no knowledge of electrical supply, electrical circuits, circuit breakers, ground wires, or even light bulbs. And yet, all those components are critical to the functioning of the light.

Experts Are Still Required When Infrastructure Fails

While simple failures can be overcome by throwing out a broken piece of equipment or software, and replacing it with a new one, many infrastructure failures are not simple. Infrastructure, especially infrastructure designed for resiliency, tends to be complex. The more complex something is, the greater the likelihood is that something can go wrong, and the more nuanced the resultant problem might be.

Troubleshooting complex problems requires deep expertise. In the cloud computing era, there is still a need for infrastructure engineers with deep expertise in managing and troubleshooting complex large-scale networks. Infrastructure engineers know how to bring the lights back on when flipping the switch no longer works.

The Cost of Reaching the Cloud Still Needs to Be Considered

Many businesses just "assume" that once they have moved their processing to the cloud, they will be able to reach the processing resources "over the Internet." The reality is moving a lot of data can still cost a lot of money. Circuits still need to be purchased and maintained, Quality of Service must be configured, local resources to "land" the data must be configured and maintained, etc. The costs of these circuits will most likely increase through any kind of cloud migration, and should be an area of particular concern in the case of hybrid- or multicloud deployments. Jitter and latency are also components of cost in operations; these are a real concern because the provider's physical infrastructure may not align with your business operations.

The Costs of Cloud Computing Can Be Increased Because Specialized Hardware Is No Longer Accessible

While cloud computing can often provide generic processing resources at a much lower cost than buying, installing, and maintaining local compute resources, the theory of cloud is grounded in treating every resource and every problem in as much the same way as possible. For instance, you might initially assume a single network device, such as a data center fabric router, can be replaced with a single processor in the cloud. In this case, 20 or 30 processors in the cloud might need to be used to replace this single device, driving the cost of the cloud deployment considerably higher than expected. Specialized hardware drives purchasing and maintenance costs higher, but it drives down the cost of actually processing data; public clouds often simply replace specialized hardware with a large number of more generic resources, shifting costs in unexpected ways.

Feature Creep Can Cause Failure Nightmares

There is a common perception in network engineering that unused features are silent and neutral to the operation of the network—so long as a feature is not configured, it is doing no harm. The reality, however, is far different. Each feature available in a network device, or a cloud-based service, represents some amount of code—code that must interact with the code providing other configured, in-use features. These features, and the code they represent, are perfect gateways into failures through unintended consequences, potential security holes in waiting, and a larger attack surface. This problem is not unique to cloud services, of course; every vendor will add a constant stream of features, most of which any particular customer will not use. This does not mean these features have no effect on the performance or stability of the product you are using, however.

Operational tradeoffs are not the only area to consider when trying to understand the full cost of moving to public clouds for processing; there are also business tradeoffs to study.

Business Tradeoffs

Taking full advantage of cloud computing requires a business to rethink its operational and business processes. First, applications that businesses have built on traditional infrastructure solutions may require significant redesign to maximize their efficiency in a cloud computing environment. Second, operational processes that businesses have built around traditional computing infrastructure will need to be updated to support cloud computing.

Shifts in operational processes and application architectures are significant changes that some businesses have avoided due to their inherent costs. These businesses have tried to replicate as closely as possible their traditional infrastructure architecture and operations, merely replacing their own hardware with a reflection cast in the public cloud mirror. This approach is analogous to "fitting a square peg into a round hole." It is possible to take this approach when the square peg is motivated to fit into the round hole with a sufficiently sized hammer, but the result is inelegant and inefficient.

This points to a larger problem many businesses do not consider when outsourcing: the mismatch between the goals of the outsourcer and the goals of the business itself. The goal of the outsourcing business is to produce a product or service that consumers want to purchase; the goal of the outsourcer is to produce a product that the business will consume as much as possible of, at the highest possible margin. It is quite possible for the outsourcer to drive internal business decisions in a direction that is not good for the outsourcing business in order to increase the outsourcer's revenue and margins.

For instance, it is common for cloud providers (and all other vendors—public cloud providers are not unique in this regard) to add new features and functions they can use to leverage their customers into paying more, whether it actually improves their customer's business or not, and locks the customer in to the cloud provider's product line.

The vendor lock-in problem is particularly acute in most business environments. When a business commits to using a specific cloud vendor, that business's operational processes become locked into how a specific vendor delivers its technology. Moving to a different vendor becomes hard, because the target vendor probably delivers its technology differently, even if the technology in question is essentially the same service.

From a networking perspective, cloud computing presents nothing new in the context of vendor lock-in. For decades, networking vendors have delivered products with limited differentiation in functionality, but via widely different consumption models. Sometimes the underlying technology is different while delivering the same result. Other times, the technology is identical, based on industry standards, but configured in unique ways. And yet other times, vendors offer truly differentiated services unavailable from anyone else.

The networking services available in cloud computing don't break the established paradigm. All vendors offer some baseline of services, but these services can be consumed uniquely. Some might offer special features to set their product apart. The challenge for network engineers is no different than it ever has been, requiring careful comprehension of the technology's capabilities and applicability to a business' problems.

The risk of the all-consuming cloud provider: Some cloud providers have, in the past, used a partnership with a customer to learn how to build and support a particular business model, and then used the experience to enter the market as a direct competitor to their own customer. Providing services for unique businesses can be a great incubation strategy for cloud providers to spin up internal analogs to the customers they are supporting, eventually broadening their market reach.

Technical Challenges of Cloud Networking

For the network engineer, cloud computing presents the challenge of providing low-latency, secure connectivity over a mix of public and private transports using a mix of physical and virtual equipment. In addition, this marvelous cloud-based transport service must also be provisioned and deprovisioned on demand in real time as workloads are stood up and torn down, consumed programmatically, and monitored centrally.

Latency

When you are considering how applications are deployed in cloud environments, workload placement becomes especially interesting. Assume an enterprise is deploying an application in a multicloud environment. In this scenario, workloads can be placed in one or more public clouds, as well as in a private cloud.

Developers often break a single application up into microservices, where each component of the application is separated out into a standalone service. The application is then reconstituted as a set of services communicating with one another across the network to support the same overall set of services as the original application.

The problem that microservices architectures face is latency. When communicating over distances measured in kilometers rather than meters, the time it takes packets to traverse the distance is measured in milliseconds instead of submilliseconds; it takes two such trips across the network, the Round Trip Time (RTT) to complete any transaction between the microservices making up an application. Since multiple microservices must interact to produce the same amount of data as the original, monolithic, application, these delays "stack up" to produce a total delay much greater than many developers expect. Figure 28-2 illustrates.

In Figure 28-2, A requests some information from the monolithic application; the RTT across the network for processing the request and returning the information is 20ms. When B requests this same information, service 1 must request information from service 2, which must request information from service 3, etc. The total network time in the microservices case is 80ms. If there is any increase in the delay across the network for any reason, the effect is multiplied by four in the microservices case, because there are four RTTs involved in every service request.

In applications previously deployed on traditional infrastructure or fully contained in a localized private cloud, latency is far less of a concern. However, applications composed of many components, such as microservices, and spread over a variety of clouds can experience reduced performance due to latency.

For network engineers faced with this problem, at least two solutions present themselves.

Work with application deployment teams to optimize workload placement. Workloads are placed in specific clouds for a variety of reasons, including capacity, cost, and functionality. For network engineers, the key is to be involved in the application design so decisions about workload placement include a clear understanding of the infrastructure implications of those placement choices. Application developers in conjunction with infrastructure engineers and business stakeholders should make design decisions jointly.

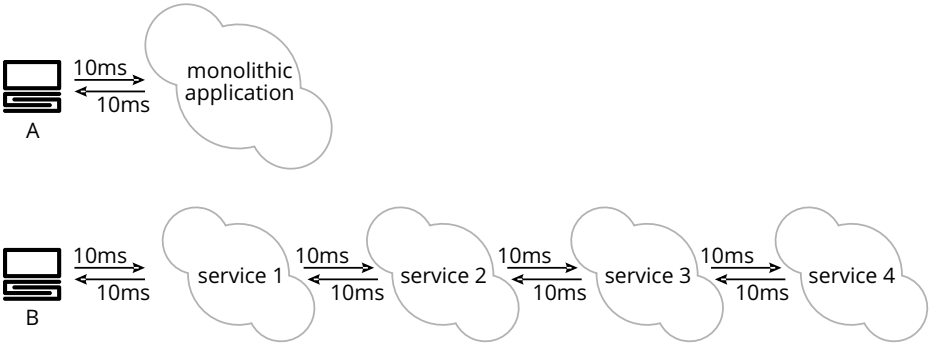


Figure 28-2 *Stacked Delay in a Microservices Architecture*

Every design is a compromise between technical idealism and practical pragmatism. For example, network latency might be an acceptable compromise for a particular design, because the overall application performance is not impacted materially enough. On the other hand, complex applications deployed in ignorance of infrastructure realities might suffer unacceptable performance compromises.

Bring clouds closer together. Many data centers offer cloud exchange services, where customers can purchase direct links to public cloud providers, often through a cloud exchange. This means a network engineer can minimize the impact of latency by designing the network to bring clouds closer together.

These services come at a cost and require a purposeful routing design. A common challenge in standing up direct connections to public clouds is that the IP address blocks in question are accessible both via the public Internet and now via the newly introduced cloud exchange circuit. Routing tables must be populated so traffic is forwarded via the cloud exchange, while also avoiding asymmetric routing.

Populating Remote Storage

When you are moving an existing application to public cloud IaaS, a second problem comes in the form of storage. How is the application data living in a local data center moved into the public cloud so the application has access to the data in the new environment?

For network engineers, this type of challenge is not a new one. Moving large amounts of data from one point to another separated by distance is a problem of constraints. First, the amount of bandwidth between two geographically diverse points is typically limited to a fraction of the bandwidth available in a data center. Second, latency can make it difficult to use the entirety of the bandwidth available to execute the transfer.

In a local area network (LAN), circuits are very high bandwidth, commonly interconnecting hosts to the network at speeds of 10, 25, 40, 50, and even 100Gbps. In the LAN scenario, bandwidth generally is not a constraint when moving storage data around the network. Bottlenecks in the transfer process are more likely to be found in the disk or host bus subsystems.

However, when the storage transfer is happening over a wide area network (WAN) such as the public Internet, bandwidth often becomes a constraint, as the bottleneck moves from host data bus or disk itself back to the network. Circuits interconnecting private and public clouds are very often less than 10Gbps. In addition, the connection might be lossy when compared to a LAN, requiring retransmissions and reducing overall throughput. This is one element network engineers must consider when computing how long it will take to move a storage volume from the local data center to the public cloud.

In addition to bandwidth constraints, latency has historically been a potential constraint. Assuming the Transmission Control Protocol (TCP) is the transfer mechanism, the amount of time waiting for an acknowledgment across the WAN means it might be difficult to fill the available bandwidth. This is a well-known issue for high-bandwidth, high-delay networks—so-called long fat networks (LFNs).

However, the challenge of fully utilizing the available bandwidth of LFNs has been addressed with several tuning techniques and variants to the TCP protocol. For instance, BIC-TCP, TCP Westwood, TCP Reno (with several variants), TCP Hybla, and TCP Vegas are all algorithmic variants of the core TCP congestion control algorithm, modifying window size in relation to round trip time to maximize throughput. Also notable, CUBIC TCP has seen recent attention in the IETF.

The point to keep in mind is populating a remote storage volume with terabytes of data via a copy operation across the public Internet will take more time than a comparable copy performed locally. This introduces a decision point. Is the performance sufficient enough so the copy can be done via network transfer? Or should data be copied onto a local, portable media and then shipped to the remote public cloud?

In a situation like this, there is no magic available to make terabytes of data in one place appear in another instantly. As such, this problem is a good example of understanding the practical limitations of the available technology and working with the business to determine the proper course of action.

Data Gravity

Once data has been populated in the remote cloud storage, moving data back out of the cloud presents challenges. One issue is a practical one: cost. While public cloud providers are keenly interested in their customers checking data in, they don't want those customers to leave. Thus, public cloud providers charge as much as three to five times the ingestion transfer costs to move data back out. This is commonly known as the data gravity problem.

Data gravity is not a networking concern, but rather a business problem that network engineers should be aware of. For network engineers focused on the technology challenge, moving large amounts of storage data out of a public cloud presents the same challenges as moving the data into the cloud in the first place. Limited bandwidth and latency introduce constraints that might increase transfer times unacceptable to the business.

Selecting Among Multiple Paths to the Public Cloud

While some organizations will connect to public cloud services using cloud exchanges, most organizations will connect to the public cloud via the Internet. Internet circuit costs have come down in price, making multiple Internet connections

at the network edge affordable. This offers network engineers an interesting network design option. Rather than a single Internet connection at the edge, multiple connections offer both resiliency and additional bandwidth.

The challenge is how, exactly, to leverage multiple Internet edge circuits? The straightforward and obvious answer is via a routing protocol. In the case of the Internet edge, the routing protocol is BGP. However, while BGP enables the use of multiple Internet connections, BGP's best path algorithm is focused on connectivity and not quality of application experience. BGP can only distinguish the relative closeness of one path versus another, and not whether a longer path might be better quality.

Since BGP is insufficiently nuanced to make optimal routing decisions at a per-application level, a market niche known as Software-Defined WAN (SD-WAN) has taken recent hold in the industry. SD-WAN solutions are typically proprietary forwarding schemes concocted by vendors. SD-WAN forwarding schemes prioritize quality of experience (QoE) for specific applications, and make forwarding decisions based on the QoE policy defined by a network engineer.

In the case of accessing the public cloud, an SD-WAN forwarding scheme will determine the best Internet circuit to use to provide the best service to the cloud consumer. For example, an SD-WAN forwarder might (allegedly) determine Internet circuit A is best to access the Microsoft Office 365 SaaS cloud, while Internet circuit B is best for Amazon Web Services IaaS hosted workloads.

Although unique to the many SD-WAN vendors offering products in this space, making a forwarding decision about what is best might include the following decision points:

1. **Circuit lossiness.** Is a circuit dropping packets? If so, to what degree? Loss will be more acceptable to some traffic, such as large file transfers, where recovery ensures data integrity. Loss will be unacceptable to traffic such as real-time voice, where a conversation will be impacted.
2. **Circuit jitter.** Is a circuit delivering packets on predictable time intervals? Like loss, jitter—a variance in the time delta between packet deliveries—is acceptable or not, depending on the packet payload.
3. **Circuit load.** How busy is a given circuit? SD-WAN solutions can choose to send traffic over a less loaded circuit to improve QoE for the traffic.

SD-WAN products take the routing design and administration out of the hands of the network engineer or the routing protocol, moving those concerns to software. For connectivity to public cloud, this means the end user QoE is optimized constantly, without the network engineer having to make unusual tweaks to the routing system. This approach has the added benefit of being able to add and remove Internet edge circuits to the scheme at will with a minimum of engineering.

The downside of SD-WAN solutions is they are proprietary. While there have been some very early conversations in the networking industry about making SD-WAN solutions interoperable, the market is too nascent and unstable to have seen progress in SD-WAN standardization. The market is focused instead on product consolidation and customer growth.

Security in the Cloud

With security breaches a regular part of the news cycle, the conversation of properly securing the public cloud becomes poignantly interesting. For network engineers, there are several concerns worth discussing:

1. Protecting data over public transport
2. Managing secure connections between cloud environments
3. Isolating data in multitenant environments
4. Understanding role-based access controls (RBAC) in cloud environments

Protecting Data over Public Transport

In a LAN, whether data should be encrypted or not is an open question. When data is being moved between two trusted endpoints across a wholly owned LAN, is there any security advantage in encrypting the data? The answer depends greatly on several factors:

1. **The nature of the data.** For example, health data and credit card data contain highly sensitive information. The data should be encrypted in all circumstances, but also might have to be encrypted for regulatory reasons.
2. **How trust is defined in an organization.** The idea of a hardened network perimeter where a trusted network resides on one side and an untrusted on the other is largely historical. While there is an inherent feeling of trust or comfort borne of familiarity, network hosts are not trustworthy just because they are part of infrastructure owned by an organization. In the modern era, malware infections are assumed, meaning all hosts on a network need to be looked at as threats. In the context of network transport, this means any host on a network should be viewed as a possible point of gathering packets. Assuming the host can see every packet on the wire, what can be done to prevent the packet's payload from being interesting to the malware-infected host?

3. **Whether the data is already encrypted or not.** In an application stack, the data could be encrypted in several ways. One of those ways is at an application level, where the client and server negotiate an encryption scheme to be used to obfuscate the data payload. For instance, Secure Hypertext Transfer Protocol (HTTPS) is HTTP over Transport Layer Security (TLS). In the presence of HTTPS, does it make sense to encrypt the traffic again with the lower-level Internet Protocol Security (IPsec) protocol of use to network engineers securing point-to-point links?

When considering the public cloud, these questions are all relevant but have a different context. For instance, most connections to public cloud services are over the public Internet. The public Internet is normally considered an untrusted transport.

While encryption might not be required, it is a best common practice to always encrypt data traveling over an untrusted transport. The encryption might be via HTTPS, which is not a concern for network engineers, as it is happening at the application level. For network engineers, the primary encryption concern will be for connecting cloud environments together.

IPsec is the most common technology used to interconnect cloud environments. IPsec offers the benefit of a tunnel mode as well as strong encryption. This means network engineers can connect an AWS Virtual Private Cloud (VPC) to a local data center across the Internet. The AWS VPC network can be treated as a network like any other network connected to the organization, using the IPsec tunnel as a WAN link.

IPsec tunnels can also be used to connect not only private and public cloud environments together, but also public clouds to public clouds. This means a workload in one public cloud could query a workload in a different public cloud with an encrypted payload via the public Internet.

Note *encryption* and *security* are not synonymous. While encryption is one part of a security infrastructure, encryption by itself does not imply a secure network or application. Additional security elements that might be required for an application to be considered secure include authentication, input sanitization, access control lists, a backup and recovery scheme, and deep packet inspection.

Managing Secure Connections

A significant challenge of IPsec is managing the connections. IPsec configuration is complex, requiring deep engineering knowledge. Maintaining the Virtual Private Network (VPN) once the IPsec tunnels have been created is an ongoing task to ensure required tunnels stay up, old tunnels are torn down when they are no longer needed, and new tunnels are built when appropriate.

IPsec endpoints are also notoriously difficult to connect if the vendors vary. IPsec is a standard, but there is enough flexibility in the standard to make the creation and maintenance of inter-vendor IPsec tunnels a frustrating experience.

In public cloud networking, IPsec tunnels are relied upon to interconnect environments, but the variety of ways in which this can be done is fraught with management headaches. To ease this burden, a market has opened for vendors to manage IPsec tunnels via a centralized management tool. In this scenario, the tool is aware of the multiple clouds an organization is using. The network engineer uses the tool to select different clouds to be interconnected. The tool takes care of the IPsec details, creating and maintaining the tunnel between environments.

The Multitenant Cloud

Another concern some raise about public cloud is that public clouds are multitenant environments. The compute infrastructure, including data, of one organization is hosted in a public cloud right alongside the compute infrastructure of another. How are these compute environments separated or compartmentalized? Is there a chance some tenant could gain access to another tenant's data because they are sharing public cloud infrastructure?

The short answer to this concern is the risk is not generally considered significant. Multitenancy is well understood in computing and networking. Virtualization is the critical technology employed to allow multiple tenants to share common hardware resources.

In addition, public cloud providers often demonstrate compliance with critical security standards, allowing their infrastructure to be used for sensitive transactions. For instance, both AWS and Microsoft Azure are PCI-DSS Level 1 Service Providers, of interest to those processing payments. PCI-DSS is just the tip of the cloud compliance iceberg. Both Azure and AWS offer certifications for several compliance-related programs the world over, as well as support organizations aiding customers impacted by these regulations.

This is a roundabout way to make the point that multitenancy is not a concern for organizations wishing to consume the public cloud. Security offerings in the cloud are robust and nuanced, moving beyond simple tenant isolation and into compliance with complex regulations.

Role-Based Access Controls

Public clouds also offer complex controls to limit what entities can access which resources in the public cloud. In networking, this is known as role-based access controls (RBAC).

In networking, RBAC has been used to control what administrative tasks network engineers can perform on network equipment. In the public cloud, resources can be similarly controlled. For example, in AWS, the Identity and Access Management (IAM) service offers granular roles and permissions for a variety of public cloud resources. In addition, extensive documentation and training are available to properly leverage this complex resource.

Monitoring Cloud Networks

Another challenge facing network engineers in the public cloud is packet capture and analysis. In wholly owned networks, access to the physical switches and wires carrying traffic means traffic can be copied from one port to another for capture, or intercepted via network taps. These copied packets flow across a visibility fabric—a collection of specialized network devices that gather, filter, and slice packets—to tools that perform packet analysis.

Networking in the public cloud presents a challenge for visibility fabrics, because there is no longer access to physical switches or wires from which to obtain copies of traffic. How can packets be captured when there is no physical network accessible?

This unique challenge is being handled by vendors via host interception. While the underlying network infrastructure of the public cloud is not accessible, the hosts running on the public cloud are. Those hosts are the virtualized workloads that public cloud consumers own and operate. Therefore, to capture traffic in the public cloud, copies of the packets are made on the virtual workload and tunneled to a tool that will perform the analysis.

The virtual workload runs an agent that facilitates the copy. The agent will also perform filtering, so not all packets are copied to the analysis tools. Copying all packets everywhere to analysis tools could overwhelm the network with excessive traffic, a pointless thing to do if just specific packets are required.

Final Thoughts

Cloud computing, for all the infrastructure complexity it masks, does not eliminate a requirement for thoughtful network design. Businesses sold on the notion that the difficulties of operating infrastructure go away because they have paid a friendly cloud provider are missing a crucial point. Best leveraging of cloud technologies means a shift in skillsets, and not an elimination of expertise.

Cloud computing can even introduce new problems in application performance if an appropriate design is overlooked. Network engineers who wish to add value to

the organizations they support will benefit their organizations by offering designs to make the best of high-latency network links.

In addition, cloud computing necessitates all technology silos in an IT team working together. Network engineers have an opportunity to lead, as the transport between cloud environments is a point of commonality touching API calls between services, storage performance, high availability, and disaster recovery. A deep understanding of how the network enables or constrains communications informs the design of all these services.

Further Reading

Erl, Thomas, Ricardo Puttini, and Zaigham Mahmood. *Cloud Computing: Concepts, Technology & Architecture*. 1st edition. Upper Saddle River, NJ: Prentice Hall, 2013.

Hawramani, Ikram. *Cloud Computing for Complete Beginners: Building and Scaling High-Performance Web Servers on the Amazon Cloud*. 1st edition. Hawramani.com, 2016.

"PCI Compliance—Amazon Web Services (AWS)." Amazon Web Services, Inc. Accessed August 25, 2017. https://aws.amazon.com/compliance/pci-dss-level-1-faqs/.

Reed, Archie, and Stephen G. Bennett. *Silver Clouds, Dark Linings: A Concise Guide to Cloud Computing*. 1st edition. Prentice Hall, 2010.

Rhee, Injong, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. "CUBIC for Fast Long-Distance Networks." Internet-Draft. Internet Engineering Task Force, July 2017. https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-cubic-05.

Ruparelia, Nayan B. *Cloud Computing*. Cambridge, MA: The MIT Press, 2016.

Weinberger, Matt. "Amazon Explains Its Secret Weapon in the Cloud Wars." *Business Insider*. Accessed August 25, 2017. http://www.businessinsider.com/amazon-web-services-lambda-explained-2015-11.

Review Questions

1. This chapter states that moving from internally owned and managed resources to a public cloud service can move CAPEX to OPEX, and make costs more predictable. What does the predictability of cost rely on in a cloud service?

2. This chapter states that feature creep in a cloud service can cause nightmares. Compare the use of proprietary features in vendor-provided network equipment to the use of proprietary features in public cloud services. How are they different or the same?
3. Explain why latency and jitter would be issues to consider when moving processing to a public cloud service.
4. Research the concept of data gravity. What are other meanings for this term, and the problems it represents, which are not covered in the text?
5. Why is selecting the best route into and out of cloud services important?
6. There are many cloud security issues not considered in the chapter, such as cross processor memory attacks, data breaches, and providing confidentiality against the cloud provider. Choose one of these problems, describe the problem, and describe at least one solution to the problem (if there is one available).

Chapter 29

Internet of Things

Learning Objectives

After reading this chapter, you should understand:

- The relationship between IoT and DDoS attacks
- The problems involved in securing IoT devices
- The concept of unikernels, and how they relate to IoT
- The basic kinds of IoT connectivity available, including BlueTooth and LoRaWAN

The world is made up of things. Television sets, radios, light bulbs, and refrigerators all surround each of us every day, providing essential services in some cases, and simply making our lives simpler in others. The *Internet of Things* (IoT) either accepts the reality or proposes to make real (depending on your perspective) the connection of every one of these devices to the Internet. Connecting this many "things" to the Internet, however, requires a radical rethinking of the systems required to collect and act on data; the sheer amount of data would require the kinds of newer design patterns outlined in Chapter 25, "Disaggregation, Hyperconvergence, and the Changing Network," and rely on the kinds of service separation and scaling considered in Chapter 27, "Virtualized Network Functions."

This chapter considers IoT from various perspectives.

Introducing IoT

On Tuesday, September 13, 2016, the security analysis website KrebsOnSecurity.com was down.[1] More accurately, the site was rendered inaccessible by a distributed denial of service (DDoS) attack. DDoS attacks take advantage of the free and open nature of the Internet. Since any part of the Internet can talk to any other part, it becomes possible to launch attacks from many Internet locations at once.

The distributed nature of a DDoS attack makes the attack difficult to mitigate. Which source addresses should be filtered as attackers, and which source addresses are those of legitimate customers? The attack patterns are purposely designed to make this challenging to discern, overwhelming the target with traffic, and resulting in service requests being denied. The overwhelming amount of traffic in this case was estimated to be as high as 620Gbps.

Brian Krebs, the security expert behind KrebsOnSecurity.com, describes himself as obsessed with security, maintaining relationships with many other smart information technology (IT) experts to keep his skills honed and his writing deeply informed. Yet, even with his technical prowess, his site fell victim to this DDoS attack. Why?

On Friday, October 21, 2016, the Domain Name Services (DNS) provided by Dyn came under a DDoS attack. This attack was even more nefarious than the one on Krebs. The attack impacted Dyn and Dyn's customers, rendering their services effectively offline. If name resolvers were unable to reach Dyn DNS servers, then the domain names hosted by Dyn would not be able to be resolved. Various media outlets reported impacts to AirBnB, Amazon Web Services, Box, FreshBooks, GitHub, Netflix, PayPal, Reddit, Spotify, and Twitter, just to name a few.

In the DDoS attack launched against Dyn, an estimated 40,000–100,000 sources were estimated to generate an aggregated peak traffic of a staggering 1.2Tbps. This attack came in waves, rendering services down and up and down again while mitigation efforts were deployed.

What do these attacks have to do with the Internet of Things (IoT)? In both the Krebs and Dyn attacks, poorly secured IoT devices were leveraged. DDoS attacks often work via botnets. In a botnet, many Internet-connected computing devices are compromised due to some security flaw. When the flaw is exploited, command-and-control software is installed, bringing the device under the control of a remote party.

When enough devices are controlled, they can be used by the controller to launch a coordinated attack against a target. The Internet is used as the network

1. Krebs, "KrebsOnSecurity Hit with Record DDoS."

to carry out the attack. IoT is making it easier to create botnets and launch powerful DDoS attacks, such as the ones against *Krebs on Security,* Dyn, and many Dyn customers.

In fairness, IoT is not specifically to blame. The issue is more one of a huge number of devices connected to an intentionally open Internet, which are not often touched, and rarely have the processing power to dedicate a lot of effort to security. After all, IoT is merely a handy term to describe the notion of a world in which unexpected things are connected. Home automation smart devices such as thermostats, garage door openers, refrigerators, lights, video surveillance, locks, and home entertainment devices, not to mention cars, are part of the brave new IoT world.

In the realm of business, smart cities can control parking, optimize traffic, and micromanage electrical power distribution. Smart buildings can optimize environmental systems, managing heating, cooling, and lighting in harmony with physical building design and efficiency protocols. Smart factories monitor manufacturing processes, rooting out the tiniest inadequacies in production, squelching problems before they become manufacturing defects.

Energy producers also add smart devices to the IoT panoply, relying on sensors to govern oil and gas production, as well as the operation of wind farms generating electricity.

As the usefulness of IoT has exploded, IoT device manufacturers have focused on functionality more than security. Far too many IoT devices are shipping with easily defeated security measures, making them easy targets to add to a botnet for use in a later attack.

Herein lies one of several networking challenges introduced by the Internet of Things considered in this chapter:

1. **IoT security.** How should IoT devices be secured? What is unique about them compared to traditional compute? What design constraints are introduced by IoT security peculiarities?

2. **IoT connectivity.** As IoT devices flourish in number, what strategies are required by network engineers to best connect them to the local networks they serve as well as the Internet? This is a more poignant consideration than it sounds, complicating both addressing schemes and communications protocols.

3. **IoT data.** The amount of data produced by IoT devices in certain applications places a burden on the Internet that network engineers must consider. For IoT data to be made use of in a timely way, it must be processed as quickly as possible. The latency of public cloud introduces an IoT data processing challenge the network engineer must consider.

IoT Security

Some writers have characterized the IoT as the *Internet of Terror;*[2] others, *The Internet of Stupid Things*.[3] Why the derision? The story of *Krebs on Security* could be the story of every website unless some method is used to secure these "smart" devices now proliferating. The question is: How can you secure devices that have very little processing power, very little memory, and generally cannot or will not be updated on a regular basis?

There are several possible answers to this question—answers many security and network engineers believe do not, ultimately, actually provide a sufficient answer.

Securing Insecurable Devices Through Isolation

One obvious thought to secure IoT devices is to treat them as you'd treat any computing device: lock them down. There are well-known processes to minimize the attack surfaces of Linux and Windows operating systems. For example, a Windows 10 workstation might have a specific policy pushed to it through centralized control. Or, a Linux server might be instantiated using a template predefined by operators to turn off unused services, reducing the attack surface.

However, IoT devices are not running full-blown operating systems and might lack the tools required to secure them in this way. In addition, securing them might break some of their functionality.

Since the IoT devices themselves cannot be secured, controlling access to the network is currently the main strategy for IoT device security. The idea is to allow the IoT device access to the network, but to strictly limit what the IoT device can reach through the network. While the Internet is an open transport, private networks containing IoT devices are not presumed to be open. Operators have the opportunity to control traffic flows and limit the chance their IoT devices become compromised. In addition, operators can prevent their IoT devices from being used as minions in a DDoS attack, even if they are compromised.

IoT device access control can be accomplished in a couple of different ways; the following sections consider both service-based isolation and endpoint isolation.

Service-Based Isolation

In the IoT service-based isolation model, IoT devices with a common purpose are assigned to a specific network segment that is isolated from the rest of the network by a security service. The security service implements a policy filtering traffic flowing

2. Neville-Neil, "IoT."

3. Huston, "The Internet of Stupid Things."

into the IoT network from the outside world. The security policy also limits what the IoT devices can access beyond the service, as illustrated in Figure 29-1.

This strategy can work in scenarios such as building environmental control. In this model, IoT devices such as thermostats and HVAC controls can participate on a common network. In fact, they might need to communicate on a common network to share data with one another or with a centralized controller. The HVAC environmental network is isolated from all other building networks by the programmed behavior of the security appliance or service policy.

Exceptions to this policy can be made as needed to support business functions. For instance, IT operations might require access to support HVAC functions and monitor equipment. Workstations supporting building maintenance might require access to the network as well. The security policy governs this traffic, limiting packet flows to what is absolutely required.

The result of this strategy is that IoT devices become much more difficult to access remotely. The IoT devices themselves are still insecure, but their attack surfaces have been isolated to expose them to as few outside hosts as possible while still allowing them to perform their functions.

Endpoint Isolation

Endpoint isolation takes the idea of appliance-based isolation one step further. In this approach, every IoT device on the network is isolated from every other device on the network. This is managed at the ingress network port. Where the IoT device is plugged into the network, there is a filter in place strictly limiting what other systems the device can communicate with, and vice versa.

The problem with endpoint isolation is one of administrative burden. Maintaining appropriate whitelists for every IoT endpoint on the network is tedious at best and an impossible-to-scale challenge for IT operations teams at worst.

To handle this challenge, some IoT security solutions relying on endpoint isolation employ central management. In this scenario, an administrator creates security policies, assigning them to groups. Then, IoT devices are placed into the proper

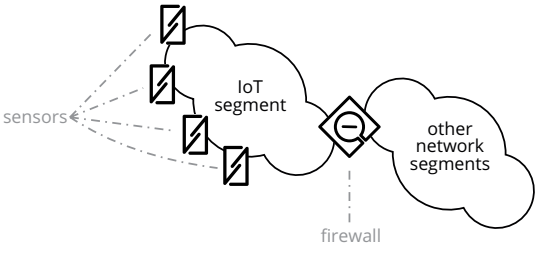


Figure 29-1 *Using a Security Service to Separate IoT Devices from the Rest of the Network*

groups. The central controller takes care of pushing the appropriate traffic filtering policy to the ingress network port, isolating the IoT device. Figure 29-2 illustrates.

Endpoint isolation is an effective approach for IoT devices that operate as loners. For example, endpoint isolation works effectively in healthcare, where medical devices might require access to just a small number of other systems on the network, and do not participate as a member of a collaborative sensor group.

Unikernels

One technology effort that could significantly impact IoT security is unikernels. Unikernels are stripped down versions of operating systems that include just the functionality strictly required for the application they support. This approach dramatically reduces the attack surface of the underlying system.

What is meant by "attack surface" in the context of IoT devices? Operating systems often ship with many libraries and supporting applications by default, selected at the whim of the operating system (OS) distribution creators. Operating systems are bundled in this way because the resources are known to be *commonly* used, even if they are not *always* used. Attackers will attempt to leverage any resources they can, hoping to discover vulnerable code when they run their exploits.

For example, an operating system might include an antiquated storage library, included by the distro makers just in case a user is running the OS on an older system. If an old storage library is not required to access any of the disk hardware in the system, it is both useless to the operator and potentially exploitable by attackers. The library makes up part of the "surface" that can be attacked.

Thus, on IoT devices shipped with full operating system distributions, a larger than necessary attack surface is present. Unikernels strip out the daemons, libraries, and applications not required by the operating system nor the application. The result

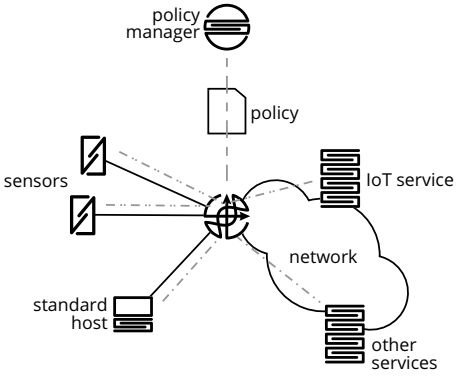


Figure 29-2 *Endpoint Isolation with IoT Devices*

is a barebones, highly efficient operating system environment containing just what is needed to support the applications running.

Don't think of "barebones" negatively in this context. Operating systems function in multiple contexts. Some are used actively by end users—for example, as desktop operating systems, programming environments, digital media creation, and so on. In this context, a barebones unikernel would be an overly constrained platform likely to inconvenience the user to the point of madness.

However, in the context of a specialty device manufactured with a singular purpose, a unikernel seems perfectly suited. An operating system built for the purpose of delivering a single application, most likely on a customized bit of hardware that will never change, cries out for a secure and efficient environment.

Process impact aside, there is little conceptual downside to an IoT device manufacturer taking the unikernel approach. Despite the advantages, unikernels have not been widely adopted by IoT device manufacturers yet.

Unikernels are raised here as a security enhancement for IoT devices, but how many consumers of IoT devices will understand this point and make their purchases accordingly? Perhaps consumers, armed with the knowledge of unikernels and sufficient buying power, could demand manufacturers properly implement unikernels as the base platform their applications run on.

The End of Open Networking?

This chapter was introduced with the notion of an open Internet, the intentionally open nature of the Internet that has made possible free communication as well as nefarious attacks. Although supportive of the open Internet, the Internet Engineering Task Force (IETF) offers perspectives on best practices, codified as Best Current Practice (BCP) documents.

One such BCP of interest to the IoT discussion is BCP38, which is a pointer to RFC2827: *Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ Internet Protocol (IP) Source Address Spoofing.* BCP38 proposes all network operators filter traffic with inappropriate source addresses. "Inappropriate" means source addresses that should not be used to originate packets or should not appear on the wire on the interface where they were received.

For example, RFC1918[4] specifies address blocks for private use only:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

4. Moskowitz et al., *Address Allocation for Private Internets*.

RFC1918 blocks are not routable across the public Internet. Traffic containing source addresses from RFC1918 address space should never appear on the public Internet. IPv6 also has several kinds of globally unroutable addresses, such as link local addresses, and unroutable globally unique addresses.[5] Therefore, BCP38 (RFC2827,[6] updated by RFC3704[7]) suggests they should be filtered.

DDoS attacks often use spoofed—fake—source addresses in their attack. The attackers don't require a response, and obfuscating source addresses makes it more difficult to track down the actual hosts propagating the attack. RFC1918 addresses are useful here, but any addresses could be, and are, used in DDoS attacks.

By dropping traffic with spoofed addresses, DDoS attacks should be at least partially mitigated. Writing filter lists that drop address blocks such as RFC1918 is straightforward. Also straightforward is the filtering of bogons, containing non-routable address blocks, plus unassigned public address blocks.[8] Unicast Reverse Path Forwarding (uRPF) ensures traffic is only forwarded if the source address is reachable through the interface through which the packet was received.

From the standpoint of IoT, BCP38 and uRPF are best practices because they help contain certain types of attacks sourced from compromised IoT devices. The recommendations in BCP38 are not as widely deployed as they could be because of various operational and performance issues involved in deploying these techniques. For instance, the Mirai botnet used in the Krebs and Dyn DDoS attacks appeared to come from spoofed addresses—reportedly tens of millions of addresses from tens of thousands of sources. And yet—the attacks were successful.

IoT Does Not Represent New Security Challenges

Interestingly, IoT does not represent any new security challenges to network engineers. The issues of DDoS, address spoofing, ingress filtering, etc., are familiar to networking professionals. However, IoT makes these issues more poignant.

Distributed denial of service attacks have always been painful. However, the proliferation of poorly secured, easily exploitable, IoT devices has rendered DDoS attacks easier to execute and more harmful once launched. Thus, the industry has been forced to address the issue, reminding network engineers and equipment manufacturers of the mitigation strategies.

IoT also raises the stakes because IoT devices tend to gather data of potential interest to attackers. An attacker, for instance, might be very interested in being able

5. Haberman and Hinden, *Unique Local IPv6 Unicast Addresses*.
6. Senie and Ferguson, *Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing*.
7. Baker and Savola, *Ingress Filtering for Multihomed Networks*.
8. "The Bogon Reference Page."

to gain access to a building through an IoT device. What about IoT sensor data coming from a natural gas pipeline? While this chapter has focused on IoT devices being used as incubators for malware, network operators should remember IoT devices also represent targets of opportunity in and of themselves because of the data they sometimes have access to.

IoT Connectivity

A different set of challenges for the network engineer is represented by IoT connectivity requirements. Typical network devices are powered predictably by the electrical grid, and are connected to a local wired Ethernet or wireless IP network. These sorts of network devices, which do represent a significant portion of the Internet of Things, are straightforward, as they are connected to the network in familiar ways.

However, many IoT devices are not able to be connected to the network in the typical fashion. They might be deployed over a wide geographic area, where enterprise-class WiFi networks do not reach.

Other IoT devices might be battery-powered, requiring an especially low power draw to remain functional for a long time without maintenance. Given these constraints of geography and battery power, what sort of networking technologies can be used?

Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) has been billed by the Bluetooth Special Interest Group (SIG) as having been built for the Internet of Things.[9] As a well-recognized industry standard, Bluetooth Classic and now BLE have indeed had a positive impact on IoT devices, particularly those in the consumer space such as wearables and smart home devices.

Bluetooth, including BLE, is a short-range protocol. Short-range means distances of approximately 100 meters or less. Therefore, Bluetooth is commonly found in home, auto, and personal area network applications such as wearables.

What does BLE do differently to reduce power consumption compared to Bluetooth Classic? The general answer is rather intuitive: BLE does less, and does "less" less often. This does not mean BLE is feature-poor or incapable. Rather, the word *less* as it is used here means BLE is designed to perform specific networking functions, eschewing others, all bound by the constraint of minimal power consumption.

9. "SIG Introduces Bluetooth Low Energy Wireless Technology, the Next Generation of Bluetooth Wireless Technology."

Compared to Bluetooth Classic, BLE notably does less in the areas of data rates, throughput, and connection setup time:

- **Data rates.** Classic is specified for 1–3Mbps, while BLE rates are as low as 0.125Mbps and as high as 2Mbps.
- **Throughput.** Classic specification is for 0.7–2.1Mbps, while BLE is specified for a much lower 0.27Mbps.
- **Connection setup time.** Classic Bluetooth connection setup time is around 100ms, while BLE reduces this to 6ms.

Power consumption itself is not part of the official Bluetooth specification, but common experience suggests Bluetooth Classic hovers around a 1W power draw, while BLE ranges between 0.01 and 0.50W.

The power savings is coming from, in part, the duty cycle. How long must a device be powered—in this case, the Bluetooth host chip—before it has completed its task and can go back to a sleep state? When comparing Bluetooth Classic to BLE, the BLE duty cycles are reduced in time and/or frequency, resulting in a greatly reduced power draw.

For example, a developer can set several duty cycle parameters affecting a BLE device in a connected state. Here are two:

- **The interval between data exchanges known as the "connection interval."** Higher intervals reduce power consumption, the tradeoff being application performance could be reduced because data can only be exchanged once each interval. Devices cannot send data if they are sleeping.
- **Slave latency.** In a Bluetooth pairing, one device is the master, and the other is the slave. Assuming no data to send, the slave can be configured to not check in with the master for a reasonable range of intervals, each skipped interval saving power.

The result of BLE is the enablement of certain IoT devices to run weeks, months, or years on coin-sized batteries. However, BLE's reduced duty cycles and resulting low power draw mean it is poorly suited for applications such as audio streaming. Therefore, BLE headphones are, at the time of this writing, not available. Why not?

Audio streaming demands frequent duty cycles, as there is always new audio information to be sent between master and slave. To make audio streaming work in a BLE context, new audio codecs might need to be devised to come up with a send/receive duty cycle amenable to fulfill the "low energy" part of BLE.

Other low-power, short-range protocols in use for IoT include Zigbee and Z-Wave.

LoRaWAN

Aside from the consumer space, the Internet of Things has seen uptake in the world of industry. Industrial applications often require networking services extending beyond the comfortable confines of a well-powered and well-connected building.

A municipality might use IoT to leverage smart city technology in areas such as fire hydrants, parking, street lighting, and waste management. Farming can use IoT to manage land and irrigation and to track animals. Utilities could use IoT to meter gas and water usage.

To handle the distance and low-power requirements of many of these scenarios, low-power, long-range communications protocols have been created. One such is LoRaWAN (Long Range Wide Area Network).[10]

LoRaWAN is a chirp spread spectrum, wireless communications protocol operating in unlicensed spectrum below 1GHz, creating a low-power wide area network (LPWAN). A LoRaWAN-based LPWAN offers data rates between roughly 0.3Kbps and 50Kbps over a range of 2Km to 15Km, depending on the environment. LoRaWAN is a secure protocol, offering both factory preprogrammed network authentication and over-the-air activation of participating nodes, as well as multiple layers of strong encryption.

A LoRaWAN network includes two major components:

- The end nodes with which to communicate. In this context, these are IoT sensors.
- The sensors communicate via LoRaWAN back to a gateway. The LoRaWAN gateway is a bridge between the LoRaWAN network and a traditional wireless or wired network used to process the sensor data. The gateway serves to decrypt inbound sensor data received via LoRaWAN radio and repackage the payload for transport across the traditional network.

The compromise LoRaWAN makes, allowing it to function as a low-power, high-range network protocol, is low bit rates. The data throughput across a LoRaWAN network is seemingly miniscule at a max of 50Kbps, especially when considering data center Ethernet speeds of 100Gbps are commonplace.

However, LoRaWAN's low bandwidth represents a design solving a specific networking challenge. For many applications, IoT sensors do not need to transmit or receive enormous amounts of data, but they do need to communicate over long distances using battery power. Thus, LoRaWAN is fit for purpose, providing a power-efficient means of transmitting small amounts of data over long distances.

10. "LoRa Alliance Technology."

LoRaWAN transmitters come in three classifications:

1. **Class A devices** are the most power efficient. Class A device radios sleep unless they have data to send. Once data has been sent, they remain awake for two receive windows, during which they can receive data. If more data needs to be sent to a Class A device than can be delivered during the two receive windows, the data must be queued until the next receive window opens.
2. **Class B devices** operate like Class A devices, except for the addition of scheduled receive windows. While Class A devices can only receive data after sending data, Class B devices can also receive data during regularly scheduled receive windows. The additional receive windows draw power to engage the LoRaWAN radio, and thus Class B devices are less power efficient than Class A devices.
3. **Class C devices** are different from Class A and B devices because Class C devices listen all the time, except when transmitting. Class C devices are appropriate for applications where the IoT sensor needs to receive data regularly from the central network, and the central network cannot wait for a remote device to open a receive window. The downside of listening constantly except when transmitting means the radio is constantly drawing power. Therefore, Class C devices are expected to use a grid-connected power supply, as batteries would be drained too rapidly in a Class C application.

Other low-power, long-range protocols in use for IoT include Sigfox and Neul.

IPv6 for IoT

IoT connectivity on IP networks faces an additional challenge—addressing. Network operators are comfortable with IPv4 addressing schemes, and might be tempted to use IPv4 in their IoT networks. While there is nothing wrong with this per se, IoT does present some interesting challenges that make IPv6 addressing attractive, including

1. **Scale.** IoT sensor networks have the potential to be vast, depending on the application. IPv6 makes the number of IoT sensors in a network a nonissue, as the address space is all but infinitely large. Starting with a well-planned IPv6 addressing scheme in an IoT network means never having to readdress the network.
2. **The elimination of Network Address Translation (NAT).** NAT is commonly used to translate blocks of private RFC1918 IPv4 address space into one or more publicly routable IPv4 addresses. Some networks regard this as a security feature, while others consider NAT a nuisance, as some applications require

workarounds to function properly in the presence of NAT. NAT also makes two-way communication between devices difficult. IPv6 has no requirement for address conservation, i.e., hiding blocks of RFC1918 addresses behind a single public IP address. IPv6 could be used in an IoT network to offer two-way communication and improve endpoint identification and authentication.

3. **Mobile IoT.** If IoT devices are not stationary, it is possible they will move physically, as well as logically within the sensor network, disappearing and reappearing as they move between associations with various access points and gateways. IPv6 networks are well suited for mobile devices.

Another question network engineers must consider about IP addressing more broadly is whether it makes sense to assign IP addresses to IoT devices. In one sense, this is a strange question to ask. "Well, of course, there needs to be an IP address on IoT devices! There needs to be an IP address on *everything.*"

The "IP or not" question is more nuanced, however. In traditional networks, networking professionals do not need to consider whether an IP header introduces inefficiency. Hardware ASICs are optimized to process these headers, and bandwidth is plentiful. In addition, IP headers are used to identify source and destination addresses, carry interesting information about the flow they are a part of, and make forwarding decisions. Network engineers used to fast Ethernet and wireless networks and devices ubiquitously connected to the Internet might have a hard time imagining networks without IP.

For IoT devices connected to traditional networks and to networks with no bandwidth concerns, IP addressing does not introduce any new issues. However, much of the IoT domain leaves traditional networks behind.

Consider low-power WANs (LPWANs) like LoRaWAN. When an IoT sensor is sending its data to a receiver, what is the most important part? The payload—the data itself. Any framing or encapsulation is overhead, even though it is necessary to deliver the data. Therefore, in an LPWAN, the key is to reduce the overhead by minimizing the number of bits encoded and sent over the air.

You might recall LoRaWAN's highest bandwidth is around 50Kbps. Suddenly, the size of an IP header becomes an interesting question. The size of the IP header is why there is no such thing as IP over LoRaWAN. LoRaWAN packets go to a LoRaWAN gateway, where the payload can be repacked into IP datagrams and sent to points beyond. Why? IP packets with their pesky headers are simply too inefficient to send over the LoRaWAN network. Figure 29-3 illustrates.

LoRaWAN's lack of IP does not imply it is impossible to use IP addressing for IoT. Rather, it means every networking tool is designed for a specific purpose. For example, in the IETF's RFC4944, 6LoWPAN is specified to integrate IPv6 with IEEE 802.15.4 mesh networks, including compression of the IPv6 header wherever

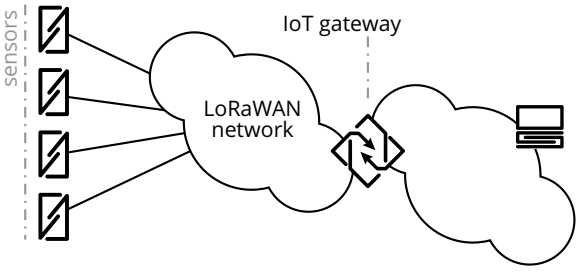


Figure 29-3 *LoRaWAN-to-IP Operation*

possible. Reading through RFC4944 exposes the many technical challenges of this integration.[11] Yes, the technology exists, and while difficult to implement practically, it can be done.

The question then comes back to the network engineer. Given a choice of connectivity technologies and addressing schemes for IoT, which one is the right one? The answer depends on the connectivity requirements. Different requirements will lead to different answers.

IoT Data

Some IoT sensors produce a significant amount of data. What happens when the data must be acted upon in real time? How should the data be processed?

The idea of fog computing—also termed edge computing—is that certain IoT sensors stream too much data to be processed far away, such as in the public cloud. In these cases, IoT data must be analyzed locally to be of real-time value in many applications, particularly industrial ones. Sending the data far away, processing it, and bringing back the results would take too long.

Besides tending toward high latency, bandwidth to the public cloud is simply not cheap enough to size pipes sufficiently large for IoT applications. Thus, the term *fog* is meant to conjure an image of a cloud close by, rather than one far away. Sending data into the local fog allows for speedy analysis and timely results.

The fog computing model is to process IoT data as close to the sensors as possible. Many IoT devices do not have local data centers in which to perform data processing. Those with a data processing center closer than the public cloud might find connectivity is often fragile. Therefore, fog computing sometimes looks like a

11. Montenegro et al., *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*.

small, dedicated device piggybacked on the sensor accepts data and performs processing. This could also mean data processing software resident on IoT network gateway devices.

Use cases for fog computing include many examples from Industrial IoT (IIoT):

1. **Locomotive fuel efficiency.** Engine sensor data is coupled with GPS data to reduce engine idle time, saving significantly on fuel. Even at idle, locomotives utilize a large amount of fuel. Avoiding excessive fuel burn requires real-time data analysis as the locomotive moves across the landscape.
2. **Cavitation alerts.** Temperature, input pressure, output pressure, and water velocity are monitored in real time to detect the conditions in which an air bubble might be introduced into a water moving system. These air bubbles, or cavitations, can destroy water pumps.
3. **Wind energy forecasting.** Wind turbine data is analyzed to predict power yield for the next 24 hours, a legal requirement in certain parts of the world where power grids are carefully managed by governments.
4. **Factory yield optimization.** Sensor data is analyzed to discover manufacturing problems resulting in a bad run of products, improving overall quality and reducing factory downtime.

While fog computing is a data processing paradigm more than a networking paradigm, the computing requirements of IoT make an implicit demand on network engineers. Where there is a great deal of data, there must be a capable network to move the data. Therefore, understanding the load created by IoT sensor data will inform the IoT network design.

Final Thoughts on the Internet of Things

The Internet of Things is an interesting area of new deployment and research that is ultimately likely to reshape the way the Internet is seen. Instead of providing connectivity for people searching for websites and social connections, the primary job of the Internet, in terms of traffic flow, will be to connect sensors to cloud-based services. These cloud-based services will, in turn, peek into every area of life, raising security and privacy concerns that need a lot of thought to untangle. This chapter has provided some ideas of how such an Internet might work and how it might be secured.

The next chapter will consider another future-looking topic, the future of network engineering.

Further Reading

Baker, Fred, and Pekka Savola. *Ingress Filtering for Multihomed Networks*. Request for Comments 3704. RFC Editor, 2004. doi:10.17487/RFC3704.

Banks, Ethan. "Foghorn: Real-Time Decision Making for IIoT." *Packet Pushers*, September 14, 2016. http://packetpushers.net/foghorn-iiot/.

"The Bogon Reference Page." Team CYMRU. Accessed July 17, 2017. https://www.team-cymru.org/bogon-reference.html.

Cantrill, Bryan. "Unikernels Are Unfit for Production." Blog. *Joyent*, January 22, 2016. https://www.joyent.com/blog/unikernels-are-unfit-for-production.

Haberman, Brian, and Robert M. Hinden. *Unique Local IPv6 Unicast Addresses*. Request for Comments 4193. RFC Editor, 2005. doi:10.17487/RFC4193.

Hilton, Scott. "Dyn Analysis Summary of Friday October 21 Attack | Dyn Blog." Corporate. *Dyn*, October 26, 2016. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

Huston, Geoff. "The Internet of Stupid Things." *APNIC Blog*, April 30, 2015. https://blog.apnic.net/2015/04/30/the-internet-of-stupid-things/.

Krebs, Brian. "KrebsOnSecurity Hit with Record DDoS." Blog. *Krebs on Security*, September 16, 2016. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/.

"LoRa Alliance Technology." Standards Body. *Lora-Alliance*. Accessed July 17, 2017. https://www.lora-alliance.org/technology.

Madhavapeddy, Anil, and David J. Scott. "Unikernels: Rise of the Virtual Library Operating System." *Queue* 11, no. 11 (December 2013): 30:30–30:44. doi:10.1145/2557963.2566628.

Montenegro, Gabriel, Jonathan Hui, David Culler, and Nandakishore Kushalnagar. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. Request for Comments 4944. RFC Editor, 2007. doi:10.17487/RFC4944.

Moskowitz, Robert G., Daniel Karrenberg, Yakov Rekhter, Eliot Lear, and Geert Jan de Groot. *Address Allocation for Private Internets*. Request for Comments 1918. RFC Editor, 1996. doi:10.17487/RFC1918.

Neville-Neil, George. "IoT: The Internet of Terror." *Queue* 15, no. 3 (June 2017): 10:19–10:24. doi:10.1145/3121437.3121440.

Senie, Daniel, and Paul Ferguson. *Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing*. Request for Comments 2827. RFC Editor, 2000. doi:10.17487/RFC2827.

"SIG Introduces Bluetooth Low Energy Wireless Technology, the Next Generation of Bluetooth Wireless Technology." Society. *Bluetooth*. Accessed July 17, 2017. https://www.bluetooth.com/news/pressreleases/2009/12/17/sig-introduces-bluetooth-low-energy-wireless-technologythe-next-generation-of-bluetooth-wireless-technology.

Review Questions

1. Explain the why IoT has garnered so much attention from security practitioners.
2. Explain the difference between isolation using an appliance or service and endpoint isolation.
3. What does the term *duty cycle* have to do with power conservation in IoT devices?
4. Give some examples of how Bluetooth Low Energy devices reduce power consumption when compared to Bluetooth Classic devices.
5. In IoT devices using LoRaWAN for radio communications, explain why a Class A device can run on battery power, while Class C devices should have dedicated power supplies.
6. Why does IP addressing introduce a technical challenge for LPWAN communications protocols?
7. In one sentence, explain why edge (fog) computing is useful.
8. Research BCP38, and explain why it is not widely deployed.
9. The text considers IoT in the context of DDoS attacks; research the impact of IoT on large-scale control systems (such as the power grid), and explain the risks involved.

This page intentionally left blank

Chapter 30

Looking Forward

Learning Objectives

After reading this chapter, you should understand:

- The concept of modeling languages and how automation may change the shape of network engineering
- The application of hyperconvergence to network engineering
- The concept of intent-based networking and the tradeoffs involved in this idea
- What machine learning is and why it might be hard to apply to network engineering
- The concept of Named Data Networking
- The concept of blockchains and how they might impact network engineering
- The ongoing reshaping of the Internet

Just about every culture in the world has some saying similar to

Those who forget the past are doomed to repeat it.

A variant of this is RFC1925, rule 11, which states

Every old idea will be proposed again with a different name and a different presentation, regardless of whether it works.[1]

This book began with a simple idea: you can use this to your advantage. By learning what is old, you can learn what will be proposed as new in the future. This mindset of looking to the past to understand the future can be codified in the process:

- What is the problem being solved?
- What range of possible solutions have been proposed to solve this problem?
- How have these solutions been implemented in the past?

Perhaps two more thoughts are in order, as well:

- What are the tradeoffs involved in solving the problem this way?
- How does this solution interact with other problems and their solutions in a larger system?

These rules, however, only give you a dim view of the future; they provide the "guard rails" of what might be developed, and a framework within which to understand and apply these developments.

What of the larger market? Will the skills and mindset so carefully laid out in the previous chapters and pages be useful in five year's time? Or twenty? Predicting the future, as they say, is hard because it changes so much. It is particularly hard in the case of network engineering, which likely has more than one future at any one time.

This chapter is going to take a different direction from the previous chapters. Each section will describe a different movement in network engineering and where this movement might lead in the future. Some of these trends will overlap, or depend on one another to some degree; others will be completely independent of the others. Remember these forward-looking snippets are spun from current trends, so any particular set of ideas will likely be changed radically by the time they come to pass—or perhaps they will be found impractical, and not come to pass at all. A more likely future is *all* of these futures become real in some networks.

It is difficult to remember, when working on a single network, in a small corner of the network engineering world, how large the network engineering world is. While network engineering is small in comparison to many other subcultures of the larger engineering world, and tiny in terms of the larger world, it is still a large world, with

1. Callon, *The Twelve Networking Truths*, 1.

many different subsets. There will always be businesses that take on the future by thinking differently. Some will succeed, many will fail, but all of them will have a different vision of what information processing needs to look like, and hence how to build a network to get done the work they need done.

Pervasive Open Automation

The programmatic configuration of network devices is already widely used in many networks; you can be confident this trend will continue and accelerate in the future. The age of the command-line interface (CLI) is largely over; programmatic interfaces will take the place of the CLI.

What has stood in the way of pervasive network automation in a multivendor network is a standardized Application Programming Interface (API). In a multivendor network, the API used to configure and manage each device will vary from vendor to vendor. Platform capabilities also vary within and between vendors. Thus, there are differences both in what can be done, preventing rapid, industrywide adoption of automation, as tooling must be written to support multiple vendors with their sundry interface nuances.

Modeling Languages and Models

The beginning of a solution in this space is rethinking the way network devices and protocols are modeled. What has traditionally been done—in fact, what the CLI does—is to focus on the information to be carried. Much like a fixed length packet encoding (see Chapter 2, "Data Transport Problems and Solutions"), the model is embedded in the CLI model. The metadata, or information about what is being configured, is carried in the configuration manuals or CLI help system.

An alternative to this is to focus on the modeling language first. In this solution, a modeling language is designed to act more like a Type Length Value (TLV) system; information about the information is provided separately from the information itself. This allows implementations to work around changes in the way data is represented, even ignoring information they do not understand how to process explicitly.

One such modeling language is YANG, a standard shepherded and managed by the Internet Engineering Task Force (IETF). Models can be built describing an interaction with a protocol or process, rather than a specific implementation, using the YANG modeling language. Rather than writing automation processes that expect a specific API or network device chipset, the idea is to automate against a model. The automation process will then work with all devices conforming to the models in use. The model functions as an abstraction layer.

One consortium of network operators creating such models is called OpenConfig. OpenConfig participants include Google, AT&T, Facebook, Netflix, CloudFlare, and Microsoft, among several other major service providers and large network operators.

OpenConfig has contributed many network models to the community, covering a diverse set of network elements, including policy, interfaces, lower- and higher-level transport protocols, and control planes. The OpenConfig group has also worked with the Internet Engineering Task Force (IETF) on these models, to help drive the industry toward a standardized way of representing the network. The IETF has taken the modeling work very seriously, attempting to bring together a complete and interoperable set of unified models.

A Brief Introduction to YANG

As a modeling language, YANG is not especially new. Many IETF RFCs have been released defining YANG or auxiliary interfaces related to YANG. Here are two key RFCs:

- In October 2010, the 173-page RFC6020, *YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, was published.
- In August 2016, RFC7950 weighed in at 217 pages, titled *The YANG 1.1 Data Modeling Language*. Even with the YANG 1.1 specification so recently published, there are rumblings within the IETF about extensions to 1.1 being added or possibly even a YANG version 1.2.

As of this writing, over 220 models are working their way through the IETF ratification process. In fact, YANG modeling has become so pervasive that the IETF has created a functional role of "YANG doctor" whose job it is to validate proposed YANG models.

YANG is meant to be human-readable, in contrast with the eXtensible Markup Language (XML), which tends to be read more easily by machines than people. YANG models are published as modules, where a module contains all the objects required to define some specific networking feature. Modules can reference other modules by importing external modules or using includes of submodules.

The structure of a YANG model is a tree with node objects, conforming to a specific hierarchy:

- A module fits into a namespace, described with a Uniform Resource Locator (URL).

- A prefix describes how a module is referenced inside the module or by other modules. Think of a YANG prefix as a shorthand description of a YANG module.
- There are at least four node types in YANG. A leaf object contains a value logically located at the end of a tree branch. Leaf-lists are sequences of leaf objects. Lists are collections of many sorts of objects, including lists and leaf-lists. Containers can hold lists, leaf-lists, leaves, and other containers. These all serve to organize elements in the YANG model.

The problem with YANG is not with the modeling language itself; YANG is well understood and in use by standards development organizations as well as consortiums such as OpenConfig. Despite this demonstrated level of industry enthusiasm, networking equipment vendors have been slow to include YANG models in their products.

Vendors are often slow to support YANG because *vendors need to differentiate to sell a product*. YANG models offer a baseline of networking functionality, or a lowest common denominator, so in some sense, configuring everything through a standard set of models described in YANG would "level the playing field."

Thus, vendors have been not overly enthusiastic with their YANG support, unless compelled financially by large, persistent customers. The OpenConfig project is one industry attempt to bring operators together to combine their buying power around specific requests to support YANG.

Looking Forward Toward Pervasive Automation

Standardized network modeling is a key to enabling pervasive network automation. Once configuring a network device is a predictable exercise, then creating automation tooling becomes a simpler task. Today's network automation tooling is burdened by a plethora of interfaces, methods, and output that must be normalized for automation processes to work in an expected way across a multivendor network. The broad industry adoption of standardized YANG models would change this aspect of network engineering. Automation without something like YANG will continue to move forward, but not as quickly or efficiently as it could with a single modeling language used by every vendor and operator.

Hyperconverged Networks

Chapter 25, "Disaggregation, Hyperconvergence, and the Changing Network," describes the rise of hyperconverged compute and storage. Because the networking market often follows the compute and storage market in a broad sense it is worth

putting some thought into what network hyperconvergence might look like. What were the components of the hyperconverged system at the edge?

First, there is white box; the networking world is already moving in this direction. While network devices such as firewalls, routers, and switches were once purchased in an "appliance" model, many parts of the networking world are quickly moving toward a disaggregated model, where the hardware and software are purchased as separate "things." This enables the concept of white box—although the box might not be white. The terms *bright box* and *gray box* attempt to capture buying boxes from brand-named vendors, but rather than buying them for their software capabilities, you can now buy them for their hardware capabilities.

Second, there is scale out. The move from traditional hierarchical network designs, particularly in the data center, and toward a flatter spine and leaf design is the equivalent scale-out solution in the networking space. Rather than buying a chassis and adding cards as needed, you buy a set of single rack unit boxes and build a network that can be increased (or decreased!) in scope and scale by wiring more boxes in.

Third, there is pooling. Here several different trends in the networking world are working together to create the beginnings of a true pooling capability: the rise of dynamic overlay networks, software-defined networks, and network function virtualization.

To combine these three, consider the spine and leaf network built out of white box devices, with a dynamically created overlay network providing virtual sets of resources as needed. This kind of network can be

- Scaled in resources by adding more boxes to the spine and leaf underlay, as well as adding more network-based services to virtual machines connected to this underlying fabric
- Pooled by building virtual networks in an overlay to consume the services of any number of underlay devices as needed

One important question is the depth of the overlay required to build such a system; most of today's overlay solutions are very heavyweight, full-scale tunneling and based on either a "second control plane," or a centralized control plane (rather than a more flexible hybrid distributed + centralized control plane). What will eventually be needed in this space is a lighter-weight set of control planes and overlay system that will work with underlying hardware better—perhaps not even an "overlay" at all, but rather a set of services that can send isolated traffic through the network without the work of building an actual virtual topology. Segment routing may provide a path to such lightweight overlay solutions.

While there are commercial solutions in this space, and custom solutions built and operated by large-scale cloud providers, this is still a nascent market. The solutions available today, either based on vendor-specific hardware and software and focused on the Top of Rack (ToR) switch in the data center fabric, or on the hypervisor in the server, are generally hampered by a lack of communication between the network resources—the network processors sitting on the ToR switches—and the overlay switching requirements. Further, these solutions are hampered by the amount of configuration required to simply get the system going, particularly in the underlay space.

But these markets are growing and changing; VMWare, Cumulus, and others are working on solutions that will, over time, likely develop into such a hyperconverged solution. There will always be, of course, an appliance-based model; there will always be software and hardware purchased as a single system.

But the disaggregation and programmable network movements are paving the way for a new kind of network, more along the lines of hyperconverged compute, storage, and network access resources.

Many hyperconverged networks are likely to be vendor specific; only a particular vendor's gear will work with a specific hyperconverged solution. The beginnings of this kind of hyperconvergence, combined with vendor proprietary APIs for automation, are already apparent in the product lines of many vendors.

Intent-Based Networking

According to the manufacturers and pundits, intent-based management is the future of network engineering. There certainly seem to be a lot of good reasons to embrace the intent-based wave.

For instance, networks are certainly hard to configure, maintain, and troubleshoot today. The 2 a.m. rule is almost always violated today simply because the networks needed to support the applications that businesses choose to run drive a lot of complexity into the design and operation of the network. Operations personnel are left trying to reverse-engineer *this* configuration on *that* device at 2 a.m., trying to tease out every application that might be impacted if any of the various pieces are modified to solve a problem *right now*.

A lot of the apparent problem is in translating the business intent into designs, which then must be translated into configurations, which then must be translated into the combined configurations of hundreds of different intent chains spread out over many years of network operation, vendor changes, and the personal preferences, strengths, and weaknesses of individual network engineers.

It would certainly, it seems, be a lot simpler to just state your intentions and let the network translate those intentions directly into configurations. The amount of money you could save on hiring all those engineers who are doing the translation work manually would probably be enough to justify the change all on its own. An artificially intelligent process running on some virtual machine (perhaps in the vendor's cloud) can adjust your network settings based on your stated intent, the applications you are running, and experience with other customers, and produce an optimal network configuration for every business, all the time.

But when so many people are saying the same thing at the same time, particularly in the normally contrary world of network engineering, it is time to take a step back and consider where the tradeoffs might be in this rush to *intent*. If you have not found the tradeoffs, you have not looked hard enough.

What are the tradeoffs involved in intent-based networking?

A good place to begin is with the engineer sitting at home, working from a laptop, at 2 a.m., trying to resolve a network problem (or at least figure out whether the problem is the network or some other part of the system). Perhaps intent-based systems will be better documented than the engineer-configured systems today, but this does not seem likely. If an AI is involved, there is very little chance there will be any documentation, in fact, as no one really understands what decision an AI might make or why. Even ignoring the problem of whether or not an AI will ever be able to do the job at hand—monitor every element of every application in a network and every element of every network device, combine this information with the capabilities of each installed device, and make fine-grained adjustments in every area to provide optimal utilization and application support for every possible network and business requirement—it is difficult to see how a particular decision can ever be reverse-engineered to determine whether the network is running properly or not.

Another hard problem to solve here is *whose intent*? There must be someone, somewhere, who is determining which factors make a difference in determining intent and what should be done in response to an expression of intent. While AI systems might be able to handle some of this around the edges, humans will always need to at least train AI systems on what action to take, or what the *intent* behind any new feature is in networking gear, etc. Moving intent into the controller moves the interpretation of intent to configuration from local engineers, who are (arguably, at least) accountable to your business goals, to a vendor's cloud or intent server.

The next question to ask is: *what does this intent look like*? Is it something like "give the president's email priority over the receptionist's?" Or is it finer grained? If it is finer grained, then someone must interpret the business problem into some form of "intent language" (an intent YANG model, anyone?), which means understanding the system and its reaction to any sort of intent statement made to the system. If the *intent* is to stop hiring engineers, this is not the path to get there. What would be needed instead to save money on engineering staff is more like the model where

the administrator says, "prioritize the president's email"—but then a host of new problems arise.

Given the system has some sort of interface, will the interface be standardized or vendor specific? The more likely answer is vendor specific, because any "intent language" must be rich enough to be useful and allow the vendors to differentiate themselves in selling into an end-to-end business model. Assuming the goal is for applications to drive the intent interface, as well as humans, each application must now be able to talk to each vendor interface in some way. The single vendor tie-in quickly moves from the networking hardware and software into the entire ecosystem of applications.

Above all of these questions is a larger, systemic one lurking in the background: intent-based interfaces are ultimately a form of abstraction. While abstractions are *very* useful—in fact, engineers could not live without them—they also have side effects that are not realized until far into the abstraction process. First, all abstractions remove information, and all information removal reduces efficiency in some way (the optimal use of resources, time, etc.). Second, all nontrivial abstractions leak: things not visible outside the system are always somehow passed through to the next level up, but in a way that is difficult to understand and manage.

None of this is to say intent-based networking is impossible, nor it will not have good uses. Intent-based interfaces will probably be useful in a narrow range of applications, perhaps broadly deployed in large-scale networks. Intent-based interfaces will probably also be useful in smaller-scale networks, or specific kinds of topologies, where the business is so far disconnected from information technology that the attendant inefficiencies and complexities just do not ever become a concern.

Machine Learning and Artificial Narrow Intelligence

Whether or not pervasive open automation ever becomes a reality, applying machine learning to network management is an area of active research. Artificial narrow intelligence (ANI) overlaps with the goals of machine learning (though not the techniques) enough that many engineers will see the two as the same thing. To provide a more formal definition:

- **Data mining** is the process of discovering previously unknown patterns in large information sets.
- **Machine learning** is the process of optimizing a set of input variables to reach a specified set of goals.
- **Artificial narrow intelligence** is combining several different data mining and machine-learning subsystems into a larger system that approaches a natural (or even human) level ability to achieve some specific task.

Figure 30-1 illustrates putting these three things together in a network engineering context.

Figure 30-1 illustrates how you might use data mining to discover things about your network that are not otherwise obvious; this information might drive a machine-learning system that consumes a specific final network state, combines the mined information with known state information, and adjusts various network inputs in order to reach the specified state. If enough of these kinds of systems are merged to form a "natural-like" system for managing some part of network operations, this might (or might not) be considered ANI.

There are major hurdles to overcome in order to apply machine learning to network management, however. Specifically, data analytics relies on being able to process a consistent set of information over long periods of time, in order to find patterns and patterns in the changes. Networks probably have too high of a rate of change, and too much noise, for data analytics to be as effective in the network

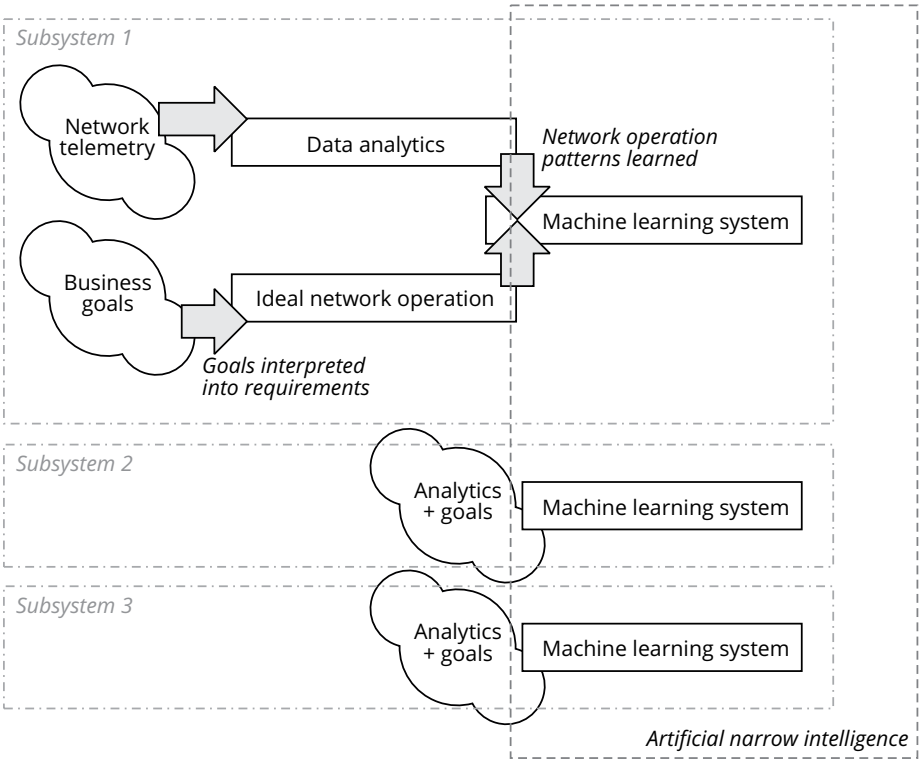


Figure 30-1 *Data analytics, machine learning, and artificial narrow intelligence in the context of network engineering*

engineering world as is in other areas. While some basic things might be learnable through data analytics, such as spotting interesting or unusual flows of information, it may be difficult to use machine learning to discover deeper patterns, as the pattern in a network as a system might just be "there is always change."

Machine learning is often narrowly focused in the same way as data analytics. Machine learning largely relies on consistent connections between inputs and outputs, no matter how many there are, to determine how to adjust the inputs to reach a certain output. There may not be enough consistency in networks as systems to allow this kind of fine-grained adjustment to be discovered through a machine-learning process, particularly given the constant rate of change that could plague the data analytics systems that machine learning would likely rely on.

Finally, machine-learning systems must be taught, or they must learn, based on an existing data set. As each network is essentially built to solve a single problem set, each network can effectively be treated as a unique machine-learning problem to solve. This could seriously hamper the ability of machine-learning systems to effectively "solve" network management issues.

These problems are a result of a basic problem in network engineering highlighted throughout this book: there is no "one right way" to build a network, a transport system, or even a protocol within a system. There is no "general theory of networks" you can rely on when building a machine-learning system to manage networks. An extended quote from someone working in this area as of this writing is useful in putting these problems into perspective:

> *Even though networking has "just massively more compute and massively more data" available, it's not yet clear how machine learning can be applied there, Meyer says. What's missing, he believes, is a theory of networking. A rich body of academic work backs the networks we use today, certainly, but there is no unifying theory defining how a network, in an abstract sense, ought to behave, or how it ought to be structured. The networks that form the Internet certainly share some core principles, but they weren't built from a central theory. They emerged through trial-and-error, "some good ideas and people telling each other how to do it," Meyer says.*[2]

Like intent-based networks, machine learning and ANI may play a narrow role in network engineering over time, but it seems unlikely you will see semiautonomous networks driven from an ANI anytime soon.

2. Matsumoto, "Why Machine Learning Is Hard to Apply to Networking."

Named Data Networking and Blockchains

Named Data Networking (NDN), which is loosely related to Content Centric Networking (CCN), relies on a simple trio of observations. First, the Internet Protocol stack of protocols, like every other networking system, is built on a narrow waist. The narrow waist is, in this case, the Internet Protocol (IP), as illustrated in Figure 30-2.

All complex systems are built with some sort of thin waist in this way; protocol and network design patterns count on these thin waist points (or *choke points)* to control complexity by hiding information (or the abstraction of state).

The second observation is that the Internet and most networks are primarily designed to distribute information—particularly information marshaled and described through metadata. The third observation is that IP is not very good at carrying information, but rather is designed to carry bits.

Named Data Networking Operation

Combining these observations, NDN asks: why should the thin waist of the Internet be a protocol that does not specialize in what the Internet does? Or rather, why not replace the thin waist of the Internet with a protocol designed to efficiently distribute data? Once you begin to look at the Internet, or any network, as a large distributed database, the problems to be solved become radically different than the transport and reachability problems considered in this book. Figure 30-3 illustrates the concept.

Assume you are looking for the song *Pleasant Valley Sunday* by the Monkees. Begin with standard IP, walking through the steps to retrieve this information at A from F:

1. A user clicks on a search result for *Pleasant Valley Sunday*, which indicates a copy of the song can be found at http://songserver.com/monkees/pleasant.
2. The host operating system looked up .com, then songserver.com, retrieving an IP address from the Domain Name Service (DNS).

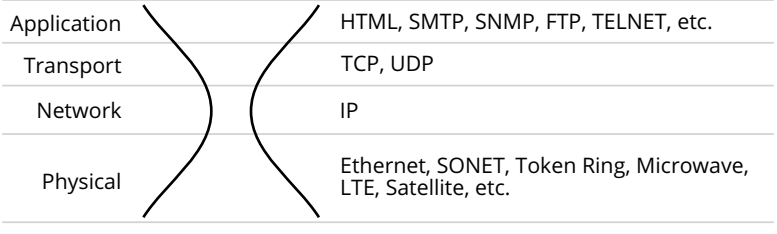


Figure 30-2 *The thin IP waist*

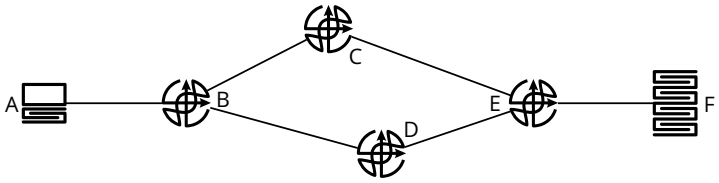


Figure 30-3 *A network on which to compare standard IP and NDN operation*

3. The host operating system begins a session with the IP address, ultimately starting a session with F.
4. The host operating system then performs any necessary authentication steps, such as putting a sign-in form on-screen, or trading some certificate—even perhaps undertaking some financial transaction to purchase a copy of the song.
5. The host operating system at A now downloads a copy of the song.

At every step in this process, the host builds a point-to-point link with some other system, such as a DNS server and the server on which the copy of the song resides. The routers along the path of this traffic just switch the packets; they do not cache any information, nor can they participate in the financial transactions or the authentication of the user. Compare the process using an NDN:

1. A user clicks on a search result for *Pleasant Valley Sunday*, which indicates a copy of the song may be obtained from /com/songserver/monkees/pleasant; note the difference in the ordering of the location of the data.
2. The host operating system sends this request to its upstream router, B, which examines the name of the object requested; it finds a path to a server claiming to have this information that is reachable via C, so it sends the request to C.
3. B again consults the name of the object requested and finds it has a path through E, so it forwards the request to E.
4. E consults the name of the object requested and finds it has a path through F, so it forwards the request to F.
5. F consults its local information store and finds it has a copy of this object in the location specified; it returns an encrypted copy of the object to E.
6. E stores a local copy of the encrypted object, examines the path through which the request for this object came, and sends a copy of the object to C.
7. C, likewise, stores a local copy of the encrypted object, examines the path through which the request for the object came, and sends a copy of the object to B.

8. B stores a copy of the encrypted object and sends it to A.
9. A, on receiving the object, now must find some way to unencrypt the encrypted object; to do this, it either contacts a third party to arrange a financial transaction or uses local information it has already stored to unencrypt the object.

The NDN version seems far more complex at first blush, but it does have several advantages. For instance:

- Rather than encrypting or hardening the session between the client (A) and the server (F), the object itself is encrypted; this means there is per object protection throughout the entire network. It does not matter if the memory of any particular device is compromised, because every object is encrypted as it is carried over the network.
- The metadata about the object is (or can be) exposed, allowing each device to handle the data according to local policy, including "this user paid more for higher-speed service," etc.
- The entire network acts as a distributed database; if a second user requests this same information, the request is routed toward where the local routing tables indicate the information can be found, as with an IP packet. However, if the information is encountered before the originating server is reached, the information can be returned. As all the objects are encrypted, there is little danger in returning the information as requested; the requestor must figure out how to unencrypt and use the data. Further, the encryption scheme can include some form of time and date stamp, so out-of-date information is discarded once a new version is available.
- Since information is being passed around, rather than packets, and each object is encrypted, the source and destination of the objects is pretty much meaningless (except in the case of a specific request and reply series).
- Since the source of the information is no longer really relevant in routing terms, this could place smaller information sources on an equal footing with larger ones.

There are, of course, many challenges to overcome in this kind of system as well. For instance:

- Network forwarding devices are not, today, designed to store and forward information in this way. Building systems able to store and forward information in this way would place a major burden on large-scale providers, who would need to rebuild their networks, and think about how to charge based

on the amount of data any particular user has requested, resulting in intermediate storage in their network. This could reshape the entire economy of the Internet by making it cheaper to always pull information from the network everyone else already wants. For instance, if you asked for a particular version of *Pleasant Valley Sunday*, the network might suggest another version, or even another song, which is already locally available, increasing the efficiency of the storage in the network. This process could squelch out less popular content in much the same way as the largely centralized content providers do today.

- It seems hard to understand how streaming services might work in this kind of network. Perhaps the best network available would be one with attributes of both the packet delivery systems and the kind of content-based networks the NDN contemplates.
- The performance of the network would seem to be difficult to understand or plan for. Information you are looking for might be close by or far away; even if it is close by, network devices might be bogged down servicing a lot of other requests, so they cannot service your request immediately. Quality of Service (QoS) would need to be completely rethought, down to the meaning of QoS itself, in this kind of network.

It does not seem as though NDN will become a commonly used technology, but it serves as a useful introduction to a very similar technology poised to have a large impact on the information technology world: blockchains.

Blockchains

To understand blockchains, you must begin with the hash. A hash is a simple concept that is quite difficult to implement in a useful way: a hash takes a string of numbers of any size and returns a fixed length number, or *hash*, (more or less) uniquely representing the original string. The simple-to-implement part is this: one rather naïve hash is to add the digits in a set of numbers until you reach a single digit, calling the result the hash. For instance:

```
23523
2 + 3 + 5 + 2 + 3 == 15
1 + 5 == 6
```

Hence, the number 23523 can be *represented* as 6. One curious property of the hash is there is no way to determine, from the hash, what the original number was—this is one of the essential observations of many uses for the hash. If I share a number with some third party, and that party then shares it with you, you can ask me for

the hash of the number (without telling me what the actual number is!), and you can verify the number you have is the same by verifying the hash that I give you matches the one you calculate.

The preceding hash is naïve because it is too easy to obtain a *collision*. In other words, there are many different sets of numbers that will result in a hash of 6 given the same process, such as 222, 33, 111111, and (probably) an almost infinite number of others. The tricky part of building a hash, then, is in ensuring collisions are rare or nonexistent.

Assuming you have developed such a hash (there are a number of them), you can then use hashes to build a *Merkle Tree*, as illustrated in Figure 30-4.

In Figure 30-4, four numbers have been processed through an algorithm to produce a hash: H1 through H4. H1 and H2 are, in turn, hashed to produce H5, and H3 and H4 are hashed to produce H6. H5 and H6 are, in turn, hashed to produce the root hash. There are a number of interesting things about Merkle trees; for instance, if you change the value of H1 for any reason, the value of the root hash also changes. Of course, this "just makes sense," but it means you can validate the contents of any group of files or values by examining a single value. Further, you can verify *which part* of the tree the change has taken place on if you have access to all the hashes in the tree *even though you do not know, or do not know if you can trust, the values themselves*.

To get to a blockchain, you string the Merkle tree out, as shown in Figure 30-5.

Here the hashes of H1 and H5 are hashed to form H2, the hashes of H2 and H6 are hashed to form H3, etc. What is interesting about a blockchain is that you can tell if any step has been repeated twice, if work has not been done, or if any of the numbers in the previous part of the chain have been changed—hence its usefulness in forming digital currencies.

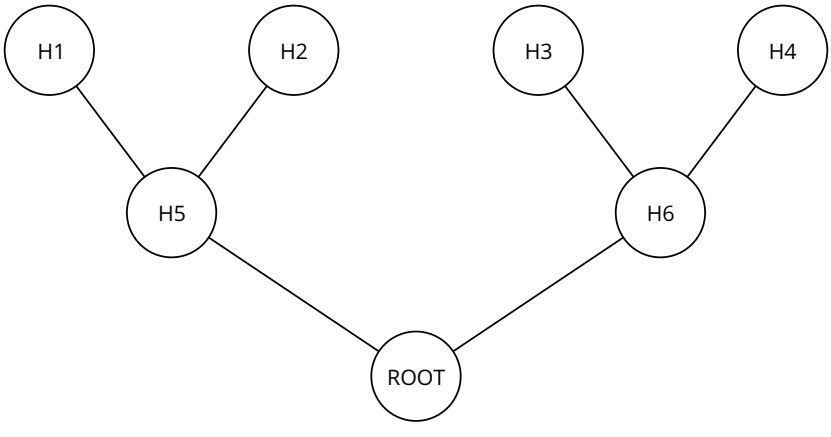


Figure 30-4 *A Merkle tree*

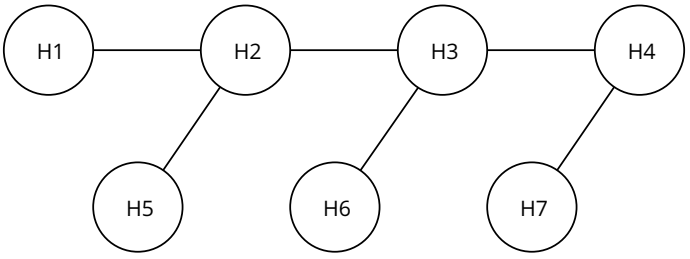


Figure 30-5 *A Merkel tree turned into a blockchain*

Note

There is, in fact, more to a real blockchain than this; there is also a consensus process. This description is a radical simplification.

Once you have a blockchain, what can you do with it? Remember the concept of the NDN described earlier? Now consider: what if every block on this blockchain were an object, as described in the NDN network? It should be possible to traverse the tree, using the information from the hash itself, to find the object you are looking for. Even if there is a newer version of the object, the older version should still exist, in its encrypted form, allowing you to compare every version of the object all the way back to its creation. There is no way to change any of the objects contained in the blockchain without invalidating the encryption on every object after this one has been modified.

Cryptocurrencies take advantage of these properties to allow users to place transactions on the blockchain across time. No transaction can be undone without invalidating the entire blockchain; there are many copies of the blockchain in existence, so a single copy being invalidated should cause the entire network of devices participating in the blockchain to quickly discard the invalidated copy.

Other blockchain systems, such as Ethereum, go beyond the idea of a cryptocurrency by allowing *executable code* to be stored in the blockchain alongside transactions. This means a virtual machine can be given an Ethereum blockchain that contains not only data to operate on (such as move some amount of money from one account to another account), but also some instructions about under what conditions the data should be acted on (when the receiver signs for the package). The operation could take place in full public view, but without information about the people involved, account numbers, etc., being exposed to the public view (because these can all be represented by hashes, instead of the real numbers, that can only be interpreted by the parties involved in the transaction).

A blockchain system like Ethereum could, in theory, provide an overlay on the entire public Internet, providing the same sort of system as the creators of the NDN originally conceived.

The Reshaping of the Internet

The Internet is, to most engineers, a constant. The protocols remain the same, and while the providers shift roles from time to time, or one provider buys another, there is very little apparent change in the Internet as a whole. This, however, is not a realistic view of the world. Figure 30-6 illustrates how the Internet has been built since the first few years of its commercialization.

This shape clearly puts the large-scale transit providers in a central role. The QoS and security protections offered by the transit providers regulate how quickly any user can send or receive traffic. If you want to start a new content or edge provider network or service, you can connect to the transit providers and reach pretty much everyone who connects to the Internet. What has been happening in the five years or so before this writing is a shift in the way content and edge providers are connected. The new connection pattern is illustrated in Figure 30-7.

The content providers have discovered a simple fact: the speed at which their content loads drives user engagement, and user engagement drives revenue. To make their pages load faster, the content providers need to be "closer" to their users. Being closer essentially means cutting out transit providers wherever possible and connecting directly to the edge providers. This means the global Internet is slowly moving away from being a mesh of peer networks to a more hub-and-spoke pattern, with large content providers in the hub, and edge providers acting as the spokes.

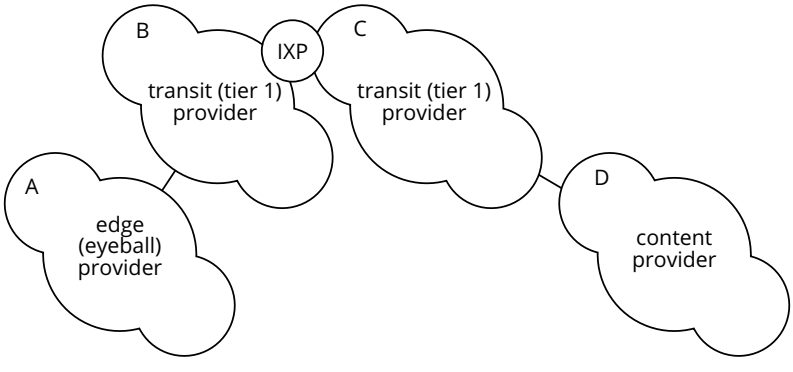


Figure 30-6 *The shape of the old Internet*

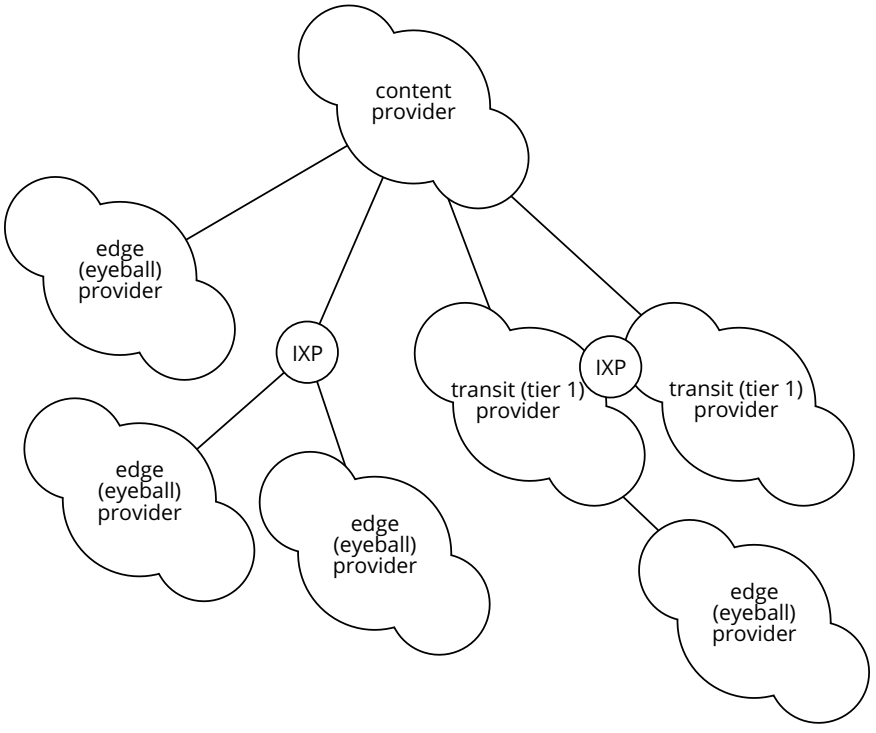


Figure 30-7 *The new shape of Internet connectivity*

There is little sense of what this means in the long term. For instance, it could mean

- The Internet will eventually fragment, with the content you can reach being determined by the edge provider you connect to (because not every edge provider will connect to every content provider).
- The transit providers could shrink, but not ultimately die off, allowing full connectivity, but with two classes of service; large content providers will be quickly reachable, while smaller and newer ones will be forced to take the slow path.

The second already appears to be happening. The ultimate effect of this "slow path/fast path" arrangement is that it becomes ever more difficult to start a new content service on the global Internet, which drives ever more power into a smaller group of players over time. Whether this trend will continue, or the ultimate end is healthy

for the Internet as a whole or the network engineering and larger information technology ecosystems reliant on the Internet, is hard to say at this point.

But this is certainly one of those trends worth factoring into any view of what the future of network engineering might look like.

Final Thoughts on the Future of Network Engineering

It often seems, in the present moment, like the world is changing too fast, there is no way to keep up, and the future of network engineering is bleak. There are some parts of the network engineering world for which this is likely true; old technologies do, ultimately, die, and others come to the front to take their place (or maybe the entire problem that the technology was designed to solve no longer exists for some reason). Through all of this, however, there will always be a need for well-trained, thoughtful engineers who understand the basic problems, and the scope of solutions available for those problems. For engineers who understand the technology at a more basic level, and hence can ask the right questions at the right time to make a difference in the way a business runs, there will always be a bright future in network engineering.

If you have read this far, studied the examples, and spent time thinking through the technologies as they have been presented here, you are at least starting on the road toward developing the skills needed to be one of those engineers who will always be in demand.

Further Reading

Bjorklund, Martin. *The YANG 1.1 Data Modeling Language*. Request for Comments 7950. RFC Editor, 2016. https://rfc-editor.org/rfc/rfc7950.txt.

Callon, Ross. *The Twelve Networking Truths*. Request for Comments 1925. RFC Editor, 1996. doi:10.17487/RFC1925.

"Ethereum Homestead Documentation." Accessed August 30, 2017. http://www.ethdocs.org/en/latest/.

Gates, Mark. *Blockchain: Ultimate Guide to Understanding Blockchain, Bitcoin, Cryptocurrencies, Smart Contracts and the Future of Money*. CreateSpace Independent Publishing Platform, 2017.

Huston, Geoff. "The Death of Transit?" *APNIC Blog*, October 28, 2016. https://blog.apnic.net/2016/10/28/the-death-of-transit/.

"HyperConverged.org." Accessed August 30, 2017. http://www.hyperconverged.org/.

Matsumoto, Craig. "Why Machine Learning Is Hard to Apply to Networking." Blog. *SDxCentral*, January 2, 2017. https://www.sdxcentral.com/articles/news/machine-learning-hard-apply-networking/2017/01/.

Theobald, Oliver. *Machine Learning for Absolute Beginners: A Plain English Introduction*. Independently published.

"What Is Ether." Accessed August 30, 2017. https://www.ethereum.org/ether.

White, Russ. "Death of Transit: A Need to Prevent Fragmentation." Accessed August 30, 2017. http://www.circleid.com/posts/20161107_death_of_transit_need_to_prevent_fragmentation/.

Zhang, Lixia, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D. Thornton, Diana K. Smetters, Beichuan Zhang, et al. "Named Data Networking (NDN) Project," October 31, 2010. http://named-data.net/techreport/TR001ndn-proj.pdf.

Review Questions

1. What kinds of network resources might be pooled like compute resources in a hyperconverged solution?
2. What is the difference between OpenConfig YANG models and the YANG models standardized by the IETF?
3. Review some of the challenges to implementing and deploying intent-based networking.
4. Where might machine learning be useful in network engineering?
5. What argument does the text use to explain why machine learning may never be used to configure networks?
6. What is the advantage of Named Data Networking over packet-based networks?
7. Research Ethereum. How might blockchains with embedded actions require routing?
8. Some engineers argue it is better to have a common modeling language, rather than a common set of models, for automation. What do you think their line of argument might be?

This page intentionally left blank

Index

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y–Z