

BROCADE



BROCADE IP PRIMER

FIRST EDITION

Everything you need to obtain a solid
foundation in networking technologies
and design concepts

JON FULLMER

BROCADE



BROCADE IP PRIMER

FIRST EDITION

**Everything you need to obtain a solid
foundation in networking technologies
and design concepts**

JON FULLMER

© 2010 Brocade Communications Systems, Inc. All Rights Reserved.

Brocade, the B-wing symbol, BigIron, DCFM, DCX, Fabric OS, FastIron, IronView, NetIron, SAN Health, ServerIron, Turbolron, and Wingspan are registered trademarks, and Brocade Assurance, Brocade NET Health, Brocade One, Extraordinary Networks, MyBrocade, and VCS are trademarks of Brocade Communications Systems, Inc., in the United States and/or in other countries. Other brands, products, or service names mentioned are or may be trademarks or service marks of their respective owners.

Notice: This document is for informational purposes only and does not set forth any warranty, expressed or implied, concerning any equipment, equipment feature, or service offered or to be offered by Brocade. Brocade reserves the right to make changes to this document at any time, without notice, and assumes no responsibility for its use. This informational document describes features that may not be currently available. Contact a Brocade sales office for information on feature and product availability. Export of technical data contained in this document may require an export license from the United States government.

Brocade Bookshelf Series designed by Josh Judd

IP Primer

Written by Jon Fullmer

Additional content by Steve Smith and Ahmad Zamer

Edited by Diana Alonzo

Design and Production by Victoria Thomas and Diana Alonzo

Based on content originally published as *Foundry Networks Certified Network Engineer Official Study Guide*.

Printing History

First Edition, October 2010

Important Notice

Use of this book constitutes consent to the following conditions. This book is supplied “**AS IS**” for informational purposes only, without warranty of any kind, expressed or implied, concerning any equipment, equipment feature, or service offered or to be offered by Brocade. Brocade reserves the right to make changes to this book at any time, without notice, and assumes no responsibility for its use. This informational document describes features that may not be currently available. Contact a Brocade sales office for information on feature and product availability. Export of technical data contained in this book may require an export license from the United States government.

Brocade Corporate Headquarters
San Jose, CA USA
T: 1-408-333-8000
info@brocade.com

Brocade European Headquarters
Geneva, Switzerland
T: +41-22-799-56-40
emea-info@brocade.com

Brocade Asia Pacific Headquarters
Singapore
T: +65-6538-4700
apac-info@brocade.com

Notice About Software Illustrations

Software related screen captures and code examples depicted in this book may vary from the version of Brocade software you are currently running. The concepts covered in this book can be applied regardless of the software version you are running.

About the Author

Jon Fullmer has held over 200 industry certifications, including FNCNE and CCNP. He has immersed himself in technology for over 25 years, focusing on various aspects from network design to assembly language programming. He has worked as a network designer and engineer for several companies, and has focused his expertise on Internet-facing infrastructures. Currently, he works as a Senior Infrastructure Engineer for the Family History department of The Church of Jesus Christ of Latter-Day Saints in Salt Lake City, Utah.

Jon currently Lives in Cottonwood Heights, Utah with his wife, Becky, and four children (Rachel, Jacob, James, and Joseph). He also lives with his collection of computers and other technology-related peripherals that he has accrued over the past 25 years. He lovingly refers to this collection as “the museum.”

Contents

Part One: Introduction to Networking	1
Chapter 1: Networking Basics	3
Why Do I Need a Network?	3
The OSI Reference Model	4
The Application Layer (Layer 7)	9
The Presentation Layer (Layer 6)	10
The Session Layer (Layer 5)	11
The Transport Layer (Layer 4)	13
The Network Layer (Layer 3)	16
The Data Link Layer (Layer 2)	17
Logical Link Control (LLC)	17
Media Access Control (MAC)	18
The Physical Layer (Layer 1)	19
Ethernet	20
Layer 1 Ethernet (Cabling)	21
Coaxial	21
Twisted Pair	22
Fiber	25
Hub	27
Switch	27
Layer 2 Ethernet (Addressing)	28
MAC Address	28
Ethernet Frame	29
Half-duplex vs. Full-duplex	30
Half-duplex	30
Full-duplex	31
Autonegotiation	31
Brocade Ethernet Switches & Routers	33
Summary	33
Chapter Review Questions	34
Answers to Review Questions	36

Chapter 2: TCP/IP	37
Layer 3 TCP/IP: IP	37
Thinking in Binary	39
Network Classes	41
The Subnet	45
How Does the Network Layer Know Where to Go?	47
CIDR	51
Private Networks (RFC 1918)	52
Layer 4 TCP/IP: TCP/UDP	53
Transmission Control Protocol (TCP)	56
The 3-Way Handshake	57
Windows and Acknowledgments	58
User Datagram Protocol (UDP)	59
Dynamic Host Configuration Protocol (DHCP)	60
Network Address Translation (NAT)	62
Port Address Translation (PAT)	64
Packet Capturing	65
sFlow (RFC 3176)	68
Summary	69
Chapter Review Questions	70
Answers to Review Questions	72
Chapter 3: Data Center Bridging (DCB)	75
Introduction	75
802.1Qbb: Priority-based Flow Control (PFC)	77
802.1Qaz: Enhanced Transmission Selection (ETS)	77
802.1Qau: Congestion Notification (QCN)	77
Making Ethernet Lossless	78
The Ethernet PAUSE Function	79
Priority-based Flow Control	80
Enhanced Transmission Selection	81
Data Center Bridge eXchange	83
Building the DCB Cloud	85
Congestion Notification	86
802.1Qau: Congestion Notification (QCN)	86
Summary	86
Chapter 4: TRILL—Adding Multi-Pathing to Layer 2 Networks	89
Why do we need it?	89
Introducing TRILL	92
The TRILL Protocol	93
TRILL Encapsulation	93
Link-State Protocols	94
Routing Bridges	94
Moving TRILL Data	94
Summary	96

Chapter 5: Load Balancing Basics	97
The Virtual IP (VIP) Address	97
Load Balancing Methods	98
Proxy	98
Layer 4 Switching	99
Layer 7 Switching	99
Server Load Balancing (SLB)	100
Source NAT	101
Direct Server Return (DSR)	102
The Health Check	103
Layer 3	104
Layer 4	105
Layer 7	107
Redundancy	108
Summary	110
Chapter Review Questions	111
Answers to Review Questions	114
 Part Two: Switching	 117
Chapter 6: The Brocade CLI	119
Switch Layout	119
Chassis	119
Stackable	120
Accessing the CLI	120
Serial	120
Network	121
Online Help	122
Configuring by Context	124
The config	125
Software (and where it's stored)	127
show commands	131
clear commands	133
Greetings	134
Configuring the Interface	135
Ping	136
Filters and Regular Expressions	138
User Access Control	141
Usernames and Passwords	141
Authentication, Authorization, and Accounting (AAA)	145
Simple Network Management Protocol (SNMP)	147
Remote Monitoring (RMON)	149
The Web UI	150
Restricting or Disabling Access	152
Summary	153
Chapter Review Questions	154
Answers to Review Questions	156

Chapter 7: Link Aggregation (Trunks)	159
Static Trunks	160
Switch Trunks	160
Server Trunks	160
Dynamic Link Aggregation (IEEE 802.3ad)	161
Configuring Trunks	162
Switch Trunks	163
Server Trunks	163
IEEE 802.3ad (LACP) Trunks	163
show Commands	164
Summary	166
Chapter Review Questions	166
Answers to Review Questions	168
Chapter 8: Virtual Local Area Network (VLAN)	169
What is a VLAN?	169
802.1q Tagging	172
Configuring VLANs	173
The Default VLAN	180
Dual Port Mode	181
Configuring a Layer 3 VLAN	183
VLAN Groups	187
Summary	188
Chapter Review Questions	189
Answers to Review Questions	190
Chapter 9: Spanning Tree Protocol	193
What is Spanning Tree Protocol?	193
IEEE 802.1d (Spanning Tree Protocol)	195
Bridge Protocol Data Unit (BPDU)	195
The Root Switch	196
Convergence	199
STP Commands	200
Speeding Up Convergence	202
IEEE 802.1w: When IEEE 802.1d	
Just Isn't Fast Enough	204
RSTP vs. STP	204
Configuring RSTP	206
802.1w show Commands	206
IEEE 802.1s (MST)	207
STP Load Sharing	209
Topology Groups	210
Virtual Switch Redundancy Protocol (VSRP)	211
Summary	213
Chapter Review Questions	214
Answers to Review Questions	216

Part Three: Layer 3 Switching (Routing)	217
Chapter 10: Routing Basics	219
Disabling Layer 2	219
The Routing Table	220
Connected Routes	221
Static Routes	222
Dynamic Routing Protocols	227
Administrative Distance	230
Routing Information Protocol (RIP)	231
Loop Avoidance	232
RIP version 1 vs. RIP version 2	233
Configuring RIP	234
The “Null” Route	235
Summary	235
Chapter Review Questions	236
Answers to Review Questions	238
Chapter 11: Open Shortest Path First (OSPF)	239
Link State vs. Distance Vector (OSPF vs. RIP)	239
Getting to Know Your Neighbors	240
Hello Protocol	241
Designated Router (DR)	241
The Designated Router Election	241
Neighbor States	242
Areas	244
Area 0 - The Backbone	245
Area Border Router	245
Autonomous Systems (AS)	246
Autonomous System Boundary Router (ASBR)	246
Getting RIP and OSPF To Talk To Each Other	246
Special Areas	248
Stub Area	248
Not So Stubby Area (NSSA)	248
Totally Stubby Area	249
Link State Advertisement (LSA)	249
Configuring OSPF	250
Loopback Interfaces	250
Cost	251
Adjusting OSPF Timers	252
Advertising a Default Route	253
Configuring Stubby Areas	254
Configuring a Virtual Link	255
OSPF show Commands	256
Summary	257
Chapter Review Questions	258
Answers to Review Questions	260

Chapter 12: Multi Protocol Label Switching (MPLS)	261
What is MPLS?	261
MPLS Concepts	261
MPLS Operations	262
Configuring MPLS Basics on Brocade	263
MPLS Virtual Leased Lines (VLL)	264
How Virtual Leased Lines work	264
Configuring VLL on Brocade	265
MPLS Virtual Private LAN Segment (VPLS)	266
Configuring VPLS on Brocade	266
MPLS Layer 3 VPNs	267
Configuring MPLS Layer 3 VPNs	269
Summary	270
Chapter 13: Border Gateway Protocol (BGP)	271
A Routing Protocol for the Internet	271
What is BGP?	272
IGP vs. EGP	272
eBGP	273
iBGP	273
The Autonomous System (AS)	274
Stub AS	276
Neighbor Peering	277
Peering States	278
Configuring BGP	279
Configuring Neighbors (Peers)	279
Advertising Routes	280
Loopback Interfaces (iBGP)	280
iBGP - next-hop-self	281
eBGP - multihop	282
BGP show Commands	284
Summary	285
Chapter Review Questions	286
Answers to Review Questions	287
Chapter 14: Security, Redundancy and More	289
Security: Access Control Lists (ACL)	289
Subnet Mask vs. Wildcard (or Inverse) Mask	289
Numbered Access Lists	291
Named Access Lists	293
Creating Access Lists	294
Applying Access Lists	299
Strict Mode	301
ACL Flow Counters	302
MAC Filters	303
Access Control Lists: Not Just For Security	304
Policy-Based Routing (PBR)	304
Network Address Translation (NAT)	307

Redundancy: Virtual Router Redundancy Protocol (VRRP)	310
What is VRRP?	310
Configuring VRRP	311
VRRP vs. VRRP-Extended	314
Configuring VRRP-Extended	314
Traffic Shaping: Quality of Service (QoS)	316
Why Quality of Service?	316
Configuring QoS	317
Summary	317
Chapter Review Questions	319
Answers to Review Questions	321
 Part Four: Layer 4-7 Switching (Load Balancing)	323
Chapter 15: Server Load Balancing (SLB)	325
The Brocade ServerIron	325
Overview of Features	325
The Application Switch Management Module (ASM)	326
The Switch Fabric Management Module	326
The Management Module	327
Interface Modules	327
Attack Protection	328
Predictors	332
Configuring SLB	334
Configuring “Real” Servers	334
Configuring Virtual Servers	335
Configuring Health Checks	336
SLB show Commands	339
State of Real Server	340
State of Real Server Ports	340
Configuring Source NAT	342
Configuring Direct Server Return (DSR)	343
Configuring Multiple VIPs to the Same Real Server	344
Hot-Standby Redundancy	345
Symmetric Server Load Balancing	347
Active/Standby	347
Active/Active	348
Remote Servers	349
Primary and Backup Servers	349
Clones	350
Summary	350
Chapter Review Questions	352
Answers to Review Questions	354

Chapter 16: Session Persistence and	
Transparent Cache Switching	355
Policy-based Server Load Balancing (PBSLB)	355
Session Persistence	357
Concurrent	358
Sticky	358
Configurable TCP/UDP Application Groups	359
Cookies	359
SSL ID Based Persistence	362
URL Switching	363
What is Transparent Cache Switching?	365
Configuring Transparent Cache Switching	366
Enabling TCS	366
Defining Cache Servers	367
Cache-Groups	367
Policy-based Caching	368
Summary	369
Chapter Review Questions	370
Answers to Review Questions	372
Chapter 17: Firewall Load Balancing (FWLB)	373
Why Would I Want to Load Balance Firewalls?	373
Configuring FWLB	375
Defining the Firewall Servers	376
Defining the Firewall Group	376
Assigning the Firewall Servers to a Firewall Group	376
Configure Firewall Paths	376
Configure Static MAC Entries For Each Firewall Server	378
FWLB Predictors	378
FWLB show Commands	379
Fine-Tuning FWLB	380
Stateful FWLB	381
Summary	382
Chapter Review Questions	383
Answers to Review Questions	384
Chapter 18: Global Server Load Balancing (GSLB)	387
What is Global Server Load Balancing?	387
Domain Name Service (DNS)	387
DNS Hierarchy	388
DNS Domains/DNS Zones	390
The A Record	391
Enter: Global Server Load Balancing (GSLB)	391
How Does GSLB Decide?	392
Check 1: Server Health	392
Check 2: Session Capacity Threshold	393
Check 3: Round-Trip Time (RTT)	393
Check 4: Geographic Location of the Server	394

Check 5: Available Session Capacity	394
Check 6: Flashback	395
Check 7a: Least Response Selection	395
Check 7b: Round-Robin Selection	395
Affinity	395
Administrative Preference	396
Configuring GSLB	396
Fine-Tuning DNS Parameters	398
GSLB show Commands	399
Summary	400
Chapter Review Questions	401
Answers to Review Questions	402
Glossary	405

Part One: Introduction to Networking

The following chapters are included in Part One:

- [“Chapter 1: Networking Basics”](#) starting on page 3
- [“Chapter 2: TCP/IP”](#) starting on page 37
- [“Chapter 3: Data Center Bridging \(DCB\)”](#) starting on page 75
- [“Chapter 4: TRILL—Adding Multi-Pathing to Layer 2 Networks”](#) starting on page 89
- [“Chapter 5: Load Balancing Basics”](#) starting on page 97



Networking Basics

The world of computers has changed faster than many people dared dream. Scanning over the changes of the last four decades reveals an unfathomable and exponential growth in technology. Computers have become significantly smaller and faster. Yet, the progress of computing technology seems to have reached the same conclusion that the progress of the human race in general has reached: *two heads are better than one*.

Why Do I Need a Network?

Over five centuries ago, John Donne penned the phrase: “No man is an island.” In my experience, that applies to computers as well as mankind. Think of how you use your computer every day. Do you check your e-mail? Look at a Web page? Download some software? Chat with a friend over Instant Messaging? What do all these examples have in common? Your computer must communicate and cooperate with other computers, and this allows you to communicate with other people.

In the early days of computing, networking wasn't the focus. Manufacturers created powerful machines but focused more on having multiple users sharing the same machine's resources. Is your computer not powerful enough to handle the number of users you need? Get a more powerful computer!

Well, as you can imagine, this method can only take you so far. It didn't take too long for people to realize that having multiple computers, and allowing them to communicate with each other, would yield a myriad of benefits. For one, remote locations could now quickly share ideas. A research lab at UCLA could easily collaborate with another research lab in New York City. Two, data could be easily shared with friends, families, and co-workers. And three, depending on the network, you could even combine the computing power of many computers to act as a single, very powerful computer.

But what about the small business who just needs a computer to keep the books? There was certainly a time when it was common for a computer to serve just that simple a function. The early 1980's saw many home computers enter the business world to keep track of sales, accounting, etc. Could a computer be used that was *not* part of a network? Sure, but its functionality would only take it so far. Even the smallest of businesses today use their computers

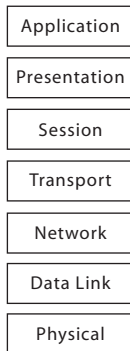
for far more than just “keeping the books.” They interface directly with their bank accounts and suppliers. They communicate with customers and partners. So much can be gained when machines work together.

When the concept of networking first began, computers could only communicate to each other if they were manufactured by the same company. This was because the computer manufacturers took it upon themselves to create every aspect of the communication between the machines, and every manufacturer had a different idea of the way computers should communicate to each other. In the 1970's, the microcomputer entered the arena. Soon, there would be many different brands of computers. In particular, there would be a large number of less-powerful computers than their mainframe relatives. This would bring computers to the people, and these people would want their computers to be able to talk to each other. But how?

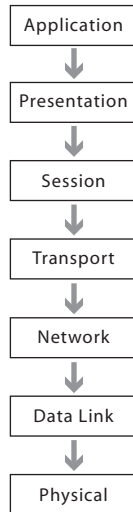
The OSI Reference Model

In 1982, the International Organization for Standardization formed the Open Systems Interconnection (OSI) initiative. The idea was to create a set of standards for computers to use when communicating to each other. Creating an open standard would allow computers made by different manufacturers to more easily communicate with one another. The result of the initiative is referred to as The OSI Reference Model.

The OSI Reference Model describes a “protocol stack” (or a method for communication) that consists of seven layers:



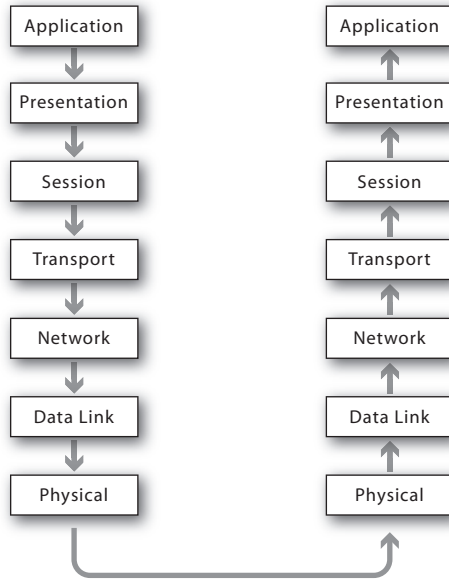
Each of these layers represents a piece of the process. They each work together to achieve communication. Starting at the Application Layer, communication flows this way:



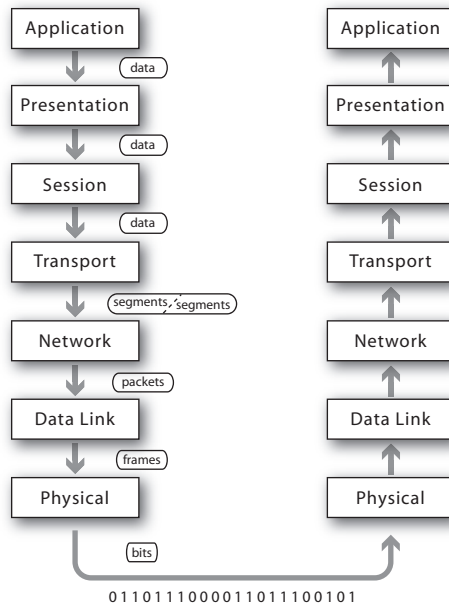
Notice that each layer only needs to be aware of its neighbor. This allows a great deal of freedom. While a single manufacturer could certainly write a protocol stack that handles the work of all seven layers, they don't need to.

Also, you can see from the diagram that communication flows from one layer to the next. This means that, for any given layer, they only need to be aware of their neighbors. For example, the Presentation Layer (Layer 6) does not need to know which Data Link (Layer 2) technology you're using. It only needs to know how to deal with the data that the Application Layer sent it, and it needs to know how to interact with the Session Layer (Layer 5). Likewise, the Transport Layer needs to know how to interact with the Session Layer (Layer 5) and the Network Layer (Layer 3). It never has to interact with any other layer.

We know that data travels down through the layers to the Physical Layer (Layer 1), but then what happens? Now, the data is sent to the remote computer. But what does the remote computer do with it? Once again, it travels through the layers of the OSI Reference Model, but this time, it travels in reverse.

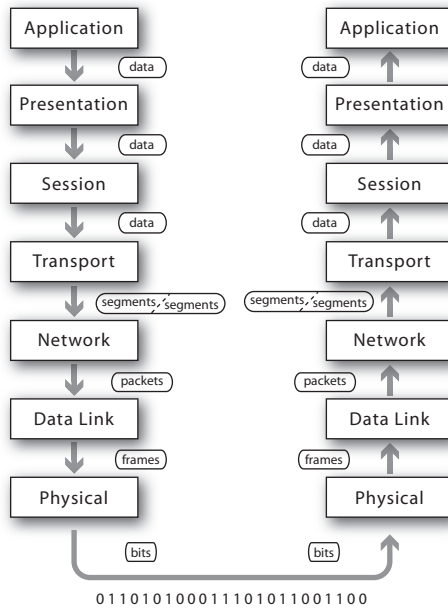


It is very important to note that each layer of the OSI Reference Model on the initiating computer communicates only with the corresponding layer on the remote computer. For example, the Transport Layer (Layer 4) on the initiating computer “speaks” only to the Transport Layer of the remote computer. It never speaks to any other layer on the remote computer.



One of the ways this is accomplished is by encapsulating your layer at each pass. For example:

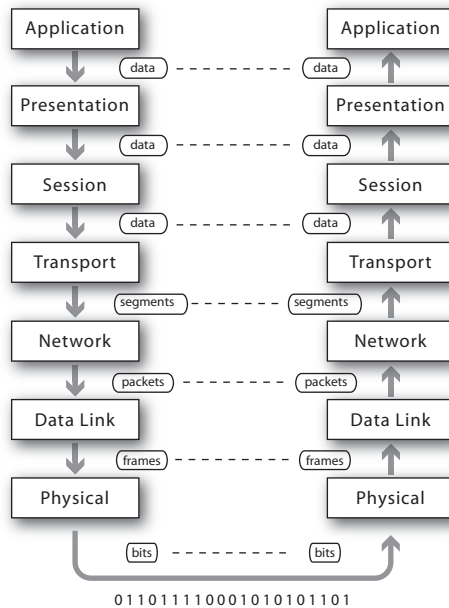
- The Application Layer prepares its data and hands off to the Presentation Layer
- The Presentation Layer formats the data in a way that conforms to its protocol
- The Session Layer adds a label to the data that identifies the “conversation”
- The Transport Layer breaks the data into *segments*, tags each segment with an order number and, perhaps, a checksum
- The Network Layer takes the *segments* and creates *packets* that contain information on the source and destination of the machines that are communicating
- The Data Link Layer takes the *packets* and creates *frames* that are compatible with whatever physical medium it may be using
- The Physical Layer receives the *frames*, converts them to *bits* (binary digits; more on this later) and physically sends the information



Now the data is being sent to the remote computer. On the remote computer's side, we see:

- The Physical Layer receives the binary signals (*bits*) that were sent, and hands them, as *frames*, to the Data Link Layer
- The Data Link Layer recognizes and processes the *frame* information, and hands the data, as *packets*, to the Network Layer
- The Network Layer recognizes and processes the *packet* information, and hands the data, as *segments*, to the Transport Layer
- The Transport Layer recognizes and processes the *segments*, and passes the data on to the Session Layer
- The Session Layer recognizes the “conversation” tags, verifies that they pertain to the proper conversation, and passes the data on to the Presentation Layer
- The Presentation Layer recognizes the format of the information it receives, and passes the data on to the Application Layer
- The Application Layer receives the data, and carries out whatever process is dictated by the protocol

This kind of communication is often diagrammed in this way:



Here you can see how the data flows, but you can also see how each layer only communicates with its own layer. Physical Layer communicates with Physical Layer. Data Link Layer communicates with Data Link Layer, and so forth.

Now that we know how the communication flows, let's get to know the individual layers a little better.

The Application Layer (Layer 7)

This layer is one of the most recognized layers in all of the OSI Reference Model. Why? It's the layer that communicates with the *users*. Remember that humans use these machines. "Computers talking to other computers" is a great notion, but if, in the end, they don't make it possible for humans to share information with other humans, it's pretty pointless.

The Application Layer is where the data is created. Let's say, for example, that you want to send an e-mail. You would use an e-mail program to create an e-mail message and send it. The program takes that data (the e-mail message), and uses a Layer 7 protocol called Simple Mail Transfer Protocol (SMTP) to send the message. It will take its journey down the other six layers and back up again, but the Application Layer is SMTP.

There are lots of other well-known Application Layer protocols on the Internet. Here are a few more examples:

- **HyperText Transfer Protocol (HTTP)** for viewing web pages
- **File Transfer Protocol (FTP)** for sending and receiving files
- **Secure Shell (SSH)** for remotely accessing a computer's command line
- **Domain Name System (DNS)** for resolving names to IP addresses (and vice versa)

Some engineers will joke that the OSI Reference Model actually has an *eighth* layer: The User Layer. In a way, this actually does work with the model. The user creates the data and submits it to the Application Layer, and it goes on down the chain. If you ever hear an engineer or administrator refer to a “Layer 8 problem,” it means that the problem may have been caused by a user.

There are many, many programs that are written just to make it easier for the user to make good use of a Layer 7 protocol. Many of these are ones that you probably use every day. They include web browsers, e-mail programs, terminal clients (e.g., Telnet, SSH, etc.), Instant Messaging clients, and on and on. The list would be too long (and too boring) to list here. But now that you know how to recognize Layer 7, let's move on to Layer 6.

The Presentation Layer (Layer 6)

People often forget about this layer. Truth be told, you will probably do very little troubleshooting (as an engineer or administrator) in this layer. Its name describes its function very well. It *presents*. The Presentation Layer's job is to take the data from the Application Layer and format it according to a standard.

The Presentation Layer is like a printing office. Let's say you want to make some invitations. You've written out on a piece of paper what you want the invitation to say. You might have even hand-drawn how you might like the information to be arranged on the invitation. Next, the printing office takes that information, selects appropriate fonts, layouts, and the best card stock, and creates a classy-looking invitation. In other words, the printing office takes your information, and makes it presentable. This is just like the Presentation Layer. It takes information from the Application Layer and presents it in its proper format.

You interact with Layer 6 protocols every day (more than you might think). Here are some examples:

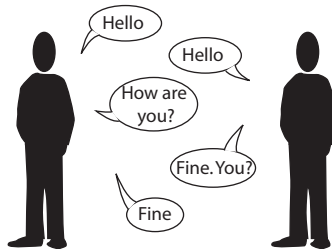
- **American Standard Code for Information Interchange (ASCII)** is a very common method for formatting text; it assigns a number to every letter, number, and symbol it displays
- **Joint Photographic Experts Group (JPEG)** is a method for presenting images, photographs, etc.
- **Musical Instrument Digital Interface (MIDI)** is a method for storing digital signals from a synthesizer or other electronic musical instruments
- **Moving Pictures Experts Group (MPEG)** describes a series of compression methods for presenting audio and video

Remember, it's all about the presentation in Layer 6. What does the data look like? How is the information arranged? How is it presented?

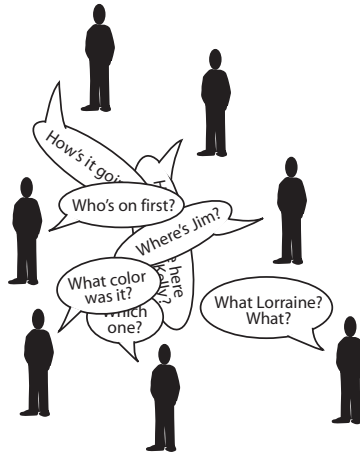
The Session Layer (Layer 5)

Now we're ready to start talking. We know *what* we're saying (Application Layer). We know *how* we're saying it (Presentation Layer). Now we need to "open a dialog." We have to start a conversation. The Session Layer's job is keeping track of conversations.

Computers have become better and better at multitasking. Think about having a conversation with a person (we're back in the human world now). You speak. The person listens. The person responds. You listen.



Now, imagine you wanted to have a second conversation with the same person *at the same time*. In fact, maybe you even want to have additional conversations with other people in the same room *at the same time*.



In the human world, that would be a near impossible feat. In the computer world, however, it happens all the time. You may find times when you are using a web browser to view a page, and you need to bring up another web browser (or maybe a new browser window or tab within the same web browser) to view a different page at the same time. You're having two different conversations or *sessions*.

The Session Layer's job is to create, maintain, and terminate conversations or sessions. It also decides how the conversation will occur. Some sessions will be *full duplex* (conversations in which you can listen and talk at the same time), and some sessions will be *half duplex* (conversations where one side speaks, while the other listens; the listener cannot speak until the speaker stops to listen). Full duplex is like talking on the telephone. Both sides can talk and listen whenever they want to. Half duplex is more like talking on a CB radio or walkie-talkie. The person talking holds down a button, speaks, and then lets go of the button when they are done speaking. While the button is being held down, the speaker can't hear anyone else.

There is a function of the TCP layer in TCP/IP that handles much of the Session Layer functions. All in all, the Session Layer is not clearly defined within the TCP/IP suite. There are a few protocols that some feel belong to this layer. They include:

- **Remote Procedure Call (RPC)** makes calls to other services running on the server depending on the type of conversation that the client requested
- **Structured Query Language (SQL)** was developed by IBM as a method to request and receive information; commonly used in database applications
- **Network File System (NFS)** was developed by Sun Microsystems; a method to share a local filesystem over the network

So, when you think Session Layer, remember “conversations.” You’re starting to talk. You’ve got your information (Application Layer). You’ve made it pretty (Presentation Layer). Now, you’re ready to open your mouth and speak (Session Layer).

The Transport Layer (Layer 4)

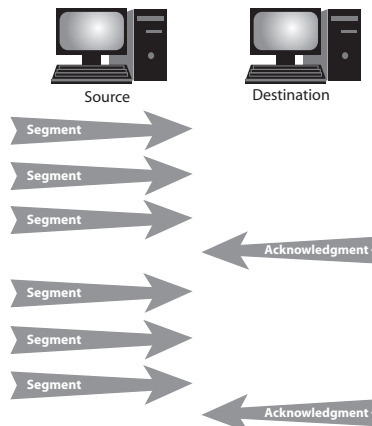
We’re ready to start talking. Now we need to make sure that our data is going to get to where it needs to go (*reliability*). Not only do we need to make sure it gets there, but we need to make sure it arrives the way we meant it to arrive (*integrity*). This is the job of the Transport Layer.

The Transport Layer receives the data from the Session Layer and breaks it up into manageable pieces called *segments*. Or, if you’re the receiving computer, the Transport Layer takes the *segments* (sent by the sending computer) and reassembles them (in proper order). Then, it hands the finished data to the Session Layer.

Typically, the Transport Layer does its job in one of two ways: *connection-oriented* and *connectionless*. In connection-oriented transporting, it’s all about reliability. The Transport Layer:

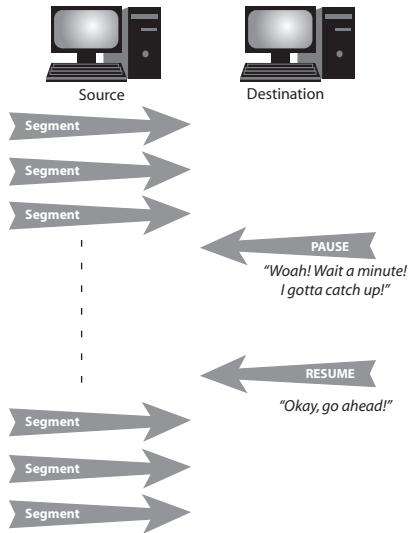
- Makes sure the receiver gets all the segments
- Makes sure the segments aren’t sent too fast for the receiver to handle
- Makes sure the segments are reassembled in proper sequence

Connection-oriented transport uses *windows* and *acknowledgments* to make sure the segments get to the other side okay. The Transport Layer will begin sending segments, but then, after a few segments it will wait for the other side to send an acknowledgment. The acknowledgment message is just like the receiver calling back, and saying, “I got it!” The window is how many segments will be sent before an acknowledgment must be sent. Here’s an example of a window size of three segments:



As you can see, the sender sends three segments and waits for the acknowledgment from the receiver. The receiver sends the acknowledgment. Now, the sender can send three more segments, and so on.

Connection-oriented transport also uses something called *flow control*. The idea here is that the sender wants to make sure he's not overwhelming the receiver with information. Let's say you have a super-fast server that needs to send information to a 15-year-old workstation (don't think this won't come up). The workstation probably doesn't have that fast a network connection, and its processor won't be able to handle information nearly as quickly as the server. This is where the Transport Layer comes to the rescue.

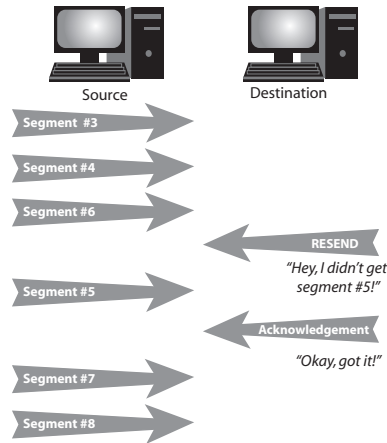


The receiver is able to send a message back to the sender saying, "Whoa! Wait a minute! Let me catch up! Okay, now I'm ready. Go ahead." This is one more mechanism that connection-oriented transporting uses to make sure that the data arrives reliably.

One other mechanism used by connection-oriented transporting is labeling the segment with an order number. Seems simple, doesn't it? It breaks up the data into segments, but it's careful to label each segment ("This is the first segment...this is the second segment...", etc.). Why would it need to do this? Well, it's always possible that something may happen on the way to the receiver that may put the segments out of order. If that happens, the receiver's Transport Layer will simply look at the segment labels and reassemble them in the correct order.

But what if a segment is missing? How will the receiver know? There will be a break in the order. The receiver will be reassembling the segments, and may notice, "Hey, I've got segment 1, 2, 3, 4, 5, and 7, but I don't have 6." At this

point, the receiver will send back a message to the sender to say, "Hey, I didn't get segment number 6." The sender will then retransmit segment number 6. The receiver will acknowledge it, and the transport will continue.



I've talked a lot about connection-oriented transporting. What about connectionless? Connectionless transporting will break the data into segments as well, but it doesn't really care whether the data gets to where it's going or not. It just sends it. There's no window. No acknowledgments. No flow control. No retransmitting. The data is sent, and whether it actually gets to the destination or not is not important.

"What kind of a stupid transport method is that?" I hear you cry. It seems undesirable on the surface, doesn't it? I mean, what's the point of the Transport Layer, if you can't make sure your data got there, right? One word: speed.

The problem with connection-oriented transporting is there's a lot of overhead. There are acknowledgments to wait for, window sizes to maintain, flow control to be conscious of, retransmits—it's a lot of extra work (and a lot of extra bandwidth). Overhead slows down the transmission. Now, if you're sending e-mail, you're typically more willing to sacrifice the small amount of speed difference to make sure that your e-mail got there okay (and the way you wanted it). But are there communications where it would actually be better for you to sacrifice reliability for speed?

Think of a sound file. One minute of CD-quality music contains approximately 10 MB of data. You're thinking of using connectionless transport for speed. You know the network well enough to know that you have a small chance of losing 0.001% of your segments, should you use connectionless transporting. That amounts to approximately 100 bytes of data lost. Now, I have audiophile friends who would argue that they could tell the difference between the full 10

MB sound and the 100 bytes lost. In reality, it would be extremely difficult for the human ear to perceive the difference. But this loss allowed you to be able to stream the audio to your receiver in real time.

Video streaming uses connectionless transport, too. Video uses even more data to show its image (and sound). If we have any hope of streaming video and audio (for, say, video conferencing), speed is much more important than reliability.

The most well-known Transport Layer protocols are Transmission Control Protocol (TCP), which is connection-oriented, and User Datagram Protocol (UDP), which is connectionless. Both of these protocols are part of the TCP/IP suite.

The Network Layer (Layer 3)

So, “we’re off to see the wizard,” to quote Dorothy from *The Wizard of Oz*, but we’re still missing an important piece. How are we going to get there? If we were in Oz, we could simply follow the yellow brick road. In the OSI Reference Model, we call it the Network Layer.

The Network Layer’s job is to decide how to get there. How does the data get from the sender to the receiver? That’s the question that the Network Layer has to resolve. It does this by using *addressing*, *routing protocols*, and *networks*.

Addressing is simply assigning a number to every object you’d want to communicate with. There are even objects called gateways or routers that will help you connect to remote addresses. These are Network Layer technologies.

Routing protocols are used between routers to help them do their job. Most routing protocols will tell all of the other routers when a particular path is no longer available (let’s say someone tripped over a network cable). Some routing protocols will tell other routers when paths become congested. The idea of routing protocols is that they are used by routers to help all of the other routers do their job: get the data to where it needs to go.

Networks, in this context, are groups of addresses. We’ll dive much further into this pool in the next chapter. For now, just know that addresses are grouped into *networks*.

Some people think of the Network Layer as the post office. The post office receives mail, sorts it, labels it with a postal meter (which contains information on where the letter is supposed to go, and how it will get there), and sends it on its way.

The Network Layer is similar. It receives *segments* from the Transport Layer, and it adds its own little pieces of information (including addressing, and the path it will take to get to its destination). This new creation is called a *packet*. Once the packet is created, it is sent to the Data Link Layer to continue the process.

By far, the most well-known Network Layer protocol is the Internet Protocol (IP). This is the IP portion of the TCP/IP protocol suite. Well-known routing protocols include Routing Information Protocol (RIP), Open Shortest Path First (OSPF), and Border Gateway Protocol (BGP), all of which we'll be talking more about in future chapters.

The Data Link Layer (Layer 2)

We're getting closer now. We have the data (Application Layer). It's pretty (Presentation Layer). We've set up our conversation (Session Layer). We'll make sure the conversation will be heard (Transport Layer). And we know how we're going to get there (Network Layer). What's next?

The Data Link Layer is the layer where the data gets packed up and ready to go. We've received the packet from the Network Layer, so we know how it's going to get there. One of the Data Link Layer's jobs is to make sure the *data gets to the physical devices* it needs to, in order to follow the Network Layer's instructions.

In a way, the Data Link Layer is similar to the Network Layer. It, too, has to determine how its data (*frames*) will get to where they need to go. But the Data Link Layer's communication is local. Think of the telephone system. Let's say a person in Los Angeles is calling someone in New York. This call will go from the person's phone to a switch, to another switch, to another switch, and so on, and so forth, until it reaches the phone of the person in New York. The Network Layer's job would be to get the call from Los Angeles to New York. The Data Link Layer's job would be to get the call from the phone to the first switch. Then, there would be a new Data Link Layer job to get the call from the first switch to the next switch, and so on. The Network Layer will take you anywhere you need to go. The Data Link Layer just takes you to the next step. It is local.

To do its job, the Data Link Layer is divided into two sublayers: the Logical Link Control (LLC) Layer and the Media Access Control (MAC) Layer.

Logical Link Control (LLC)

This sublayer has two jobs:

- Detection and retransmission of dropped frames
- Protocol multiplexing

The first job, detection and retransmission, is just as it sounds. The LLC must detect when frames are dropped, and then retransmit them. Protocol multiplexing sounds fancy, but it's actually very simple. This is where the frame identifies what Network Layer protocol it received information from. For example, if the Network Layer protocol were IPX, the Logical Link Control sublayer would identify that it received the packet from IPX (on the Network Layer). On the receiving side, the Logical Link Control sublayer will demultiplex the frame, and realize that it should be handed to the IPX protocol at the Network Layer for further processing. Think of it as a name-stamp to make sure that the Net-

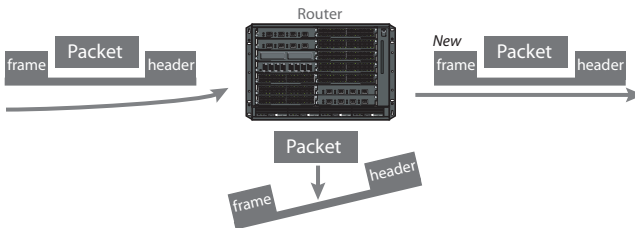
work Layer protocol sending the packet is the same Network Layer protocol receiving it. After all, you wouldn't want an IP packet processed by IPX on the receiving side, would you?

Media Access Control (MAC)

This sublayer introduces its own addressing method, similar to the Network Layer. But where the Network Layer's addressing is considered logical, the Data Link Layer is considered physical. Every device on the network has some piece of hardware that allows it to communicate to the network called a Network Interface Controller (NIC). For most hardware, that NIC has a Data Link Layer address within its hardware. It's like a serial number imprinted by the manufacturer. It's not as permanent as a serial number, and there are ways in most operating systems to temporarily override and change the number. In most circumstances, you won't. This is the NIC's physical address. The logical address (the Network Layer address) is assigned and can be easily changed. It is the physical address that the Data Link Layer uses to carry out the Network Layer's orders.

Why two addresses? Remember that the Data Link Layer is local. The physical address stays within the local network. It does not go outside the boundary of the *broadcast domain* (more on this later). The logical address spans all boundaries. It is meant to traverse many broadcast domains, networks, and so on. Think of the physical address as similar to the last seven digits of a telephone number (in the United States). If you're making a local call, you usually only need to dial seven numbers. But what if you want to call someone in a different state, or a different country? You need additional numbers to get you there (area codes, country prefixes, etc.). You need a higher level layer, a logical layer. Networking works the same way.

The Data Link Layer takes the *packet* from the Network Layer, and creates a *frame*. The frame includes the source physical address (of the sender) and the destination physical address of the next physical device in the path to get to the logical address destination. What happens if a packet has to travel through several networks to get to its destination? At each physical device, that passes the packet from network to network, the frame information is dropped and new frame information is created. The packet itself remains unaltered. I think this could be best summed up with another picture.



One other feature in a lot of Data Link Layer protocols is error correction. There are a lot of physical media to choose from (as we will discuss more in the next segment). Some media are very reliable. Some are not. A Data Link Layer protocol does well to know what the next layer is. In many cases, it will anticipate the unreliability and employ error correction to compensate.

Of all the Data Link Layer protocols there are, clearly Ethernet is the one that is best known. It is an extremely common protocol, and we'll be discussing it in more detail later in this chapter. Some other examples include Token Ring, Asynchronous Transfer Mode (ATM), and High-Level Data Link Control (HDLC).

The Physical Layer (Layer 1)

Well, we made it. After all the work that's been done in the upper six layers, the finished frame is handed from the Data Link Layer to the Physical Layer. The Physical Layer is the layer that does the *actual transmitting of data*. This is the layer where the data actually moves. You can think of the upper six layers as "getting the data ready." When it reaches Layer 1, "it goes."

The Physical Layer takes the *frame* from the Data Link Layer and converts it to *bits*. We'll go into this much more in Chapter 2, but for now, all you need to know is that it's a long stream of 1's and 0's.

0 1 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 0 0 0 0 1 0 1 0 1 1 1

Once converted to bits, the Physical Layer uses its physical media to convey the data. There are a lot of different ways to do this. Some physical protocols send electrical signals across copper wire. Some physical protocols send a laser signal across fiber optics. Some protocols generate pulses of microwaves into the air. Each different method has their advantages and disadvantages, but they all get the job done.

Once the Physical Layer has sent its bits, the receiving Physical Layer picks them up, shapes them into a frame, hands it off to the receiver's Data Link Layer, and we start the journey back up the stack. The Physical Layer has done its job. It's the layer that actually sends the data.

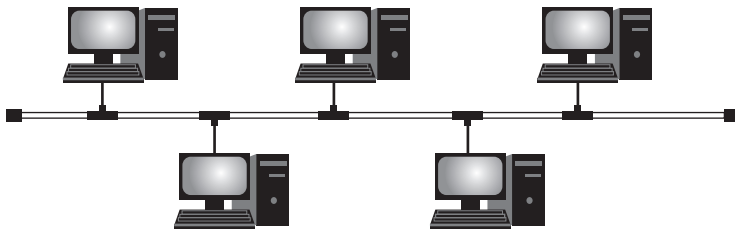
Examples of Physical Layer protocols include Unshielded Twisted Pair (UTP), Fiber, IEEE 802.11 (wireless networking), and so many other variations. There have been people that have demonstrated the ability to network using carrier pigeons. You don't typically get as much bandwidth that way as you would with, say, fiber, but the point is, it doesn't matter. Whatever suits your need. Whether you need to beam signals to an orbiting satellite and back to Earth, or whether you're using two cans and a string, the Physical Layer sends the data.

And that's the OSI Reference Model, folks. Next, I'd like to talk about a Data Link Layer Protocol that anyone reading this book is going to have to get really cozy with.

Ethernet

Ethernet was the brain child of an engineer named Bob Metcalfe, and was developed in the mid-1970s. The concept of networking was clearly in its infancy. There were some, at the time, who considered the design of Ethernet to be an utter failure waiting to happen. It's funny how some things work out, isn't it? Today, Ethernet is easily the most popular Data Link Layer protocol there is.

While Ethernet is technically a Data Link Layer protocol, there is also a mechanism associated with it that is technically Physical Layer. It's called Carrier Sense Multiple Access/Collision Detection (CSMA/CD). When Ethernet first started, it used a single coaxial cable that connected to every single computer in the network. It looked liked this:



The coaxial cable has one thin strand of copper in the middle of it, surrounded by quite a bit of insulation. This is surrounded by an additional meshwork of copper, which is, in turn, surrounded by more insulation. This cable is designed to carry an electrical signal. The Ethernet NIC (in the computer) generates an electrical signal on the cable. This signal is interpreted by the receiving Ethernet NIC as a binary 0 or a binary 1.

The problem is that only one NIC could generate an electrical signal at a time. There was only one medium (the cable). To deal with the possibility that two machines on the network might try to send a signal at the same time, collision detection was invented. The transmission process worked like this:

1. Convert the frame to binary digits and prepare it for transmission
2. Check to see if the medium (the cable) is idle (there's no signal); if it is not idle, wait until it is
3. Start transmitting your signal
4. Check for a collision; if there's a collision:
 - a. Continue the transmission for a certain period of time so that every machine on the network realizes there's a collision
 - b. Stop transmitting and wait for a random amount of time
 - c. Try transmitting again (start at Step 2)
5. The signal has been transmitted

What if a collision was detected, you waited for a random amount of time, tried again, and you still got a collision? Well, you just kept on trying. There are a maximum number of collision retries in the protocol in which it will declare a failure and give up, but until it reaches that threshold, it just keeps trying.

So there's the CSMA/CD. You have the carrier sense. This is what detects the medium and decides whether or not it's idle. The multiple access is the process listed above. This allows many computers to use the same medium (cable, in this example). And finally, the collision detection is a set of procedures to follow if two (or more) computers attempt to use the medium at the same time.

Many times, CSMA/CD is compared to a group of people talking in a room. In a well-behaved room, one person talks while the others listen. If anyone else wants to talk, they wait until the speaker is done talking (carrier sense). As often happens, two people may attempt to talk at the same time (a collision). When that happens, each person stops talking, and waits to see if the other will begin speaking. If they don't, given a polite amount of time, one will start talking, and the conversation will continue.

Well, now that you know how Ethernet communicates with the physical media, let's look at the different physical media you can use with Ethernet.

Layer 1 Ethernet (Cabling)

What did that say? Layer 1 Ethernet? I thought Ethernet was a Layer 2 protocol! It is. Calm down. I deliberately named this section. This is because there are many common physical media that have become synonymous with Ethernet. And while, technically, Ethernet is strictly Layer 2, a vast majority of the industry will commonly refer to Ethernet hardware, and you need to be aware of what they mean.

Coaxial

As I mentioned in the previous section, Ethernet started life on coaxial cables. The ends of the cable looked like this:



This connector is called a BNC connector. To have multiple machines connected to the network, you needed to have T-connectors to allow a coaxial cable to be connected from the computer into the main line. At the end of the main line, you had a terminator. This was a terminating resistor that reflected the signal back through the cable.

This is called a bus topology. This signal travels back and forth along the main line. Every machine that is attached to the main line sees the frames that every other machine sends, but it only pays attention to the ones addressed to itself. This topology has plenty of weaknesses.

For one, the bus topology doesn't expand very well. A single length of coaxial cable is only rated for 185 meters. If the cable is longer than that, the signal will experience attenuation. This is the distance beyond which the electrical signal will be so weak that it won't be properly detected. When an electrical signal is sent along the copper, it will immediately start "losing steam." The signal will become weaker and weaker the farther it goes. After 185 meters on a coaxial line, the signal will become so weak, other devices in the chain won't really be sure there's a signal there. It becomes unreliable beyond that point.

Another weakness in the bus topology is that if there's a break anywhere in the main line, the whole network is down. Think of an electrical circuit. If there's a break in the cable, there's a break in the circuit. Now, no computer is talking to anybody. Network Administrators in yon days of yore will likely reflect on the ol' coaxial network days and shudder. Compared to what we have now, they were a nightmare to maintain.

Twisted Pair

What do we have now? Well, the most common medium is called twisted pair. This comes in many varieties. On the high level, there's Unshielded Twisted Pair (UTP) and Shielded Twisted Pair (STP). STP is expensive, and the shielding can make it difficult to bend or shape. On the other hand, the additional shielding does a better job of preventing outside interference. UTP is, by far, the most common. They both come in different grades called categories. Some of the most common categories include category 3, category 5, category 5e, and category 6. These are most commonly abbreviated to CAT3, CAT5, CAT5e, and CAT6. CAT5 uses thicker cables inside than CAT3. It also has more twists per meter. Likewise with CAT5e and CAT6, the higher the number, the thicker the copper cables and the more twists per meter. Also, the higher the category, the greater the performance. CAT3 is only rated for 10 Mbps Ethernet, while CAT5e and CAT6 are required for gigabit. There are many more than these, but these are the categories you're most likely to deal with.

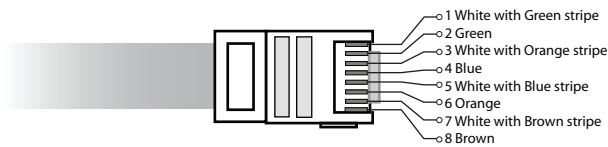


Twisted Pair is actually a bundle of eight smaller cables-four pairs. Each pair is twisted around each other many times throughout the length of the cable. The cables are twisted to help block crosstalk. This would be when the electrical signal on one of the other pairs crosses over to any of its neighboring pairs. It also helps to minimize electromagnetic interference (EMI). Any time an electrical circuit is rapidly changing signals, EMI becomes the by-product. It's a signal field emitted by the circuit, and it can interrupt other nearby legitimate circuits, if you're not careful. STP is still used for this purpose. This adds additional insulation and EMI shielding.

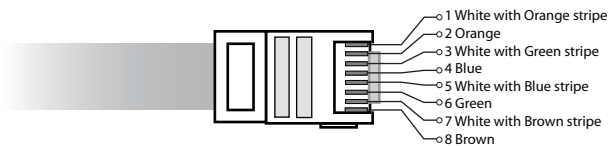
The cable comes to an end called RJ45. As you can see from the diagram, it has eight metal positions called pins. Each pin connects with one of the eight wires inside the cable.

You may, or may not, have noticed that the eight cables inside are aligned in a certain order. This conforms to one of two standards put forth by the Telecommunications Industry Association/Electronic Industries Association (TIA/EIA). The standard is called TIA/EIA-568-B.1-2001. It specifies that there are two methods for aligning the cables. They are labeled T568A and T568B. Casually, these are referred to as the A-standard and B-Standard.

To see which standard a particular cable uses, hold the RJ45 end in front of your face with the clip-side away from you. Then, read the colors from left to right.



T568A



T568B

T568A	T568B
1. White with Green stripe	1. White with Orange stripe
2. Green	2. Orange
3. White with Orange stripe	3. White with Green stripe
4. Blue	4. Blue
5. White with Blue stripe	5. White with Blue stripe
6. Orange	6. Green
7. White with Brown stripe	7. White with Brown stripe
8. Brown	8. Brown

Which one do you use? In the end, the important thing is to be consistent. If you are using the A-standard, always use the A-standard. If you are using the B-standard, always use the B-standard. This is especially true of the cable itself. If you have used the A-standard on one end of the cable, use the A-standard on the other end of the cable. There's one exception that I'll talk about in a minute.

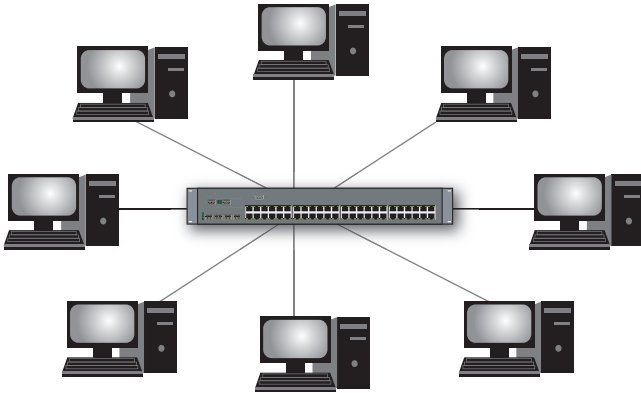
For Ethernet 10Base-T and 100Base-T, only pins 1, 2, 3, and 6 are important. The first pair, 1 and 2, is used to transmit data. The second pair, 3 and 6, is used to receive data. In the A-standard, transmitting is the green pair and receiving is the orange pair. In the B-standard, it's the opposite. The orange pair transmits, and the green pair receives. The blue pair and brown pair are unused.

As I mentioned earlier, a standard Ethernet cable is one that has the same standard arrangement of cables on both ends of the cable. This is commonly referred to as a straight-through cable. This is because a signal sent on pin 1 on one side will be received on pin 1 of the other side; a signal sent on pin 2 on one side will be received on pin 2 of the other side; and so forth. This cable is used to plug a computer (or most any other device) into an Ethernet hub or switch.

But what if we need to connect two hubs or switches together? Or what if we want to directly connect two computers together? You would use what is called a crossover cable. The crossover cable uses one standard on one side, but the opposite standard on the other. For example, if you were using the T568A standard, you would create a crossover cable by making one end of the cable A-standard, and the other end of the cable B-standard. You're matching your transmit pair directly to the receiving pair of the other side.

If you've never had to make or use a crossover cable, don't be too surprised. Many NIC manufacturers have added a technology to automatically sense whether a crossover signal is needed or not. It simply compensates for whatever hardware is in place. Brocade switches are included in this. While it is technically proper to use a crossover cable when connecting two Brocade switches, it is not necessary. The switches are intelligent enough to know what you meant to do.

Twisted pair networks use a star topology. At the center of the star is a hub or switch. All of the participants in the network plug into the star.



Unlike the bus topology, this is very expandable, and can accommodate virtually any number of hosts you may need. If a cable is broken, only one participant is negatively affected. The rest of the network may carry on. A disadvantage to this topology is that now the hub or switch becomes a single point of failure. If it goes down, the network goes down. There are methods of protecting yourself from this kind of failure which we will discuss in later chapters.

Twisted pair can achieve throughput of up to gigabit (1000 Mbps) speeds. Unfortunately, like coaxial, it has an attenuation limit as well, and it's shorter. Signal is rated for up to 100 meters.

Fiber

What if you need lengths of cables much longer than 100 meters? Or what if your facility has a lot of EMI? Here comes fiber.

The major advantages with fiber are its attenuation limit (up to 70 km) and, because it uses light rather than electricity, it's invulnerable to EMI. It can also achieve much greater speeds than its copper counterparts. The current maximum is 10 Gbps, but technology is always moving forward. Speeds of 100 Gbps or more are realistically not too far away. And the media that will bring these speeds to you? Fiber.

The main disadvantage to fiber is that it's expensive. The cables are made by combining fiber optic strands and insulating them. A laser light generates the signal. The more powerful the laser, the longer the attenuation limit. There are two varieties you need to make yourself familiar with:

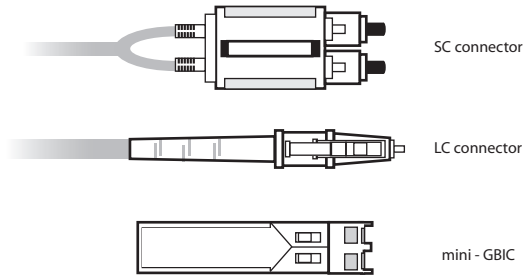
- Multimode Fiber (MMF)
 - 1000 Mbps (SX) attenuation: 550 meters
 - 10 Gbps (SR) attenuation: 300 meters
- Single-mode Fiber (SMF)
 - 1000 Mbps (LX) attenuation: 5 km
 - 1000 Mbps (LHA) attenuation: 70 km
 - 10 Gbps (LR) attenuation: 10 km
 - 10 Gbps (ZR) attenuation: 70 km

The distance you can achieve with fiber depends on the type of fiber you use, the speed of the protocol, and the type of laser. Gigabit fiber uses a connector called a Gigabit Interface Converter (GBIC). Brocade uses a smaller version of this connector called a mini-GBIC. On the switch that supports a gigabit fiber interface, there will not be a port to plug a cable into. There will only be an open module slot. This slot houses the mini-GBIC. The mini-GBIC contains the port that you plug the cable into. There are four different mini-GBICs you could purchase for this slot:

- 1000SX (used with multimode fiber)
- 1000LX (used with single-mode fiber)
- 1000LHA (used with single-mode fiber)
- 1000TX (used with twisted pair copper)

This provides a great deal of flexibility for the customer. If the purpose of the switch port changes over time (say, we want to go from 1000SX to 1000LHA), the customer doesn't need to purchase a new switch. They just need to replace the mini-GBIC.

The cable terminates using either an SC or an LC connector. The SC is an older connector. It can still be found in some equipment, but it has largely been replaced by the smaller LC connector. Brocade's mini-GBICs and 10G ports use LC connectors.



Hub

With twisted pair, you need to have some way to have more than two computers share information with each other. In the beginning, this device was called a hub.

A hub is a device with many RJ45 ports on it. As a device, it's not remarkably intelligent. It has a very straight-forward purpose. Whenever it receives an electrical signal on one of its ports, it replicates that signal out all of its other ports. That's it.

This method is similar, in some respects, to the bus topology we talked about earlier. All of the computers connected to the hub (or the main line, in bus topology) saw all of the frames that were sent. It didn't matter whether the frames were meant for that computer or not, it saw them. Now, by default, the Ethernet protocol ignores the frames not specifically addressed for its own NIC. Still, though, you can see that this is a fairly inefficient way to communicate. It's also potentially insecure.

Switch

While hubs still exist, you will find that, more commonly, they have been replaced by the switch. The switch looks very similar to a hub. It's a device with several RJ45 ports on it. What sets it apart is that it has a brain (a CPU). Where a hub is strictly a Physical Layer device (it just replicates the signal), a switch is a Data Link Layer device.

When the switch receives a signal on one of its ports, it keeps track of physical addresses connected to it. It then has an address book to draw from. When a signal comes in, the switch checks to see what physical address the frame is bound for. It will then look at its address book to see if it knows what physical port (of itself) the destination physical address is connected to. If the switch finds the destination address, it will forward the frame out of that port and only that port. If the switch doesn't find the destination address in its address book, it will act like a hub, and forward the frame out all of its ports. The difference is that the switch will pay attention to replies. If the switch gets a response frame from the destination address, it will add that address to its address book and remember it for next time.

Switches provide another wonderful advantage over hubs. They are capable of something called full-duplex communication. This is the concept of transmitting data and receiving data at the same time. We'll talk much more about this a little later in the chapter.

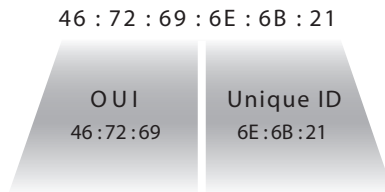
Layer 2 Ethernet (Addressing)

Now we know how Ethernet is typically laid out physically. Let's talk more about the protocol itself.

MAC Address

We talked about addressing in the Data Link Layer section. Ethernet uses a 48-bit (6 byte) number to uniquely identify every physical device. This address is called the Media Access Control (MAC) address. It is typically represented in hexadecimal (because writing out 48 1s and 0s makes for a very long number), and looks like this: 46:72:69:6E:6B:21. Different operating environments represent MAC addresses differently, but the number is the same. For example, Brocade represents the MAC address by separating it into two-byte pairs. The same number would be represented this way: 4672.696E.6B21.

With 48 bits, you can represent over 281 trillion unique addresses. While this number looks big and intimidating, there is a rhyme and reason to it. The address is split into two 24-bit sections. The first 24-bits are assigned by the Institute of Electrical and Electronics Engineers (IEEE). This 24-bit number is referred to as an organizationally unique identifier (OUI). Every legitimate Ethernet hardware manufacturer has at least one of these OUIs. Larger manufacturers have several. The second part of the address (the last 24-bits) is a unique identifier created by the manufacturer. It's the unique serial number of the device. This gives the company 16,777,216 unique addresses per OUI.



So, if we take a look at an example MAC address, 46:72:69:6E:6B:21, we can see that the OUI of the manufacturer is 46:72:69, and that the NIC's unique identifier (assigned by the manufacturer) is 6E:6B:21. There are software programs available that will help you identify the OUI with a known manufacturer. This can be very valuable when troubleshooting problems on your network.

Ethernet Frame

We've mentioned the concept of a Data Link Layer frame frequently in this chapter. Let's take a good look at an Ethernet frame.

Preamble 8 bytes	Destination address 6 bytes	Source address 6 bytes	Length/ Type 2 bytes	Data 46 to 1,500 bytes	Frame check sequence 4 bytes
---------------------	-----------------------------------	------------------------------	----------------------------	---------------------------	---------------------------------------

Ethernet Frame

As you can see, the IEEE 802.3 Ethernet frame is broken up into six sections. Let's get to know each one individually:

Preamble. This is an 8-byte section of alternating 1s and 0s. It provides a 5 MHz clock to help synchronize the sender and receiver for the coming bit stream.

Destination Address (DA). This is a 6-byte (48-bit) number. It is the MAC address of the device that the frame is being sent to.

Source Address (SA). This is a 6-byte (48-bit) number. It is the MAC address of the device sending the frame.

Length/Type Field. This is a 2-byte field that can either be a Length Field or a Type Field. If the number is smaller than 1,536 (0x600 in hexadecimal), it is a Length Field, and the number describes how many bytes of the Data field are used as the IEEE 802.2 LLC header. If the number is greater than 1,536, it is a Type Field. The number identifies what Network Layer protocol should process the frame (0x0800, for example, is IP). If this is a Length Field, the type will be described in the LLC header.

Data. This is the data. This is the packet that was handed down from the Network Layer. This can be any length from 46 bytes to 1,500 bytes, using a standard Maximum Transmission Unit (MTU). As mentioned in the previous field, this may contain an IEEE 802.2 LLC header as part of the field.

Frame Check Sequence (FCS). This is a 4-byte (32-bit) number. It is a cyclical redundancy check (CRC) that is calculated using the DA, SA, Length, and Data sections. When the frame is created, the CRC is calculated and applied to the end of the frame. On the receiving end, the receiver recalculates the CRC and makes sure that it matches with the FCS that was sent. This allows for error checking.

These are the individual pieces that make up each Ethernet frame. The frame reveals where it's coming from, where it's going to, how long it is, and, finally, a checksum to make sure the receiver got the data that the sender intended. So, now that we're communicating, let's talk about how to communicate more efficiently.

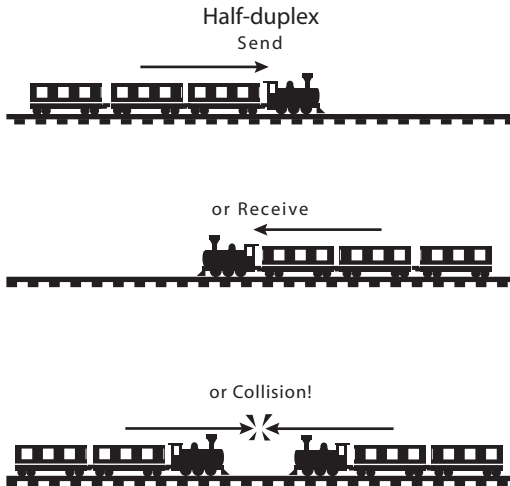
Half-duplex vs. Full-duplex

Ethernet uses two ways to communicate from one machine to the next. Let's look at them both.

Half-duplex

Remember when we talked about the early days of Ethernet? It used a bus topology (several computers connected to one cable)? This worked okay, but it only provided half-duplex communication. Half-duplex is the polite way to communicate. You don't speak while the other person is talking (in this case, you can't). *One device speaks at a time.*

Think of half-duplex as a train on a railroad track. The train can obviously only move in one direction, and there can't be any other trains on the track. If there are, that could cause a collision.



As we described earlier, when a collision occurs, both talkers stop talking, and wait for a random period of time before they try again. This is certainly a polite way to communicate, but it's not very efficient. But because the cable could only carry one signal, that's all the original Ethernet could do. One electrical signal. One train. One track.

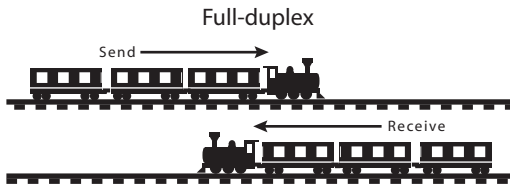
Hubs run into a similar problem. These are incapable of anything other than half-duplex communication as well. They simply repeat the signal, much like the main cable of the bus topology (even though a hub would make it a star topology).

A group of computers connected to a hub (or even several hubs) would be considered a *collision domain*. This defines an area in the network where collisions may occur. Just like two people trying to have a conversation in a crowded room, the more people in the room, the greater the possibility that two people will be trying to talk at the same time (collision). As you can imag-

ine, this would make it harder and harder to hold a conversation. For best network efficiency, one should always minimize the size of the collision domain (or, if possible, eliminate it completely).

Full-duplex

What if you could *listen and talk at the same time*? Wouldn't that be great? This is called full-duplex communication. If we go back to the railroad example, now you've got two railroad tracks that are parallel to each other. On one, we have a train traveling in one direction. On the other, we have another train traveling in the opposite direction.



Full-duplex is not only efficient, but it takes away the risk of collisions. Recall that using twisted pair, you only needed two pairs (of the four pairs) for 10/100 Mbps Ethernet. When using full-duplex, one pair is used to transmit data, and the other pair is used to receive data. You completely eliminate the risk of having two devices wanting to use the same pair at the same time. Each pair has its purpose, and travels in only one direction.

But we can't use a hub for full-duplex communication. For full-duplex, you have to have a switch. When using a switch with full-duplex communication, you've essentially eliminated your collision domain. Each device plugged into the switch has its own private communication channels to and from the switch.

A switch can operate at half-duplex. In fact, most switches will give you the option to configure the speed (10 Mbps, 100 Mbps, 1000 Mbps) and duplex of each port. For example, you could have one port set at 100 Mbps full-duplex, and another set to 10 Mbps half-duplex. Why would you want to do this? Well, there may be some places you may work that are dependent on older equipment. Some of this equipment may only be capable of half-duplex communication. If you have such equipment, and it's plugged into a switch configured to be half-duplex, collisions are still possible. Fortunately, they'll be contained within the switch and the half-duplex equipment. No other elements of the network will be affected. Here, the collision domain exists between the node (the half-duplex equipment) and the switch port it's plugged into.

Autonegotiation

It didn't take long for manufacturers to realize that all of these different settings (speed, duplex, etc.) may confuse a majority of their customers. In the mid-90's, autonegotiation became the mainstream default setting for both switch ports and NICs. Autonegotiation uses Physical Layer electrical pulses to investigate what speed and what duplex it can operate at. If you have a switch

port that is set to autonegotiate, and you have a workstation with a NIC that's set to autonegotiate, and you plug the workstation into the switch port, electrical pulses will be sent to detect what the highest speed and duplex setting the two network devices can communicate on.

Let's say you have a 100 Mbps full-duplex NIC and a 100 Mbps full-duplex switch, and both are set to autonegotiate. When you connect these devices together, they should "autonegotiate" to operating at 100 Mbps full-duplex. They negotiate or agree to operate at these settings. What if your NIC was 10 Mbps half-duplex, but your switch was still 100 Mbps full-duplex? Using autonegotiation on both sides, the connection will communicate at 10 Mbps half-duplex. The NIC and switch negotiated, and the switch realized that 10 Mbps half-duplex was the best that the NIC could communicate at. They negotiated.

Sounds great, doesn't it? Well, there's a downside to it. Suppose you have a workstation with a 100 Mbps full-duplex NIC set to autonegotiation. You plug it into a switch which is configured to 100 Mbps full-duplex, but is not set to autonegotiate. It is configured specifically to communicate at 100 Mbps full-duplex. It is forced to 100 Mbps full-duplex, as they say. What happens? Well, typically, the NIC will correctly deduce the speed (100 Mbps, in this example), but won't get the expected responses when negotiating duplex. Just like in real life, it's difficult to negotiate with someone who won't negotiate with you. The NIC fails to detect the duplex setting, so it fails to the lowest common denominator- half-duplex. So, in the end, you have a switch that is configured to communicate at 100 Mbps full-duplex, and a NIC that has autonegotiated to function at 100 Mbps half-duplex. This is called a duplex mismatch.

The duplex mismatch occurs more times than you might think. The result is that the switch believes it can talk and receive at the same time, and the NIC doesn't. What's going to happen? Well, frames will certainly be lost. Depending on the upper layer protocols (the Transport Layer, most likely), the data will still get there, but very slowly. With the errors and dropped frames, the Transport Layer will have to resend segments many times to make sure all of the data reaches the destination.

What can you do with a duplex mismatch? You have two options:

1. Change the NIC's setting so that it is also forced to communicate at 100 Mbps full-duplex.
2. Change the switch port's setting so that it will autonegotiate.

While autonegotiation seems like a great idea, its success still depends on all the devices involved being configured the same way. In real world experience, you'll also find that different manufacturers have different ideas about how autonegotiation should work. The result is that you could have a NIC set to autonegotiate and a switch port set to autonegotiate and you could still end up with a duplex mismatch!

What's a poor network engineer to do? Well, it's similar to the duplex mismatch solution listed above. In the end, you need to make a decision as to which one you will use.

You could set all of the devices and switch ports to autonegotiation all the time. If you do this, you won't have to worry about changing so many settings, certainly. Almost all network equipment (NICs, switches, etc.) are set to autonegotiate by default. The downside is that, unless your equipment is all made by one manufacturer, you may run into autonegotiation failures that will create duplex mismatches. You may also run into problems with someone bringing in outside equipment (say, a consultant bringing in a laptop) that is forced to full-duplex (autonegotiation is off).

You could set all of the devices and switch ports to be forced to full-duplex all the time. You won't have to worry about autonegotiation failures this way. It will require a lot more setup time, though. All existing switch ports and NICs will have to be configured to full-duplex. Any new machine or switch will have to be configured to full-duplex before it's put into use. And you'd still have the risk mentioned in the previous paragraph. What if a consultant brings in a laptop that's configured for autonegotiation?

Each solution has its advantages and disadvantages. Every experienced engineer will have his or her preferred method of dealing with the problem. Experience will teach you what is best for you.

Well, now we know the technologies. Let's take a look at what Brocade has to offer us.

Brocade Ethernet Switches & Routers

Brocade offers a complete line of enterprise and service provider Ethernet switches, Ethernet routers, application management, and network-wide security products. With industry-leading features, performance, reliability, and scalability capabilities, these products enable network convergence and secure network infrastructures to support advanced data, voice, and video applications. The complete Brocade product portfolio enables end-to-end networking from the edge to the core of today's networking infrastructures. To learn more about Brocade products, visit <http://www.brocade.com>.

Summary

- The OSI Reference Model describes how machines can talk to one another
- The OSI Reference Model consists of seven layers:
 - **Application** gets data from the user
 - **Presentation** formats the data to a certain standard
 - **Session** keeps track of which conversation the data belongs to
 - **Transport** makes sure the data gets to its destination
 - **Network** decides what path the data will take to get to its destination on a logical level

- **Data Link** prepares data for physical transmission, and decides how the data will get to its destination on a physical or local level
- **Physical** the method that actually transports the data
- Ethernet is a very popular Data Link layer protocol
- Ethernet is usually cabled using either coaxial, UTP, or fiber
- Ethernet is usually in a star topology, centrally connected to a hub or switch
- MAC address is the physical address of the Ethernet device
- Half-duplex communication can only listen or speak at one time
- Full-duplex communication can listen and speak simultaneously

Chapter Review Questions

1. What layer of the OSI Reference Model interacts with the user?
 - a. Physical
 - b. Presentation
 - c. Application
 - d. Data Link
2. What address is considered the physical address of a device in Ethernet?
 - a. MAC address
 - b. LLC address
 - c. IP address
 - d. IPX address
3. What layer of the OSI Reference Model makes sure the data reaches its destination?
 - a. Network
 - b. Physical
 - c. Session
 - d. Transport
4. What does autonegotiation decide?
 - a. The speed and duplex of the communication
 - b. The MAC address of the NIC
 - c. The physical media
 - d. The IP address of the host

5. What layer of the OSI Reference Model decides how the data will get to its destination?
 - a. Physical
 - b. Application
 - c. Network
 - d. Presentation
6. On the Data Link layer, what form is the data in?
 - a. Segments
 - b. Frames
 - c. Packets
 - d. Bits
7. What layer of the OSI Reference Model actually sends the data?
 - a. Application
 - b. Network
 - c. Data Link
 - d. Physical
8. On the Transport layer, what form is the data in?
 - a. Segments
 - b. Frames
 - c. Packets
 - d. Bits
9. Which layer of the OSI Reference Model keeps track of the different conversations that may be occurring?
 - a. Presentation
 - b. Data Link
 - c. Physical
 - d. Session
10. On the Network layer, what form is the data in?
 - a. Segments
 - b. Frames
 - c. Packets
 - d. Bits

Answers to Review Questions

1. c. The user sends the data to the Application layer, and it is the Application layer that finally presents the data to the user in a reply.
2. a. The Media Access Control (MAC) address is the physical address of an Ethernet device.
3. d. The Transport layer uses different methods (depending on the protocol) to make sure the data gets to its destination.
4. a. Autonegotiation decides what speed (10 Mbps, 100 Mbps, etc.) and what duplex (full or half) the Ethernet communication will use.
5. c. It's the Network layer's job to decide, logically, how the data will get to its destination.
6. b. The Data Link layer works with frames.
7. d. The Physical layer is the one that actually creates an electrical, optical, or other physical signal and transports the data.
8. a. The Transport layer works with segments.
9. d. The Session layer keeps track of the different conversations.
10. c. The Network layer works with packets.

TCP/IP

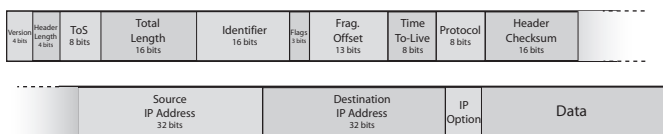
The most popular protocol stack to date is TCP/IP. It gained a great deal of popularity as more and more people and businesses started using the Internet. The Internet uses TCP/IP as its protocol suite. As a result, many businesses have adopted TCP/IP for their internal networks as well.

Some might ask what layer TCP/IP resides in. It doesn't, really. TCP/IP is not a specific protocol. It is a collection of different protocols, extending from Layer 3 (Network) to Layer 7 (Application). In this chapter, I will introduce you to the two layers in the TCP/IP protocol stack that you will want to be most comfortable with: Layers 3 & 4.

Layer 3 TCP/IP: IP

As the title shows, the Layer 3 (Network) protocol in TCP/IP is called Internet Protocol (IP). This is the main protocol, but it is not the only protocol in the TCP/IP Network Layer. As we said in the previous chapter, the Network Layer's job is to decide how the data will get to where it needs to go. I also mentioned that it uses addressing, networks, and routing protocols to accomplish its task. We're going to look at each of these in detail in this section.

But first, let's take a look at what we're dealing with in Layer 3. This layer would have received the *segment* from Layer 4 (Transport). IP adds a few bytes of its own to the front of the segment to make it a *packet*. These bytes are referred to as the *IP header*. The header is subdivided this way:



Version. This is a 4-bit number. In most networks, this number will be "4" (or 0100, in binary; more on this later). The current version of IP is referred to as IPv4 (or "IP version 4"). A new version, which is already gaining ground, is IPv6.

Header Length (HLEN). This is also a 4-bit number. It represents the length of the header, but it uses a 32-bit (4-byte) word as its unit. For example, most headers are 20 bytes long. To represent this as 4-byte words, this number would be 5 (5 x 4 bytes [32-bits] = 20 bytes).

ToS with IP Precedence Bits. This is one byte (8 bits). The first three bits are priority bits. They're used to set IP Precedence. The next four bits describe the packet's Type of Service (ToS). This allows for special instructions in handling different types of packets. The last bit is unused.

Total Length. This is a 16-bit (2 byte) number. It is the length (in bytes) of the entire packet, including the header and the data that follows.

Identifier. This is a 16-bit (2 byte) number that uniquely identifies the packet.

Flags. This is a 3-bit field. Should the packet need to be fragmented, it will be indicated here.

Frag Offset. This is a 13-bit number. If the packet is too large to fit on a single frame, it will be fragmented. The offset will help in reassembling the packet.

Time-To-Live (TTL). This is an 8-bit (1 byte) value that is set when the packet is generated. The idea is that if the packet doesn't get to its destination before the TTL expires, the packet is dropped. This is a safeguard to keep the network free of orphaned packets (e.g., packets that never get to their destination but hop from router to router to router).

Protocol. This is an 8-bit (1 byte) value that indicates the IP Protocol. This number could represent a Network Layer protocol, or it could represent the Transport Layer protocol that generated the segment (and that will need to process the segment on the other side). Some well-known values here are: 1 (ICMP), 6 (TCP), 17 (UDP), 50 (ESP) and 89 (OSPF).

Header Checksum. This is a 16-bit cyclical redundancy check (CRC) of the header. It allows the receiving side to make sure the header arrived intact.

Source IP Address. This is the 32-bit address of the device that created the packet.

Destination IP Address. This is 32-bit address of the device that the packet is to be delivered to.

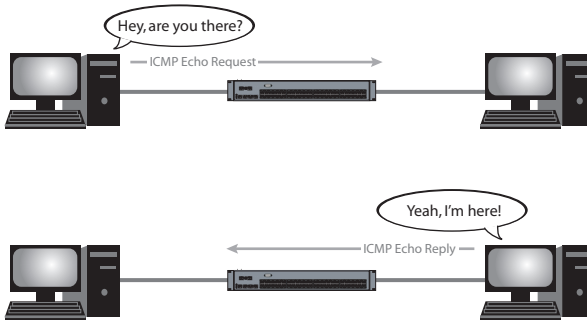
IP Option. This field is typically used for testing and low-level network debugging.

Data. This is the segment received from the Transport Layer.

You can see from this that IP uses 32-bit addresses. These are usually separated into four 8-bit numbers separated by periods. They are also usually represented in decimal (base 10). An example of an IP address might be: 192.168.100.1.

I mentioned earlier that there are other Network Layer protocols in IP. Here are a couple of the more well-known ones:

Internet Control Message Protocol (ICMP) is used to send messages between Layer 3 devices. It can check to see if an address is responding. It can tell another Layer 3 address that it's sending information too fast. It can help trace the path through several routers. The most common use for ICMP is its echo protocol, more commonly known as *ping*. This is a really simple protocol. One device will send an “echo request” message to another device. If that device is active, it will send an “echo reply” back to the first device. Simply stated, it's as if one device shouts out, “Hey, are you there?” And if the device is there, it responds with, “Yeah, I'm here!”



Address Resolution Protocol (ARP) is used to match an IP address with a MAC address. It kind of bridges the gap between the Data Link Layer and the Network Layer. We'll talk more about ARP later in the chapter.

Thinking in Binary

The numbers that we use in our daily lives (with rare exception) are decimal. This means base 10. What does “base 10” mean? It means we have ten digits (single numbers) we can use before we have to start using two digits. For example, we have the number 0 (zero). That's one number. We also have nine other numbers: 1, 2, 3, 4, 5, 6, 7, 8, and 9. If we're going to represent any number higher than that, we need to start using two-digit numbers (e.g., 10, 11, 12, etc.). That's decimal, or base 10.

Computers, overly simplified, are made of up of billions of transistors. Transistors are electrical components that can be in one of two states. There's either a voltage (electricity is passing through them), or there is no voltage (no electricity is passing through them). You can think of transistors like a light switch. The light switch is either on, or it's off. You've got two options.

It didn't take computer engineers long to realize that they could use this anomaly to count. But we can only really represent two numbers: 0 (off) and 1 (on). So, how can that be useful? Well, if you're only talking about a single digit, it really isn't very useful. But when you string a bunch of digits together, now you can represent all kinds of numbers.

What we're talking about is *binary*. Binary is a way to represent numbers. It's like decimal, but in decimal, we have ten digits. In binary, we have two. In decimal, you would count 0,1,2,3,4,5,6,7,8,9,10. In binary, you would count 0,1,10. That last number looks like a number ten, doesn't it? It's not. If you're using binary, that number represents the number two.

In decimal, each digit place represents a multiple of ten. For example, 10, 100, and 1000 represent ten (10×1), one hundred (10×10), and one thousand ($10 \times 10 \times 10$). If we see "10,000," we know that as ten thousand ($10 \times 10 \times 10 \times 10$), because there's a "1" followed by four zeros. Binary works in a similar fashion, but instead of multiples of 10, it's multiples of two. For example, 10, 100, and 1000 in binary represent two (2×1), four (2×2), and eight ($2 \times 2 \times 2$). And if we were to take it one step further, "10000" would represent sixteen ($2 \times 2 \times 2 \times 2$). As decimal is referred to as base 10, binary is referred to as base 2.

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	1

Why am I telling you all of this? You'll find out very quickly. As I mentioned in the previous chapter, switches and routers are computers. They have a CPU. They have RAM. As such, they also operate in binary.

You would do very well to memorize the diagram on this page. It will help you in more ways than just impressing your friends.

The first example of using binary is in the IP address itself. I gave you an example of an IP address: 192.168.100.1. There are four decimal numbers separated by a period. But what you need to know is that the computers, switches, and routers see that number this way:

1100 0000 . 1010 1000 . 0110 0100 . 0000 0001

Notice that each of the four decimal numbers are represented by eight binary digits. You usually call a binary digit a bit. So, each of the four decimal numbers are represented by eight bits. Let's look at the first number: 1100 0000. You've got a 1 in the 128 bit place, and a 1 in the 64 bit place, and all the rest are zeros. So, if you add $128 + 64$, you'll see that you get the number 192. Likewise, the next number: 1010 1000. Here, we've got a 1 in the 128 bit place, a 1 in the 32 bit place, and a 1 in the 8 bit place. This number, in decimal, would be $128 + 32 + 8$, or 168. The third number is 0110 0100. This has a 1 in the 64 bit place, a 1 in the 32 bit place, and a 1 in the 4 bit place. The number is $64 + 32 + 4$, which equals 100. And the last number's the easiest. You've only got one "1," and it's in the 1 bit place. The binary number "0000 0001" equals our decimal "1."

Let's look at eight bits. To get the smallest possible number (with just eight bits), we could set all of the bits to 0, right? This would give us the number 0. To get the largest possible number, we could set all of the bits to 1. This would give us the number 255 ($128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$).

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	1
0000 0001 = 1				1100 0000 = 128 + 64 = 192			
0000 0110 = 4 + 2 = 6				1100 1000 = 128 + 64 + 8 = 200			
0010 1000 = 32 + 8 = 40				1111 1010 = 128 + 64 + 32 + 16 + 8 + 4 + 2 = 250			
0110 0100 = 64 + 32 + 4 = 100				1111 1111 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255			

By the way, eight bits are referred to as one byte (yes, nerds think they're very funny). You can think of the diagram above as either eight bits, or one byte.

What does this mean with our IP address? Well, we see that each of the four numbers (192.168.100.1, for example) is actually represented by eight bits (or one byte). So for each of the four numbers, the smallest number it can be is 0, and the largest number it can be is 255. You will never see an IP address that looks like this: 359.278.913.4. All of the digits (besides the last one) are bigger than 255. They would require more than eight bits to represent them. This means that the smallest possible IP address is 0.0.0.0, and the largest possible IP address is 255.255.255.255.

An IP address is a 32-bit number. As I mentioned previously, it is usually represented by four 8-bit numbers separated by periods. On top of that, the four 8-bit numbers are usually converted to decimal. Rarely will you see anyone referring to an IP address as “1100 0000 . 1010 1000 . 0110 0100 . 0000 0001.” You will almost always see it as “192.168.100.1.”

Network Classes

Now we know what an IP address is. What's next? Well, the authors of TCP/IP wanted to use the 32-bit number of the address to represent both the *network* (group of IP addresses) and the *host* (the specific address of the host). Let's compare this to the Post Office delivering a letter. The address on the letter says, “123 Fake Street.” Now that's one line, but it represents two things for the postman. For one, the postman knows the house is on Fake Street. Fake Street has a lot of houses on it. It's a group of houses, just like a network is a group of hosts. It's the “123” part of the address that tells the postman to deliver it to a specific house. An IP address works the same way. It has a network portion (like the street name) and a host portion (like the specific house).

How do we distinguish which part is the network and which part is the host? Whenever you configure a device with an IP address, you must include one other number called a *subnet mask*. The subnet mask is also a 32-bit number that is usually represented by four 8-bit numbers separated by periods. It is also usually represented in decimal, but its meaning is far more clear when seen in binary. If we go back to our previous example of an IP address, 192.168.100.1, we could accompany that with a subnet mask of

255.255.255.0. Looks like an IP address, doesn't it? Instead, it actually shows us (and the networking equipment) which part of the IP address is the network and which is the host. Let's look at it in binary:

Address – 192.168.100.1
Subnet Mask – 255.255.255.0

Address: 1100 0000.1010 1000.0110 0100.0000 0001
Subnet Mask: 1111 1111.1111 1111.1111 1111.0000 0000

Notice that from left to right, the subnet mask shows every bit to be “1” until it gets to the 25th bit. The 25th bit is “0,” and so are the rest of the bits to the end. The subnet marks all of the bits that belong to the network with a “1,” and it marks all of the bits that belong to the host with a “0.”

The first 24 bits (in this case) mark the network address:

Address – 192.168.100.1
Subnet Mask – 255.255.255.0

Address: 1100 0000.1010 1000.0110 0100.0000 0001
Subnet Mask: **1111 1111.1111 1111.1111 1111.0000 0000**
Network Address: **1100 0000.1010 1000.0110 0100.0000 0000**
In Decimal: **192 . 168 . 100 . 0**

And the last 8 bits mark the host address:

Address – 192.168.100.1
Subnet Mask – 255.255.255.0

Address: 1100 0000.1010 1000.0110 0100.0000 0001
Subnet Mask: 1111 1111.1111 1111.1111 1111.**0000 0000**
Host Address: 0000 0000.0000 0000.0000 0000.**0000 0001**
In Decimal: **0 . 0 . 0 . 1**

In every properly formed subnet mask, you will notice the first few bits starting with a string of “1”s, and the remaining bits as “0”s. Again, the ones show the network portion of the IP address, and the zeros show the host portion. In this example, the address is 192.168.100.1 with a subnet mask of 255.255.255.0. The network portion is **192.168.100**, and the host portion is **.1**. Because of this subnet mask, 24 bits are used to represent the network, and the last 8 bits are used to represent the host. Remember that with 8 bits, you can represent numbers from 0 (all 8 bits are “0”) to 255 (all 8 bits are “1”). This gives you a clue as to how many hosts you can have belonging to this same network (group of IPs).

What we haven't talked about yet, though, is that you can't quite use all of the numbers from 0 to 255 to assign to hosts. When you define a network, two addresses are reserved. They are referred to as the network address and the broadcast address.

The network address is when the host portion is all zeros. In our example, the network portion of "192.168.100.1 subnet mask 255.255.255.0" is 192.168.100. The host portion is .1. This tells us that we have eight bits for our host portion. If all eight of those bits are zero, that represents the network address, and it cannot be assigned to a device as a legitimate address. In this example, the network address would be 192.168.100.0. Notice that the network portion of the address stayed the same. Only the host portion changed, and it is all zeros. This number identifies the network.

The second address that already has a special purpose is the broadcast address. This is defined when the host portion is all ones. In our example, we know that we have eight bits to use to define a host address. If all of those bits are one, we get this number: 192.168.100.255. Notice, again, that I did not change the first 24 bits. I just changed the last eight, and I made them all ones. The broadcast address is used when you want to send a message to every address in the network.

So, in our example, we have eight bits to use for the hosts. With eight bits, we can represent numbers from 0 to 255 for a total of 256 numbers (remember to include 0). But we can't use "0," because it's the network address. That leaves 255 numbers we can use, but we can't use "255" because that's the broadcast address. That leaves us with a total of 254 numbers to use. We can use any number from 1 to 254. In our example, we used 1 (192.168.100.1). If we wanted to configure another device, and we wanted it to be in the same network, we could assign it 192.168.100.2, or 192.168.100.15, or 192.168.100.213. I think you get the idea. But to be absolutely certain that the IP address would be in the same network, we have to use the same subnet mask (in our example, it would be 255.255.255.0). If you have two addresses that look similar, but they have different subnet masks, they are in different networks.

You can always calculate how many hosts you can have by looking at the number of bits. In our example, we have eight bits for the host portion. You could calculate how many hosts you can use by using this formula: $2^n - 2$. So, if we have eight host bits, that would be $2^8 - 2$. As $2^8 = 256$, that would be the same as $256 - 2$, which equals 254 hosts. The "28" is the total number of hosts we could represent. We subtract two (and always two) for the network and broadcast address.

When TCP/IP was formed, they divided all the possible IP addresses into classes. Each class would be specially defined, and each class would have a default subnet mask.

Table 1. Classful Routing

Class	First Few Bits of the Address	First Octet Range (in decimal)	Subnet Mask	Network	Host per Network
A	0	0-127	255.0.0.0	128	16,777,214
B	10	128-191	255.255.0.0	16,384	65,534
C	110	192-223	255.255.255.0	2,097,152	254
D	1110	224-239			
E	1111	240-255			

Class A. This class is defined by the very first bit in the address being zero. If that first bit is zero, it is a Class A network address. Because the first bit has to be zero, that means any address from 0.0.0.0 to 127.255.255.255 is considered to be Class A. Class A's default subnet mask is 255.0.0.0. This means that only the first 8 bits are used for the network portion, and the last 24 bits are used for the host. This means that, in the same Class A network, you could have $2^{24} - 2$, or 16,777,216 - 2, which equals 16,777,214 hosts. As you can see, this is a class that allows you to have a few different networks, but each network could have a lot of hosts in them. To calculate the number of networks you could create, you would leave out the first bit (as that defines the class). This leaves you with seven bits, and $2^7 = 128$ networks, or the numbers from 0 to 127.

Class B. This class is defined by the very first bit in the address being a one, but the second bit must be a zero. This means that any address from 128.0.0.0 to 191.255.255.255 is considered a Class B. Class B's default subnet mask is 255.255.0.0. This makes it fairly even-handed. You have 16 bits for the network portion, and 16 bits for the hosts. This would mean that you could have $2^{16} - 2$, or 65,536 - 2, which would give you 65,534 hosts. Now, for networks, you have to leave out the first two bits (as they define the class; a one followed by a zero). This leaves you with 14 ($16 - 2$) bits, and $2^{14} = 16,384$ networks. These would be the network portions from 128.0 to 191.255.

Class C. This class is defined by the very first bit and the second bit being one, followed by the third bit being a zero. This means that any address from 192.0.0.0 to 223.255.255.255 is considered to be a Class C. The default Class C subnet mask is 255.255.255.0 (look familiar?). This means that the first 24 bits are the network portion, and the last 8 bits are for the hosts. As we showed in our earlier example, with eight bits you could have $2^8 - 2$, or 256 - 2, which gives you 254 hosts. For networks, however, you have to leave out the first three bits (as they define the class; "110" for the first three bits). This leaves you with 21 ($24 - 3$) bits, and $2^{21} = 2,097,152$ networks. These would be the network portions from 192.0.0 to 223.255.255. As you can see, this gives you a lot of networks with fewer hosts per network.

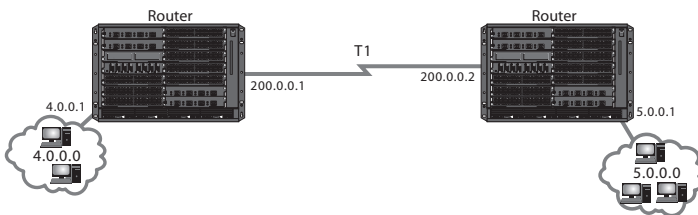
Class D. This class is defined by the first bit, second bit, and third bit all being one, followed by the fourth bit being a zero (do you see a pattern yet?). This means that any address from 224.0.0.0 to 239.255.255.255 is considered to be a Class D. Class D is a special class that is reserved for *multicast* (more about this in Chapter 9). You should not create networks or hosts in this space. It is reserved.

Class E. This class is defined by the first bit, second bit, third bit, and fourth bit all being one. This means that any address from 240.0.0.0 to 255.255.255.255 is considered to be Class E. Class E is also a special class that is reserved. You should not create networks or hosts in this space.

The Internet uses only Classes A, B, and C for address space. As mentioned in their descriptions, Class D and E are reserved for special purposes, and are not to be used as addressable space on the Internet.

For a while, the default subnet masks were the only ones used. This made routing remarkably easy. All a router had to do to determine class was to look at the first few bits of the IP address. It could easily identify Class A, B, C, D, or E. This is referred to as *classful* networking.

But soon, problems started to arise. As the Internet grew in popularity, it became clear that there would not be enough addresses to go around. In addition, there were many addresses that were being wasted. Consider this:



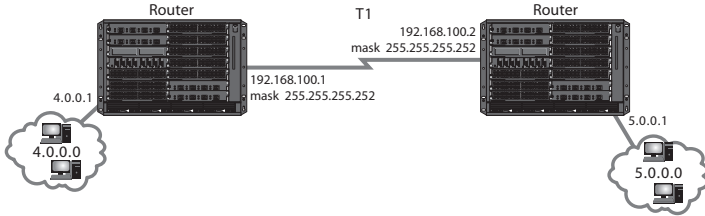
Here we have a T1 line connecting two points. To create an IP network to the connect the two points, we want to use the class with the smallest number of hosts available to it. If you look at the previous table, this would be a Class C (254 hosts). But as you can see, this wastes 252 hosts. We only need two. This is only one example of the problems that occur using only classful networking.

The Subnet

The answer came with the invention of the *subnet*. Simply stated, a subnet is a portion of a network within a network. It allowed for the creation of much smaller networks where needed.

Let's take the example we mentioned in the previous section. You have two endpoints connected by a T1 line. You want to create an IP network to connect the two points, but you don't want to waste addresses. Let's say, you want to use the Class C network 192.168.100.0. Remember that a Class C network has a subnet mask of 255.255.255.0, or 24 bits for the network, and 8 bits for

the hosts. This gives us 254 usable hosts (remember to subtract the network and broadcast address). We only need two hosts, so using the default subnet mask is pretty wasteful. What if we used 255.255.255.252 as the subnet mask? In binary, this would be 1111 1111 . 1111 1111 . 1111 1111 . 1111 1100. This means that 30 bits are used for the network portion and only two bits are left for the hosts. This gives us $2^2 - 2$, or $4 - 2$, which equals 2 hosts. Perfect! So, with the subnet mask of 255.255.255.252, the network address is 192.168.100.0, one host is 192.168.100.1, the other host is 192.168.100.2, and 192.168.100.3 is the broadcast address.



This is referred to as *classless* networking, as you're not conforming to classes. You can subdivide (or subnet) as much as you need. Remember that this is a division. You can't make the subnet mask smaller than the default for the class. For example, you can't define a Class C network to have a subnet mask of 255.255.0.0 (or 16 bits for the network portion). A Class C subnet must have at least 24 bits for the network portion. Likewise, a Class B subnet must have at least 16 bits for the network portion, and a Class A network must have at least eight bits for the network portion.

Let's try another example. Let's say we have a network that will require 400 hosts. If we use a Class C, the largest number of hosts we can have per network is 254. We could use a traditional Class B, but this would give us 65,534 hosts, a lot more than we need. So, let's start with a Class B, say, 172.16.0.0 with a subnet mask of 255.255.0.0. We've got to create a subnet to allow for 400 hosts. We know that eight host bits gives us 254 hosts. How many does nine host bits give us? This would be $2^9 - 2$, or $512 - 2$, which would equal 510 hosts. This is still more than we need, but it's the smallest subnet we can make that will still meet our requirement. If we have nine host bits, that leaves us 23 ($32 - 9$) network bits. With 23 network bits, this gives us a subnet mask of 1111 1111 . 1111 1111 . 1111 1110 . 0000 0000, or, in decimal, 255.255.254.0. So, our final network address is 172.16.0.0 with a subnet mask of 255.255.254.0. The first usable address would be 172.16.0.1. The last usable address would be 172.16.1.254. The broadcast address would be 172.16.1.255.

Table 2. Understanding Subnets

Network Address	Subnet Mask	First Usable Address	Last Usable Address
172.16.0.0	255.255.254.0	172.16.0.1	172.16.1.255
Broadcast Address		Number of Usable Hosts	
172.16.1.255		510	

Why would we want to shrink the host size of a network? Why not just use classful networking? There's another reason. Your host size determines your *broadcast domain*. Remember how we talked about a broadcast address? This is an address that you send a message to that every device in the subnet will receive.

How Does the Network Layer Know Where to Go?

We talked a little bit about ARP (Address Resolution Protocol). ARP is a great example of a broadcasting protocol. Let's say, we're using a Class A network: 10.0.0.0. Because we're using classful networking, its subnet mask would be 255.0.0.0. This means eight bits are for the network portion, and 24 bits are for the host portion. Let's say we have 4,000 devices in this network. When one device wants to talk to the other, the Network Layer on the source machine will first check to see if the destination device is in the same network. If it is not, the Network Layer will check the source machine's *routing table*.

A routing table exists on any Layer 3-capable device. This includes workstations. You don't believe your PC has a routing table? Pull up a command prompt and type: **netstat -rn**.

```
$ netstat -rn
```

```
Kernel IP routing table
```

Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
192.168.222.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	192.168.222.1	0.0.0.0	UG	0	0	0	eth0

This command will work on most modern operating systems. If not, your operating system will have a way for you to view your machine's routing table. The table contains network addresses in one column, and how to get there in the others. It includes the address of a gateway, as well as which interface the frame should be created on, to get to its destination. A gateway is a device that is either connected to both networks directly, or it's a device that knows how to get to a remote network that it's not connected to.

“Why does my PC already have entries in the routing table? I didn't add these!” Technically, when you configured the IP address on your PC, you did. When you configure a NIC with an IP address, say, 10.0.0.4 with a subnet mask of 255.0.0.0, the operating system adds an entry for the 10.0.0.0 network with a subnet mask of 255.0.0.0, and shows that the destination is...the NIC you configured. In this case, the destination doesn't have to be a gateway, because you're connected to the network.

It is also common (though not required) to see a default gateway. This can sometimes be shown as a destination network address of 0.0.0.0. Essentially, it acts as a catch-all. You're basically saying, “Hey, if I don't know how to get somewhere (I don't have the network address in my routing table), send it to this guy.” Typically, the default gateway is a router. By standard convention, the default gateway must be in the same network as the device using it. Why?

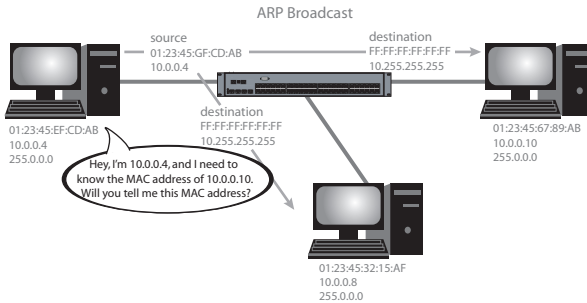
Well, let's go back to our example. We talked about what would happen if the host were trying to communicate to another host that was not in the same network. What if the host were trying to communicate to another host that was in the same network? Let's say we want to reach 10.0.0.10. The Network Layer will now check another table—the ARP table. Most operating systems will show you your machine's ARP table when you type: **arp -a**.

```
$ arp -a
jasper.springfield (192.168.100.2) at 00:0C:BC:49:E2:F0 [ether] on eth0
abe.springfield (192.168.100.3) at 00:04:08:DE:FC:01 [ether] on eth0
gateway.springfield (192.168.100.1) at 02:04:08:10:2C:A0 [ether] on eth0
```

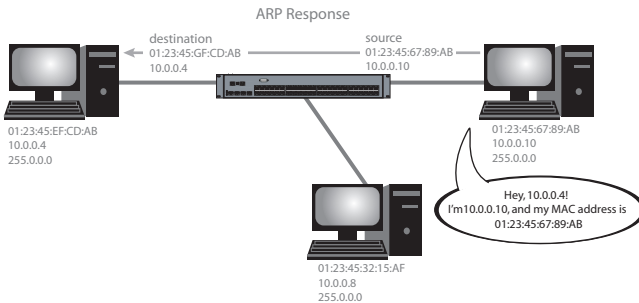
The first column shows you the IP addresses that are in the same network (that your workstation knows about). The second column shows you the MAC address that belongs to each specific IP address. If the destination IP address is listed in the ARP table, the Network Layer knows that if it sends its packet to the Data Link Layer, it will be able to properly create a frame (with the correct destination MAC address).

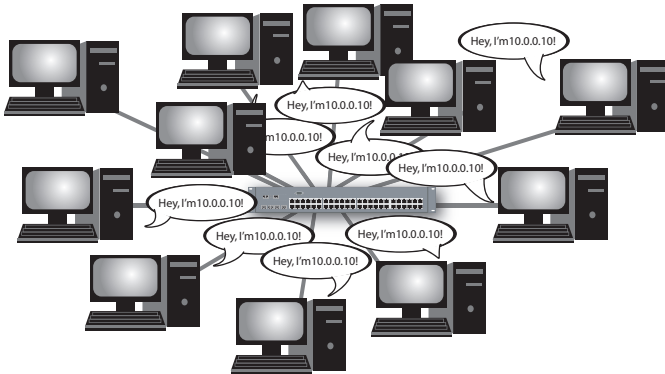
But what if the address you want to communicate with isn't in the ARP table? The Network Layer sends an ARP broadcast. The ARP broadcast is addressed to the broadcast address of the network. In our example, the network is 10.0.0.0 with a subnet mask of 255.0.0.0. This means that the broadcast address for this network is 10.255.255.255. So the ARP broadcast is sent from the workstation (10.0.0.4) to the broadcast address (10.255.255.255). When the Data Link Layer receives this packet (the ARP broadcast), it uses the workstation's MAC address as the source address, but it uses a broadcast MAC address as the destination. The broadcast MAC address sets all 48 of the bits in the address to “1.” In hexadecimal, it looks like this: FF:FF:FF:FF:FF:FF. The broadcast calls out “Hey, I'm 10.0.0.4, and I need to know the MAC address of 10.0.0.10. Will the host 10.0.0.10 send me a reply and tell me your MAC address?” This message is sent to every host in the 10.0.0.0 subnet, or every host in the broadcast domain.

Typically, 10.0.0.10 will now send an ARP response. This message basically says, “Hey, 10.0.0.4! I’m 10.0.0.10, and my MAC address is 01:23:45:67:89:AB.” The response is sent directly to 10.0.0.4, not the whole subnet. How does 10.0.0.10 know the MAC address of 10.0.0.4? Easy. It was in the ARP broadcast that it sent. The host 10.0.0.10, when it received the ARP broadcast message, added the address 10.0.0.4 and its MAC address into its own ARP table. When the ARP response reaches 10.0.0.4, the host 10.0.0.4 will add 10.0.0.10 and its MAC address (01:23:45:67:89:AB) into its ARP table. Now, it can send the packet that it originally wanted to send, knowing that the Data Link Layer will be able to form a proper frame.



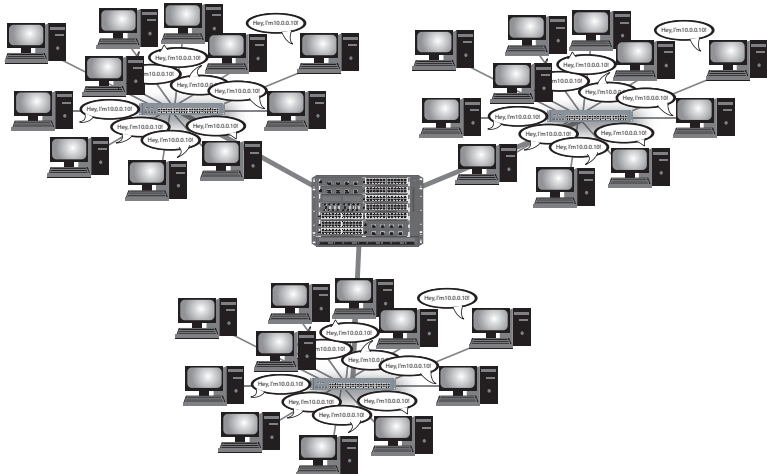
By default, the information stays in the ARP table for five minutes. If the information is used, the timer is reset. After five minutes, it is removed from the table. If the host wants to communicate with it again, it will need to send another ARP broadcast. Now, when you're dealing with three or four workstations, this may not seem like a big deal. But picture 4,000 workstations.

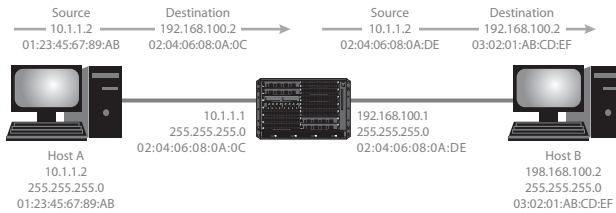




ARP is not the only broadcast protocol, but it is very commonly used. You can begin to see why it might be a good idea to break some of these nodes up into multiple broadcast domains.

If the workstation wants to send a packet to a destination that is not on the same network, the packet will still show the source and destination IP addresses, but the Network Layer will instruct the Data Link Layer to use the destination MAC address of the gateway. This way, the gateway can pass the packet to where it needs to go. The gateway will strip off the frame header and trailer it received, and it will create a new frame header and trailer containing its MAC address as the source, and the destination IP address' MAC address as the destination. Perhaps we need another illustration.





This is why the gateway needs to be part of the same network as the host. The host needs to know how to get to the gateway. It will communicate to the gateway using its MAC address. If it doesn't know the MAC address, it sends an ARP broadcast. If the gateway is not in the same network (or broadcast domain), the gateway will never see the ARP broadcast. Broadcasts never leave the broadcast domain. If the gateway never sees the ARP broadcast, it will never reply to it. The gateway must have an address in the same network as the hosts that use it.

CIDR

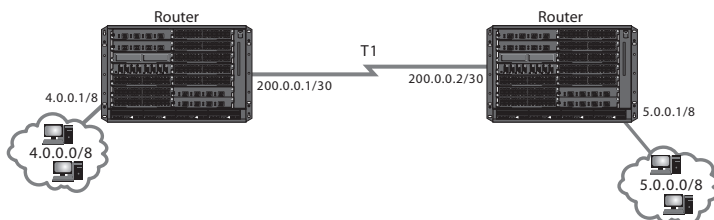
With classless networking, there came a new way to label a network. With classful networking, you could see a network address of, say, 10.0.0.0, and you would know that this was a Class A network, and therefore, its subnet mask must be 255.0.0.0. With classless networking, you could have a subnet length of eight bits, 13 bits, 29 bits, almost anything! A new naming schema had to easily reflect the subnet mask.

Enter Classless Inter-Domain Routing (CIDR). It's usually pronounced like "cider." The convention states that you can represent the number of network bits in a subnet mask, by adding a number after a slash ("/") character. For example, 10.0.0.0, in classful networking, has a subnet mask of 255.0.0.0 (Class A). This means that the network portion has eight bits. Using CIDR, we would represent this same network like this: 10.0.0.0/8. Notice that I used the network address (10.0.0.0), followed it by a slash (/), and then followed that by the number of network bits in the subnet mask (8, in this example).

Using CIDR notation, you can easily represent any subnet. Let's look back at a couple of our previous examples. Remember the network that needed 400 hosts? We came up with a subnet mask of 255.255.254.0, or 23 network bits. We used a Class B network to start with-172.16.0.0. The CIDR notation would be 172.16.0.0/23. Or, how about the network where we only needed two hosts? We started with a Class C network, 192.168.100.0. We decided on a subnet mask of 255.255.255.252, or 30 network bits. The CIDR notation would then be 192.168.100.0/30.

Classless networking is the default today. Get to know CIDR notation well. It is not only used throughout this book, it's used throughout the networking industry. It's a concise way to indicate the network address and the subnet mask of a network.

CIDR notation can also be used to indicate a specific host's subnet mask. Remember our two-host network? The CIDR notation is 192.168.100.0/30. The specific hosts can be represented this way: 192.168.100.1/30 and 192.168.100.2/30. This not only indicates their specific address, but it also tells us what their subnet mask is.



As there is a subnet, there is also such thing as a *supernet*. This just allows you to group multiple addresses together, using CIDR notation. For example, let's say you're using 172.16.0.0/24, 172.16.1.0/24, 172.16.2.0/24, and 172.16.3.0/24. Instead of individually referring to all four networks, you could simply refer to 172.16.0.0/22. You would not have assigned this as a subnet mask to hosts. It is purely used to refer to large groups of networks. Supernets are not restricted to the minimum subnet mask length. For example, if I wanted to represent 8.0.0.0/8, 9.0.0.0/8, 10.0.0.0/8, and 11.0.0.0/8, I could refer to the group as 8.0.0.0/6. Notice that the "6" is shorter than the minimum subnet length of a Class A ("8"). This is okay, as the "6" is not really a subnet mask length, but a mask length to summarize all of the subnets within the range. This will come into play more when we start talking about routing protocols in Section III.

Private Networks (RFC 1918)

With classless networking, the Internet could grow and grow, but what about organizations that may want to use TCP/IP, but may not want to use the Internet? For example, a business may have 500 workstations that their employees use to communicate locally (e.g., to file servers, internal servers, each other, etc.). If the business wanted to use TCP/IP for its network, it would have to register a network range with its proper registry for Internet numbers (see [Chapter 13](#)) in order for each workstation to have an IP address. Also, once addressed, each workstation could be accessible from anywhere on the Internet. This may not be desirable, if you have sensitive information on some of your servers or workstations.

The Internet Engineering Task Force (IETF) publishes a set of standards called Request for Comments (RFC). These detail all kinds of different standards for the Internet. Among them is RFC 1918: Address Allocation for Private Internets. This addresses (no pun intended) the problem by allowing private networks to be created. You can still use TCP/IP, but you don't have to worry about registering addresses with an Internet registry.

RFC 1918 specifies a network range in each of the three addressable classes (A, B, and C).

Table 3. RFC 1918

Class	Address	Range
A	10.0.0.0/8	10.0.0.0 - 10.255.255.255
B	172.16.0.0/12	172.16.0.0 - 172.31.255.255
C	192.168.0.0/16	192.168.0.0 - 192.168.255.255

Because of the standard, Internet Service Providers (ISPs) know not to route these particular addresses across the Internet. They are private. They are meant to only be used locally. Today, many homes and business use these ranges of addresses. It's important for you to be familiar with them. You will use them quite often. Memorize the above table.

Layer 4 TCP/IP: TCP/UDP

We've made it all the way to Layer 4 (Transport). In the TCP/IP stack, the two most common Transport Layer protocols are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). These are not the only Transport Layer protocols in the TCP/IP stack, but these are the ones that you'll be working with the most. Let's get started.

First off, I should mention that the Transport Layer (Layer 4) also has an addressing method. Layer 2 uses MAC addresses. Layer 3 uses IP addresses. Layer 4 uses *ports*. A port is a 16-bit number, meaning that it can represent a number from 0 to 65,535. This part of TCP and UDP actually has a little hint of being related to the Session Layer (Layer 5), because ports keep track of the conversation.

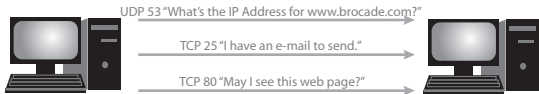
Each port number represents a type of communication. Many ports are associated with specific Application Layer protocols. For both TCP and UDP, this port number range is 1 to 1023. These are services that are registered with the IANA. They are often referred to as *well-known ports*. Ports above 1023 are open, but, for commercial applications that use them, it is most proper to register these with the IANA as well.

Table 4. Well-Known Ports

Port	Service
TCP 21	FTP
TCP 22	SSH
TCP 23	Telnet
TCP 25	SMTP
TCP/UDP 53	DNS
UDP 67	DHCP
TCP 80	HTTP
TCP 110	POP3
UDP 161	SNMP
TCP 443	SSL

I should mention that just because a port number is registered to a particular Application Layer protocol, it does not mean that you must use that application protocol. For example, it is possible to use the SSH Application Layer protocol using TCP 80, instead of TCP 22. It is possible to use HTTP using TCP 25, instead of TCP 80. It is possible to run SMTP using TCP 443, instead of TCP 25; however, if you did this, very few mail servers would be able to communicate with your server properly. They would expect to be communicating with you on TCP 25. But the point is, it is possible. Just because a port is well-known or registered to a particular application does not mean that only that application can be used.

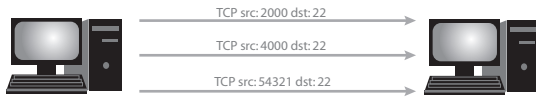
With ports, one host (one IP address) may provide several services simultaneously. Say, for instance, that you wanted one host to be able to listen on TCP 80 (HTTP), TCP 25 (SMTP), and UDP 53 (DNS). With ports, this is certainly possible. From the same client, you can address the same destination host, but take part in multiple conversations in different services.



What if a host wanted to initiate two (or more) conversations to the same host using the same service? You don't think this comes up? Ask any Systems Administrator. Let's say, you want to use SSH (TCP 22) to remotely connect to a server. You've started a process in your first session, and you'd like to start a second SSH session to the same server to perform another task. Both connec-

tions will attempt to connect on TCP 22. Both connections will have the same source IP address and the same destination IP address. How will either machine know which conversation is which?

First off, the connection from the client to host uses TCP 22 as its *destination port*. On the Transport Layer (just like addressing on the lower layers), you need a *source port* and a destination port. What actually happened when you made that SSH connection is that the client used TCP 22 for its destination port, but it used another port number for its source port. The source port number is a random number between 1024 and 65,535 that is chosen by the operating system. Every time a new Transport Layer session is initiated, a new source port is chosen. So, in our example, the first SSH connection will use one source port, and the second SSH connection will use a different source port.



The source port is also used so that the destination host knows how to talk back to the client. Just like in Layer 3 (IP) and Layer 2 (MAC) addressing, the client will initiate the connection to the destination host using a source port and a destination port, and the destination host will reply by reversing the source and destination. This is probably seen best in an illustration.



In this example, the client is initiating a SSH session to the server. The client's source port is TCP 1024 and its destination port is TCP 22. When the server replies, it uses a source port of TCP 22 and a destination port of TCP 1024.

A *session* is distinguished by the following items:

- Source IP address
- Destination IP address
- Source port
- Destination port
- Protocol (e.g., TCP or UDP)

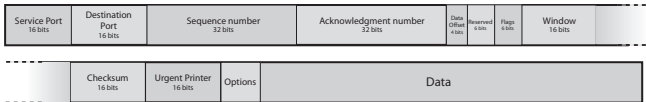
Both TCP and UDP use source and destination ports to distinguish the service and multiple sessions. Now let's see what makes them different.

Transmission Control Protocol (TCP)

TCP is a connection-oriented protocol. As you may recall from the last chapter, this means that it goes to great lengths to make sure that the data gets to its destination. Remember our Transport Layer discussion in Chapter 1? To get its job done, TCP uses acknowledgments, windowing, and flow control. As a refresher, acknowledgments are used to make sure the data was delivered. Windowing tells the session how many segments should be sent before an acknowledgment is required. Flow control prevents the data from being sent too fast.

In every TCP session, there is a *client* and a *server*. The client is the host that initiated the connection. The client is requesting something from the server. Think of a client as a user going to a Web page. The client is the user requesting the Web page. The server is the host that receives the request, and delivers the data requested.

TCP adds a header to the data it receives from the Session Layer. It's typically 160 bits long, and it looks like this:



Source Port. This is a 16-bit number. As we mentioned before, the source port is used to help distinguish one session from another.

Destination Port. This is a 16-bit number. This also participates in distinguishing one session from another, but it is most well known for defining what kind of Application Layer communication we intend to do.

Sequence Number. This is a 32-bit number. Each of the segments is numbered. This is so the segments can be properly reassembled when they are received.

Acknowledgment Number. This is a 32-bit number. When using windowing, this number is the next sequence number that the server expects.

Data Offset. This is a 4-bit field. It specifies the size of the TCP header. The number is in 32-bit words. The minimum size of the TCP header is 5 words (160 bits; $5 \times 32 = 160$). The maximum size is 15 words (480 bits; $15 \times 32 = 480$).

Reserved. This is a 6-bit field that is reserved for future use. All bits in this field should be zero.

Flags. This is a 6-bit field. Each bit represents a special message in TCP.

- **URG** Urgent pointer
- **ACK** Acknowledgment
- **PSH** Push function
- **RST** Reset the connection
- **SYN** Synchronize the sequence numbers
- **FIN** Request to close the connection (finished)

Window. This is a 16-bit number. It is the size of the window in bytes.

Checksum. This is a 16-bit number. This provides error checking for TCP. Each segment has a checksum that allows the client to verify the data.

Urgent Pointer. This is a 16-bit number. If the URG flag is set, this number acts as an offset to the sequence number.

Options. This is an optional field that must be a multiple of 32-bits. It is used in conjunction with the Urgent pointer. It provides additional settings.

Data. This is the data received by the Session Layer.

The 3-Way Handshake

TCP uses what's referred to as a "3-way handshake" to initiate a conversation. Let's start by showing you what it looks like.



The client sends a TCP SYN segment. The SYN is short for *synchronize*. It synchronizes the segment sequence numbers between the client and the server, but it also initiates the session. The first packet (the SYN) determines what destination port you're communicating with, and what source port the server should reply on.

The server sends back a TCP SYN/ACK segment. To me, this is really a misnomer. It might be easier to think of this as an "ACK/SYN" packet. The "ACK" part is *acknowledging* the SYN that the client sent. The "SYN" part is the server's SYN packet back to the client. It's as if the client is saying, "Hey, do you want to talk?" And the server replies, "Yeah, I want to talk. Do you want to talk?"

And the client replies, "Yeah." He does this by sending the third segment: an ACK. An ACK, put simply, is an "I got it" segment. It's TCP's method for communicating that you've received a segment.

In summary, the 3-way handshake looks like this:

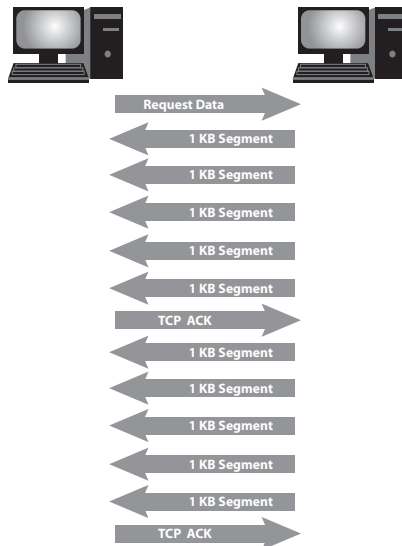
1. Client sends TCP SYN to server
2. Server sends TCP SYN/ACK to client
3. Client sends TCP ACK to server

Windows and Acknowledgments

Now you've initiated the session. It's time to start communicating. Here, the client requests some data. The server sends an acknowledgment (ACK) to tell the client that it has received the request. Now the server will start sending segments.

In our example, let's say that the server is going to send 20 kilobytes (20,480 bytes) of data. The Transport Layer, on the server side, is going to divide that 20 KB into several segments. It's going to number the segments so that the client will know how to reassemble them, and to make sure that it received them all. When the server starts sending segments, it will send a few at a time and wait for an acknowledgment from the client. This is the window.

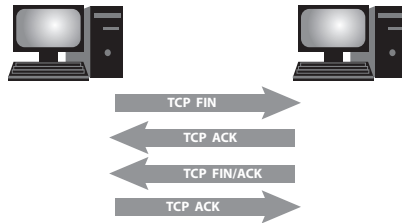
The window is measured in bytes, not segments. The server will send segments until it has reached the window size. In our example, let's say that the window size of the server is 5,120 bytes (5 KB). That means that the server will send segments until it has sent 5 KB worth of segments. Then it will wait to receive an acknowledgment (ACK) from the client. When it does, it will send another 5 KB worth of segments, and so on.



When it is finished communicating, the client will send a TCP FIN (request to close) message to the server. The server will send an ACK to acknowledge the FIN. The server will send a message to its application to close the session, and then it will send a TCP FIN/ACK segment to the client. Finally, the client will reply with an ACK segment to the server. It's as if the client is saying, "I want to stop now, okay?" The server replies with, "Okay," followed by, "I'm stopping now, okay?" And finally, the client responds with, "Okay." And this closes the session. Any more communication between these two devices would result in a new 3-way handshake and, thus, a new session.

In summary, the way TCP closes a session looks like this:

1. Client sends TCP FIN to server
2. Server sends TCP ACK to client, to acknowledge the FIN
3. Server sends TCP FIN/ACK to client
4. Client sends TCP ACK to server



User Datagram Protocol (UDP)

UDP is a connectionless protocol. This means that there is no 3-way handshake, acknowledgments, windows, nor flow control. If the receiver gets the segment, great. If it doesn't, too bad. We talked about this in Chapter 1, but it bears repeating. The reason that connectionless protocols are used is that they're very, very speedy.

Think about TCP. You spend three segments just starting the session. You spend untold number of segments sending acknowledgments. You spend time waiting for acknowledgments. It even takes four segments just to finish. Oh, sure, your data gets there, and you know it gets there, but at a cost.

In reality, in a perfect world, every protocol would be connectionless. It's fast and it's based on trust. How do I know my packet got to my destination? Because I sent it! That's how I know! Sadly, of course, networks just aren't that reliable. To be fair, networks are a lot more reliable now than they used to be, and on a well-designed Ethernet switched network, UDP works great! Still, always keep this in mind: UDP is fast, but you might lose a few segments; TCP is slower, but it is safe.

UDP adds a Transport Layer header to create its segment, similar to TCP. Let's look at UDP's header:

Source Port 16 bits	Destination Port 16 bits	Length 16 bits	Checksum 16 bits	Data
------------------------	-----------------------------	-------------------	---------------------	------

Source Port. This is a 16-bit number. You wouldn't think this would be necessary with a connectionless protocol, and you're right. It's not...always. If no reply is expected, this number can be zero. If a reply is expected, it should be the port number on which to expect the reply. It is very common for UDP protocols to use the same source and destination ports for their communication.

Destination Port. This is a 16-bit number. This can help distinguish which Application Layer protocol you're communicating with.

Length. This is a 16-bit number. This number is the length, in bytes, of the entire segment (including the header). The minimum length is 8 bytes, as this would be the minimum size of the header with no data.

Checksum. This is a 16-bit number. Believe it or not, this is a checksum of the entire segment. Wait a minute! Isn't UDP supposed to be unreliable? Yes, but it does provide a checksum to each of its segments. The checksum will verify whether or not data has been altered or damaged in transport. What is done about that really depends on the Application Layer. There are some Application Layer protocols that use UDP, but implement some reliability at Layer 7. Likewise, there are some Application Layer protocols that will even ignore the checksum.

Data. This is the data received by the Session Layer.

Dynamic Host Configuration Protocol (DHCP)

It's hard to have a conversation about UDP without talking a little bit about DHCP. DHCP is a protocol that is used to automatically configure a host with an IP address, subnet mask, default gateway, DNS configuration, and other network-related configurations. It uses UDP to do its job.

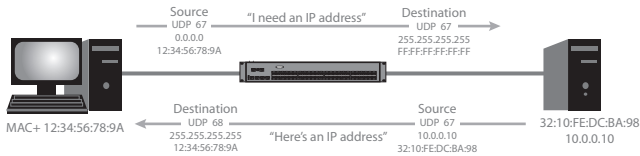
For DHCP to work, you need to have a *DHCP* server. This can be an actual server, a router, or any other network device that has DHCP-serving capability. The idea is that the DHCP server keeps a bank of usable IP addresses within its subnet. When it receives a request, it *leases* out an address to the client. It keeps track of the life of the lease (which is configurable; it could be a week; it could be five minutes). When the lease is at its half-way point, the server sends a packet to the client asking if it would still like to keep using the address. If it does, the lease timer is reset.

Let's say you have a client that wants to receive an IP address. It needs to create a message to ask for one. DHCP uses UDP, so we have the Transport Layer figured out. It sends a segment with a destination port of 67 (well-known port for DHCP request).

What about the Network Layer? What will it use for an address? For a source address, it will use, well, nothing: 0.0.0.0. For a destination address, it uses the universal broadcast address: 255.255.255.255. This is a broadcast address no matter what subnet you may find yourself in.

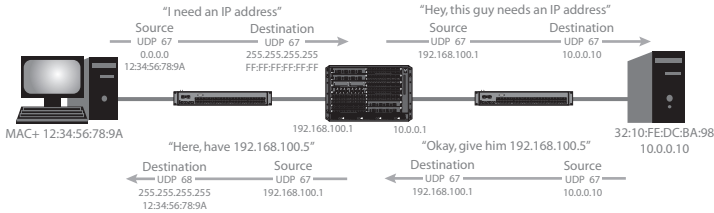
And what about the Data Link Layer? The frame will use the client's MAC address (because that's burned into the NIC's ROM, remember?) as a source MAC address, and it will use the broadcast MAC address, FF:FF:FF:FF:FF:FF, as the destination MAC address.

The reply is sent on destination UDP port 68 (well-known port for DHCP reply). The DHCP server uses its own IP address and MAC for the source addresses. On the Network Layer, it uses 255.255.255.255 as the destination address, but it uses the proper destination MAC address in the frame. It learned the MAC address from the DHCP request. The reply contains all of the information requested, which usually includes at least an IP address, subnet mask, and default gateway.

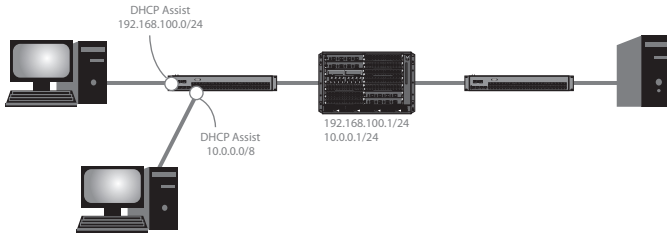


But notice, for the client to use DHCP, it must send a broadcast. This means that a DHCP server must have an address in the broadcast domain. What if you have several broadcast domains. Does this mean you have to have a DHCP server in each domain?

Well, you can do that, but most engineers find that using a DHCP *helper* address is the better way to go. This acts as kind of a DHCP proxy. The helper address is usually configured on the broadcast domain's gateway. Essentially, you're telling the router that it should listen for DHCP broadcasts. If it sees one, it should encapsulate it and send it to the DHCP helper address (wherever that may be), which is the address of the remote DHCP server. The DHCP server sends an encapsulated DHCP reply to the router, and the router sends the reply to the client. The client will not know that its broadcast ever left the broadcast domain.



Now, what if you have one router that is serving as a gateway for several broadcast domains? You can still use a helper address, but how will the router keep track of which broadcast domain made the request? This is where you need DHCP Assist. This is configured on a Layer 2 switch. In a nutshell, you configure the switch port that belongs to a particular broadcast domain to forward DHCP requests stamped for their particular broadcast domain. Let's look at a picture.



When the request is received by the Layer 2 switch (configured with DHCP Assist), it stamps the request so that the router knows which broadcast domain made the request. Without this, the router may end up responding with addresses in the wrong broadcast domain! This doesn't come up very often, but when it does, you'll be glad to know it's there.

Network Address Translation (NAT)

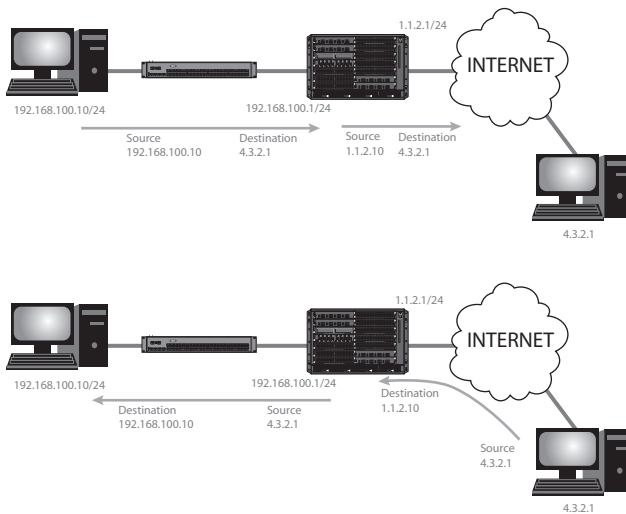
Earlier in the chapter, we talked about private IP networks that are defined by RFC 1918. These addresses can be used by anyone, but they can never be routed across the Internet. This seemed like a good system. After all, you want your network to be private, right? It's funny how consumers change their mind.

With Network Address Translation (NAT), you can use your private network locally, *and* you can use the Internet. NAT would usually be configured on a router that sits between your private network and the destination network (whether that is the Internet, a newly-merged business, etc.). NAT does not have to be an RFC 1918 address being translated to a publicly-routable Internet address. It is simply one address being translated into another.

The router is performing a *swap*. Let's say for example that you're using the RFC 1918 network 192.168.100.0/24. You have a client: 92.168.100.10. You also have a connection to the Internet, and a publicly-routed range of IP addresses: 1.1.2.0/24. If your client needs to get out to the Internet, you could configure your router to translate 192.168.100.10 to a public address, for example, 1.1.2.10. Now, every time the client attempts to go out to the Internet, the router will translate the client's private address (192.168.100.10) to the translated public address (1.1.2.10). Likewise, when replies or initial connections come in from the Internet to 1.1.2.10, the router will translate them to 192.168.100.10. Notice that I chose to keep the last octet of the client's private IP address and its public IP address the same ("10").

This was completely arbitrary. I could just as easily have chosen “1.1.2.25” or “1.1.2.113.” For the functionality of NAT, it doesn't matter. Keeping the last octet the same is a good organizational tactic, but it won't always work in all situations.

But anyway, how does the router do this translation? Well, a router is a Layer 3 device. It receives a packet bound for some destination (in this example, the Internet). If NAT is configured, the router will examine the IP header of the packet. If the source address field in the IP packet header shows an IP address that needs to be translated, it simply overwrites the source field and puts the public address in it. Likewise, when the router receives an inbound packet (from the Internet, in this example), it will examine the destination IP address in the IP header. If it matches the public address, the router will overwrite the destination address in the header with the private address.



The important thing to remember with NAT is that it's a one-to-one translation. One public IP address translates to one private IP address (and vice-versa). The term “NAT” is used very loosely these days. Keep reading to understand why I would say that.

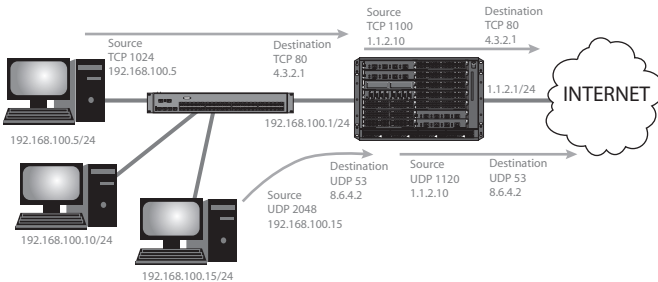
You can also configure a NAT pool. This is a group of public addresses. When a client needs to have its address translated, the router will choose an address out of the NAT pool (a range of public addresses). This allows several private users to be translated.

Port Address Translation (PAT)

Many times, when people say “NAT,” they’re actually referring to “PAT.” Port Address Translation takes a different approach. Instead of being a one-to-one translation, it can be a many-to-one translation. PAT has enjoyed a huge increase in usage, due to the popularity of broadband Internet access (DSL, Cable modems, etc.).

Instead of just paying attention to the IP addresses (Layer 3), PAT also pays attention to the source and destination port (Layer 4). When a packet that needs translation enters the router, it creates a table and records the original source and destination IP addresses, as well as the source and destination (Layer 4) ports.

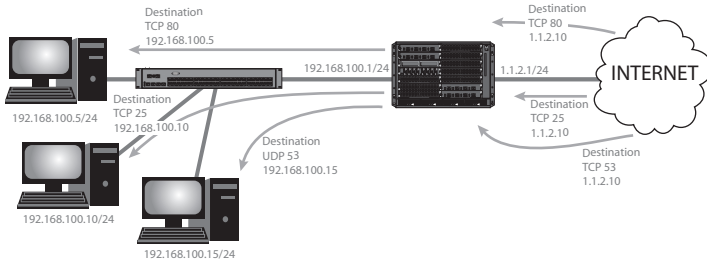
When a packet (that needs to be translated) comes through the router, the router will record the source and destination IP addresses and source and destination Layer 4 ports in the translation table for later reference. It will then translate the source IP to its public PAT address. It will also replace the source Layer 4 port with a new, randomly-chosen port. The router will remember that when it receives a reply on that randomly-chosen source port, it will translate the port to the original source port before handing it back to the original sender. This method allows for thousands of private IP addresses to share the same single public IP address.



This is all well and good for outbound traffic, but what about inbound? Let's say you set up PAT on our Internet-facing router. Let's say you have a workstation and three servers, each with private IP addresses, behind the router. All four devices are sharing the same public PAT address. What would happen if someone on the Internet tried to initiate traffic to the public PAT address (not reply, initiate)? The router wouldn't know which machine to send it to. The traffic would be dropped.

Staying with the previous example, let's say you didn't want anyone on the Internet to initiate traffic to the workstation, but you did want Internet users to be able to reach the servers. One server is a Web server listening on the well-known HTTP port TCP 80. Another is a mail server listening on the well-known SMTP port TCP 25. The third server is a DNS server listening on the well-known DNS port UDP 53. To do this, we need to be able to tell the router, “Hey, if you

receive any traffic bound for the public PAT address using TCP destination port 80, forward it to the Web server. If you receive any traffic bound for the public PAT address using TCP destination port 25, forward it to the mail server. If you receive any traffic bound for the public PAT address using UDP destination port 53, forward it to the DNS server.” And that’s essentially what you do with PAT.



What if you wanted to add a second Web server into the mix? Well, you’re out of luck. You can only forward one incoming Layer 4 port to one destination. One way that you could get around this is to use a different port (say TCP 81), and have it translate back to the second web server on TCP destination port 80. The problem with this method is that you’d have to train your users to access the website by specifying port 81 when they put it in their browsers. In the end, you’re best served by using NAT in that scenario (giving each web server its own public NAT address).

To summarize, the whole point of NAT and PAT is to be able to translate IP addresses to appear to come from another network. NAT allows you to accomplish this at Layer 3, using a one-to-one relationship. PAT allows you to accomplish this at Layer 4, using a many-to-one relationship.

Packet Capturing

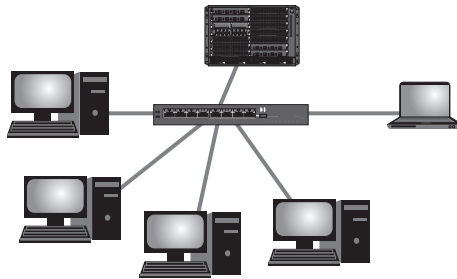
Whether you’re trying to troubleshoot a network problem, or just trying to understand your network’s normal traffic better, packet capturing is your friend. The idea behind packet capturing is to let you see the packets as they traverse the network. This is usually done with packet capturing software.

The term “packet capturing” is a bit of a misnomer. All of the packet capturing software packages that I know actually capture frames (Layer 2). This, of course, includes the frame header, the packet header, the segment header, and the original data generated by Layer 7. Even though “packet capture” is the popular term, *you’re actually capturing frames*.

Picture three servers plugged into an Ethernet hub (not a switch, a hub). You’re the network administrator, and you want to view how the three servers are communicating among themselves. You plug your laptop into the hub. You know that a hub receives a frame, and then sends a copy of the frame out every one of its ports. You also know that, by default, your laptop’s NIC will only

pay attention to frames bound for its MAC address (or broadcast frames). If only there was a way to temporarily tell your laptop's NIC not to ignore any packets...

There is a way. This is called *promiscuous mode*. When your NIC is in promiscuous mode, it will listen to any frame. Because your laptop is plugged into a hub, it will now see every frame that the hub sees. Your laptop will not respond to the frames, but it won't ignore them. Now you can run packet capturing software on your laptop, and view all of the frames that are going through the hub.



What if you take the advice of friends (and this book) and replace your hub with a switch? Sure, the server communication will be more efficient, but what about being able to capture packets? The switch is smart enough to only forward packets to your laptop that are bound for the laptop's MAC address (or broadcast frames). Even with your laptop's NIC in promiscuous mode, this is all it's going to see.

Brocade includes a wonderful tool in all of its switches. It's called a *mirror port*. The idea of a mirror port is to give the administrator the ability to capture packets. You configure the switch port that you've got your laptop plugged into as a mirror port. You then configure the other switch ports you want to watch, and tell the switch to send a copy of every frame it sees on those ports to the mirror port. Now, with your laptop's NIC in promiscuous mode, you're back in business.

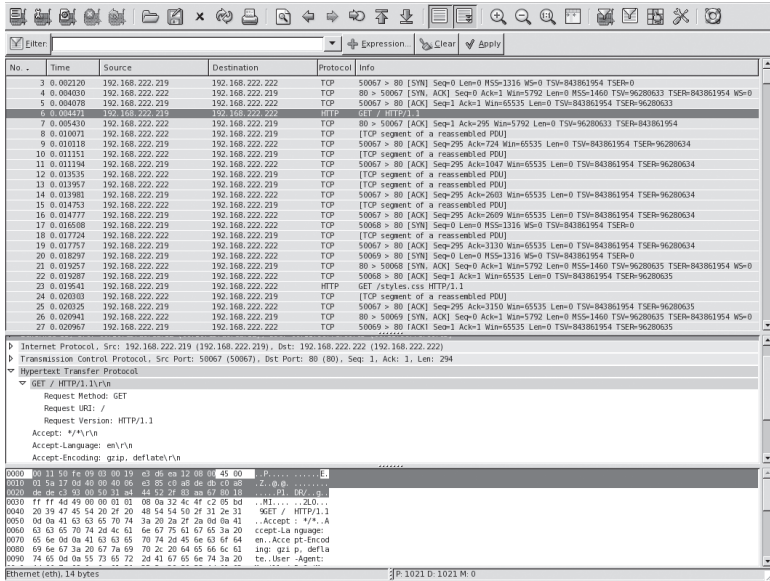
You do have one potential problem. If you set your mirror port to watch all three server ports (in this example), you'd best hope that the total traffic generated by the servers will fit in your mirror port. For example, if all the ports in this example were 100 Mbps, and your servers were using all of the capacity of their NICs at the same time, this would be 300 Mbps of traffic. The mirror port is also 100 Mbps. In this scenario, approximately 2/3 of your frames would be dropped. This is called oversubscribing your mirror port.

There are many software packages available that provide packet capturing. Two of my favorites are tcpdump (<http://www.tcpdump.org/>) and wireshark (<http://www.wireshark.org/>). Both of these software packages are open-source, and freely available to download from their respective web sites.

Tcpdump is a command-line text-based packet capturer. It's my personal favorite. You get a lot of power and flexibility out of a relatively small package. It will automatically put your NIC into promiscuous mode. It will allow you to set up *capture filters*, so that it only captures traffic that you're interested in. It can display the frames as it sees them, or it can dump them to a file for later analysis. Tcpdump is very commonly found on UNIX-based operating systems. Some operating systems even include tcpdump in the initial default install.

```
$ sudo tcpdump -i en1 host 192.168.222.222
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en1, link-type EN10MB (Ethernet), capture size 96 bytes
23:25:41.797578 IP 192.168.222.219.50119 > 192.168.222.222.http: S 2468845578:2468845578(8) win 65535 <seq 1316,nop,wscale 0,nop,nop,timestamp 843863445 0,sack0,K,eq>
23:25:41.799274 IP 192.168.222.222.http > 192.168.222.219.50119: S 1560305123:1560305123(8) ack 2468845579 win 5792 <seq 1460,sackOK,timestamp 96355196 843863445,nop,wscale 0>
23:25:41.799336 IP 192.168.222.219.50119 > 192.168.222.222.http: . ack 1 win 65535 <nop,nop,timestamp 843863445 96355196>
23:25:41.799799 IP 192.168.222.219.50119 > 192.168.222.222.http: P 1:295(294) ack 1 win 65535 <nop,nop,timestamp 843863445 96355196>
23:25:41.801785 IP 192.168.222.222.http > 192.168.222.219.50119: . ack 295 win 5792 <nop,nop,timestamp 96355196 843863445>
23:25:41.805637 IP 192.168.222.222.http > 192.168.222.219.50119: P 1:724(723) ack 295 win 5792 <nop,nop,timestamp 96355196 843863445>
23:25:41.805683 IP 192.168.222.219.50119 > 192.168.222.222.http: . ack 724 win 65535 <nop,nop,timestamp 843863445 96355196>
23:25:41.806891 IP 192.168.222.222.http > 192.168.222.219.50119: P 724:1047(323) ack 295 win 5792 <nop,nop,timestamp 96355196 843863445>
23:25:41.806925 IP 192.168.222.219.50119 > 192.168.222.222.http: . ack 1047 win 65535 <nop,nop,timestamp 843863445 96355196>
23:25:41.809613 IP 192.168.222.222.http > 192.168.222.219.50119: . 1047:2351(1304) ack 295 win 5792 <nop,nop,timestamp 96355197 843863445>
23:25:41.809725 IP 192.168.222.222.http > 192.168.222.219.50119: P 2351:2603(252) ack 295 win 5792 <nop,nop,timestamp 96355197 843863445>
23:25:41.809751 IP 192.168.222.219.50119 > 192.168.222.222.http: . ack 2603 win 65535 <nop,nop,timestamp 843863445 96355197>
23:25:41.810584 IP 192.168.222.222.http > 192.168.222.219.50119: P 2603:2606(3) ack 295 win 5792 <nop,nop,timestamp 96355197 843863445>
23:25:41.810611 IP 192.168.222.219.50119 > 192.168.222.222.http: . ack 2606 win 65535 <nop,nop,timestamp 843863445 96355197>
23:25:41.811268 IP 192.168.222.222.http > 192.168.222.219.50119: P 2606:2608(2) ack 295 win 5792 <nop,nop,timestamp 96355197 843863445>
23:25:41.811293 IP 192.168.222.219.50119 > 192.168.222.222.http: . ack 2608 win 65535 <nop,nop,timestamp 843863445 96355197>
23:25:41.811965 IP 192.168.222.222.http > 192.168.222.219.50119: P 2608:2609(1) ack 295 win 5792 <nop,nop,timestamp 96355197 843863445>
23:25:41.811989 IP 192.168.222.219.50119 > 192.168.222.222.http: . ack 2609 win
```

Wireshark used to be known as Ethereal. It is very similar in functionality to tcpdump, but it adds a graphically-based interface. It still automatically puts your NIC in promiscuous mode. It has capture filters and can dump frames to files. The big difference is that through windows and menus, it provides a much more graphically-rich analysis tool for the administrator.



Both programs are available to run on most popular operating systems. They're very valuable tools to have in your tool belt as a network administrator or network engineer.

sFlow (RFC 3176)

Brocade actually gives us one more very valuable tool for capturing frames. sFlow is an open standard defined by RFC 3176. It is a protocol for capturing a sampling of frames and transmitting that information to a central location (usually, a server) for analysis. The beautiful part? Brocade integrates this protocol into the ASIC (hardware) of every port on their switches. This means that any of the ports of a Brocade switch can be monitored through sFlow, without increasing the resource load on the switch.

With sFlow integrated into the ASIC, the switch's CPU is not involved, and can focus on its primary functions. sFlow uses an adjustable *sample rate*. This is a number that tells sFlow how many packets (of all the packets) to capture. This rate can be as high as 1 packet out of every 2,147,483,648 packets, or as low as 1 packet out of every 2 packets. You can also change the size of the sample. For example, using 128 bytes as your sample size will typically show you all of the headers, enough to see what type of traffic is being sent. You may want to see part of the data, so you can increase the size.

You'll typically need some software running on a server to collect the sFlow data. While there are a few generally available programs that will do this, Brocade has included this function in their IronView Network Manager (INM) software. Brocade INM also supports graphs and other reporting functions based on the sFlow data. This allows a network administrator to get a really good view of what kind of traffic is traversing the network (and how much of each kind of traffic).

Summary

- Internet Protocol (IP) is the Network Layer protocol for TCP/IP
 - ICMP is a Layer 3 protocol used to send messages between Layer 3 IP devices
 - Ping (or “echo request” and “echo reply”) is a function of ICMP that allows you to see if a host is active on the network
 - Address Resolution Protocol (ARP) matches an IP address with a MAC address
- Computers (and routers) use binary
 - A byte is eight bits
- IP addresses are divided into Classes
 - The subnet mask defines what part of the IP address is the network address, and what part is the host address
 - IP network addresses can be further subdivided into subnets
 - The subnet mask can be written using CIDR (e.g., 192.168.100.0/24)
 - CIDR uses the network address followed by a slash (/) and the number of bits that make up the subnet mask
- All IP devices have an ARP table that shows all the IP-address-to-MAC-address translations that the host knows about
- All IP devices have a routing table which lists a network (including subnet mask), and the IP address of the gateway to that network
- Some IP networks are private; they cannot be routed through the Internet
- The most common Transport Layer protocols in TCP/IP are TCP and UDP
- TCP and UDP use ports to distinguish their services and as a form of addressing
- TCP and UDP ports from 1 to 1,023 are well-known ports
- TCP and UDP use a source port (usually a random number from 1,024 to 65,535) and a destination port
- TCP is connection-oriented

- UDP is connectionless
- TCP uses a three-way handshake (SYN - SYN/ACK - ACK) to initiate a connection
- TCP uses a four segment process to close a connection (FIN - ACK - FIN/ACK - ACK)
- A TCP window is the amount of data that will be sent before waiting for an ACKnowledgement
- DHCP uses UDP to dynamically assign IP addresses to hosts
 - DHCP is a broadcast protocol
 - A DHCP helper can forward DHCP broadcasts to a DHCP server in a different broadcast domain
 - DHCP Assist can help distinguish subnets on a DHCP helper that is serving multiple subnets
- Network Address Translation (NAT) translates one IP address into another IP address
 - Port Address Translation (PAT) is used to translate many IP addresses to the same IP address
 - PAT may also translate inbound connections to a specific IP address depending on the destination port
- Packet capturing can help you troubleshoot your network
- sFlow is an open standard that is integrated into all Brocade switches
 - sFlow acts like a packet capture, by taking a sample of all packets coming through any port

Chapter Review Questions

1. Which Layer 4 protocol provides connectionless service?
 - a. IP
 - b. UDP
 - c. TCP
 - d. ARP
2. What is the well-known port for SSL?
 - a. TCP 443
 - b. TCP 80
 - c. TCP 25
 - d. UDP 443

3. What is the broadcast address of 172.16.25.33 255.255.255.224?
 - a. 172.16.25.255
 - b. 172.16.0.0
 - c. 172.16.25.63
 - d. 172.16.25.128
4. What is the last segment sent in a TCP three-way handshake?
 - a. SYN
 - b. FIN
 - c. SYN/ACK
 - d. ACK
5. What subnet mask for 192.168.100.0 will give me only 14 usable host addresses?
 - a. 255.255.255.0
 - b. 255.255.255.240
 - c. 255.255.224.0
 - d. 255.255.255.224
6. Which translation protocol will translate based on Layer 3 addresses and Layer 4 ports?
 - a. NAT
 - b. ARP
 - c. PAT
 - d. ICMP
7. Which subnet mask should I use if I need 50 subnets and 1,000 hosts for each subnet, if I'm using 172.16.0.0?
 - a. 255.255.0.0
 - b. 255.255.252.0
 - c. 255.255.255.0
 - d. 255.255.224.0
8. What Layer 3 protocol matches an IP address with a MAC address?
 - a. NAT
 - b. PAT
 - c. ARP
 - d. ICMP

9. I have an IP address of 192.168.100.100/27. What is my network address?
 - a. 192.168.100.0
 - b. 192.168.100.100
 - c. 192.168.100.27
 - d. 192.168.100.96
10. What class is 138.232.49.3 in?
 - a. Class A
 - b. Class B
 - c. Class C
 - d. Class D

Answers to Review Questions

1. b. UDP is connectionless. TCP is connection-oriented. IP and ARP are not Layer 4 protocols.
2. a. SSL uses TCP 443. TCP 80 is HTTP. TCP 25 is SMTP. There is no well-known protocol for UDP 443.
3. c. The subnet mask translates to 27 bits. 172.16.25.33 translates to 1010 1100 . 0001 0000 . 0001 1001 . 0010 0001. The first 27 bits are in bold. This is the network address. That means that the broadcast address would be: 1010 1100 . 0001 0000 . 0001 1001 . 0011 1111 or 172.16.25.63.
4. d. The sequence is SYN - SYN/ACK - ACK. The last segment is an ACK.
5. b. How many host bits do you need? Well, two bits will give you two usable hosts ($2 \times 2 = 4$, then subtract 2 for network and broadcast addresses; you're left with 2). Three bits will give you six usable hosts ($2 \times 2 \times 2 = 8$; $8 - 2 = 6$). Four bits will give you 14 usable hosts ($2 \times 2 \times 2 \times 2 = 16$; $16 - 2 = 14$). If you need four bits for the hosts, that leaves 28 bits for the network ($32 - 4 = 28$). A 28-bit mask looks like 1111 1111 . 1111 1111 . 1111 1111 . 1111 0000 (or 255.255.255.240).
6. c. PAT translates based on both Layer 3 (IP) addresses and Layer 4 (TCP/UDP) ports. NAT translates based only on Layer 3 addresses. ARP is not really a translation protocol, but it does match IP addresses to MAC addresses. ICMP is not a translation protocol.
7. b. For questions like these, it's usually easier to work out the small number first, so let's look at the network requirement. We know from question 5 that four bits will give us 14 usable addresses. Five bits will give us 30 usable addresses ($2 \times 2 \times 2 \times 2 \times 2 = 32$; $32 - 2 = 30$). Six bits will give us 62 usable addresses ($2 \times 2 \times 2 \times 2 \times 2 \times 2 = 64$; $64 - 2 = 62$), so six bits is what we need for the networks. 172.16.0.0 is a Class B IP network, so we

can't use the first 16 bits (that's the default subnet mask). If we need six bits, we'll add them to the default mask for a total of 22 bits ($16 + 6 = 22$). A 22-bit subnet mask looks like 1111 1111 . 1111 1111 . 1111 1100 . 0000 0000 (or 255.255.252.0). As you can see from the mask, this leaves 10 bits for the hosts. Conveniently, $2^{10} = 1,024$, and $1,024 - 2 = 1,022$. Just what we needed (or close enough).

8. c. ARP matches an IP address with its MAC address. NAT and PAT translate IP addresses. ICMP sends messages between IP devices.
9. d. The address looks like this in binary (with the first 27 bits - the network portion - in bold): 1100 0000 . 1010 1000 . 0110 0100 . 0110 0100. The network portion translates to 192.168.100.96 (in decimal).
10. b. It is a Class B address because the first octet is between 128 and 191 (138.232.49.3). If you really want to get tricky, you can translate the first octet (138) to binary, and see for yourself. In binary, 138 is 1000 1010. Notice that the first two bits are "10." If the first bit is "0," it's a Class A. If the first two bits are "10," it's a Class B. If the first three bits are "110," it's a Class C. And so on.

Data Center Bridging (DCB)

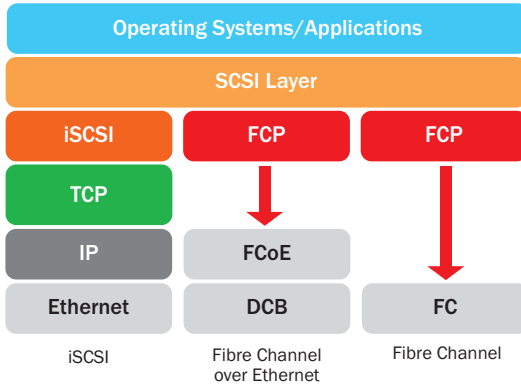
Introduction

Data networking and storage networking technologies have evolved on parallel but separate paths. Ethernet has emerged as the technology of choice for enterprise data networks, while Fibre Channel (FC) became the dominant choice for enterprise shared Storage Area Networks (SANs). Ethernet and Fibre Channel continue to evolve achieving higher speeds and adding new features. Anyone you ask is bound to have an opinion about Ethernet vs. Fibre Channel and they'll probably all be right to some extent!

Most large organizations invested in both technologies for data center needs: Ethernet for front-end Local Area Networks (LANs) linking users to enterprise servers and Fibre Channel for back-end SAN links between server and storage.

These days, server virtualization requires powerful CPUs and servers, placing higher demands on server networking and storage I/O interconnects. Maintaining separate data and storage networks also adds to the complexity of managing and maintaining data centers. As enterprises embrace virtualization and multiple applications are consolidated onto a smaller number of powerful servers, the next step involves simplifying server I/O links by converging the data and storage networks onto a common shared transport. New industry standards are needed to enable the convergence of data and storage networking interconnects and manage the behavior of diverse converged traffic flows sharing 10 GbE links.

The new transport of storage traffic over Ethernet had to be enhanced to handle storage traffic. New features have been added to Ethernet to improve its ability to transport storage traffic in a reliable and predictable lossless manner. A new flow control mechanism allows 10 GbE links to handle multiple traffic flows and deal with flow control and congestion problems of each flow separately. A new mechanism enables management of the various traffic flows to deliver a quality of service to applications that need higher priority. Efforts to enhance Ethernet making it lossless, more reliable, and predictable are carried out by IEEE under the framework of Data Center Bridging, or DCB. The following diagram shows the possible protocols that may be converged onto a shared lossless 10 GbE link.

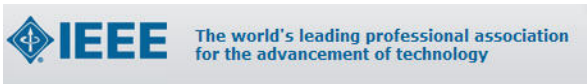


The changes, or enhancements, made to Ethernet as part of the DCB effort to support convergence do not prevent the simultaneous deployment of a TCP/IP stack. More importantly, since the new lossless Ethernet can be beneficial to the flow control aspects of TCP/IP, the lossless behavior of the new Ethernet can be turned on or off for LAN TCP/IP traffic. That means that data center professionals can deploy LANs using all of the following on one converged medium:

- Traditional Ethernet, lossless
- Ethernet with DCB features
- Fibre Channel over Ethernet (FCoE) with DCB features enabled (the FCoE protocol enables the transport of FC storage traffic over a new lossless Ethernet medium)

The combination of DCB and FCoE technologies provides a solution for the challenges of physical infrastructure reduction and cabling simplicity. Subsequent sections describe how this was accomplished. OK, this standards stuff is pretty dry but on some level, it's all about the standards.

IEEE—Data Center Bridging



The Data Center Bridging (DCB) effort undertaken by the IEEE 802.1 work group adds new extensions to bridging and Ethernet, so that it becomes capable of converging LAN and storage traffic on a single link. Often, you hear that new DCB features will make Ethernet "like Fibre Channel." That's true, because the new features being added to Ethernet are solving issues that FC faced in the past and successfully resolved. IEEE is expected to complete its work on the components of DCB by the end of 2010. The new enhancements are PFC, ETS, and DCBX.

802.1Qbb: Priority-based Flow Control (PFC)

Establishes eight priorities for flow control based on the priority code point field in the IEEE 802.1Q tags to control individual data flows on shared lossless links. PFC capability allows FC storage traffic encapsulated in FCoE frames to receive lossless service from a link that is shared with traditional LAN traffic, which is loss-tolerant.

802.1Qaz: Enhanced Transmission Selection (ETS)

ETS provides the capability to group each type of data flow, such as storage or networking, and assigns a group identification number to each of the groups, which are also called traffic class groups. And why should we care about that? Well, it means that you can manage bandwidth on the Ethernet link by allocating portions (percentages) of the available bandwidth to each of the groups. Bandwidth allocation allows traffic from the different groups to receive their target service rate (e.g., 8 Gbps for storage and 2 Gbps for LAN). Bandwidth allocation provides quality of service to applications.

Incorporates Data Center Bridging Exchange (DCBX), a discovery and initialization protocol that discovers the resources connected to the DCB cloud and establishes cloud limits. DCBX distributes the local configuration and detects the misconfiguration of ETS and PFC between peers. It also provides the capability for configuring a remote peer with PFC, ETS, and application parameters. The application parameter is used for informing the end node which priority to use for a given application type (e.g., FCoE, iSCSI). DCBX leverages the capabilities of IEEE 802.1AB Link Layer Discovery Protocol (LLDP).

802.1Qau: Congestion Notification (QCN)

An end-to-end congestion management that enables throttling of traffic at the edge nodes of the network in the event of traffic congestion.

IETF - TRILL



The Internet Engineering Task Force (IETF)

The goal of the IETF is to make the Internet work better.

Internet Engineering Task Force (IETF) is developing a new shortest path frame routing for multi-hop environments. The new protocol is called Transparent Interconnection of Lots of Links, or TRILL for short, and is expected to be completed in the second half of 2010.

- TRILL provides a Layer 2 (L2) multi-path alternative to the single path and network bandwidth limiting Spanning Tree Protocol (STP), currently deployed in data center networks.
- TRILL will also deliver L2 multi-hop routing capabilities, which are essential for expanding the deployment of DCB/FCoE solutions beyond access layer server I/O consolidation and into larger data center networks.

For more about TRILL, see “[Chapter 4](#)” starting on page 89.

Making Ethernet Lossless

In its original form, Ethernet was designed as best-effort delivery architecture. It does its best to be sure that a frame is correctly transmitted from one node to another until it has reached its final destination. As a result, Ethernet is a lossy transport that is not suitable for transporting storage data, which requires a lossless medium to ensure data delivery and protect against loss of valuable customer data.

As originally designed, many Ethernet nodes would listen to the same segment of media and would copy all the frames they heard, but keep only those frames intended for that node. When a node wanted to transmit, it would begin by listening to the media to be sure that no one else was transmitting. If so, it would wait until the media was silent and then begin to modulate the frame out onto the shared media. At the same time, the node would listen to the media to be sure that it heard the same data that it was sending. If it heard the same frame from beginning to end, then the node considered the frame sent and would do no further work at the Ethernet layer to be sure that the message arrived correctly.

If the node happened to hear something other than what it was transmitting, it would assume that another node began transmitting at about the same time. This node would then continue to transmit for a period to be sure that the other node was also aware of the collision but would then abort the frame. After a random time interval, the node would try to send the frame again. By using this approach, a node can be confident that a frame was sent correctly but not whether the frame was received correctly.

Ethernet implementations have moved from this shared-media approach to one in which each segment of media is shared by only two Ethernet nodes. Dual unidirectional data paths allow the two nodes to communicate with each other simultaneously without fear of collisions. Although this approach addresses how frames are delivered between Ethernet nodes, it doesn't change the behavior of how frames are treated once they're received.

The rules of Ethernet allow a node to throw away frames for a variety of reasons. For example, if a frame arrives with errors, it's discarded. If a non-forwarding node receives a frame not intended for it, it discards the frame. But most significantly, if a node receives an Ethernet frame and it has no data buffer in which to put it, according to the rules of Ethernet, it can discard the frame. It can do this because it's understood that nodes implementing the Ethernet layer all have this behavior. If a higher-level protocol requires a lossless transmission, another protocol must be layered on top of Ethernet to provide it.

Consider an implementation of the FTP protocol running across an Ethernet network. FTP is part of the TCP/IP tool suite. This means that from the bottom layer up, FTP is based on Ethernet, IP, TCP, and finally FTP itself. Ethernet does not guarantee that frames will not be lost and neither does IP. The TCP layer is

responsible for monitoring data transmitted between the FTP client and server, and if any data is lost, corrupted, duplicated, or arrives out of order, TCP will detect and correct it. It will request the retransmission of data if necessary, using the IP and Ethernet layers below it to move the data from node to node. It will continue to monitor, send, and request transmissions until all the necessary data has been received reliably by the FTP application.

The Ethernet PAUSE Function

A decision was made by the IEEE committee members that lossless behavior for Ethernet would be implemented by using a variant of the PAUSE function currently defined as part of the Ethernet standard. The PAUSE function allows an Ethernet node to send a PAUSE frame to an adjacent node. The PAUSE semantics require the receiving node not to send any additional traffic until a certain amount of time has passed. This time is specified by a field in the PAUSE frame.

Using this approach, lossless behavior can be provided if a receiving node issues PAUSE requests when it doesn't have any buffer space available to receive frames. It assumes that by the time the PAUSE request expires, there will be sufficient buffer space available. If not, it is the responsibility of the receiving node to issue ongoing PAUSE requests until sufficient buffer space becomes available.

The PAUSE command provides a mechanism for lossless behavior between Ethernet nodes, but it's only suited for links carrying one type of data flow. Remember that one of the goals of FCoE is to allow for I/O consolidation, with TCP/IP and Fibre Channel traffic converged onto the same media. If the PAUSE command is used to guarantee that Fiber Channel frames are not dropped as is required by that protocol, then as a side effect, TCP/IP frames will also be stopped once a PAUSE command is issued. The PAUSE command doesn't differentiate traffic based on protocols. It pauses all traffic on the link between two nodes, even control commands.

So a conflict between what must be done to accommodate storage traffic in FCoE and TCP/IP traffic—both of which need to coexist on the same segment of media. And problems could arise because one type of network traffic may interfere with the other.

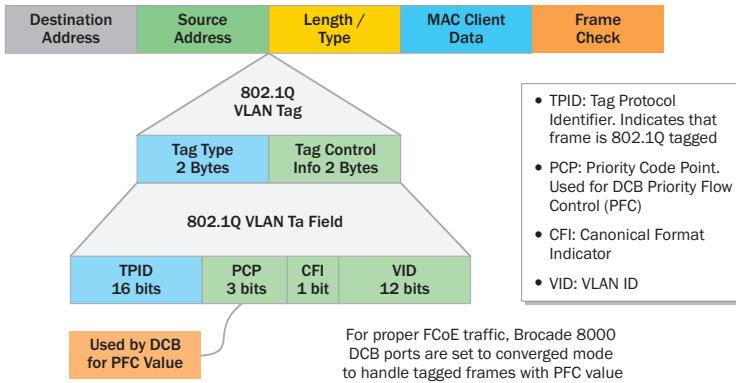
Suppose, for example, that storage traffic is delayed because of a slow storage device. In order to not lose any frames relating to the storage traffic, a PAUSE command is issued for a converged link carrying both FCoE and TCP/IP traffic. Even though the TCP/IP streams may not need to be delayed, they will be delayed as a side effect of having all traffic on the link stopped. This in turn could cause TCP time-outs and may even make the situation worse as retransmit requests for TCP streams add additional traffic to the already congested I/O link.

The solution to this problem is to enable Ethernet to differentiate between different types of traffic and to allow different types of traffic to be paused individually if required.

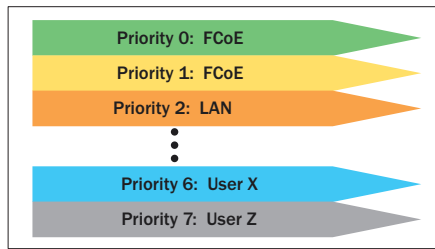
Priority-based Flow Control

The idea of Priority-based Flow Control (PFC) is to divide Ethernet traffic into different streams or priorities. That way, an Ethernet device can distinguish between the different types of traffic flowing across a link and exhibit different behaviors for different protocols. For example, a PAUSE command could be implemented to stop the flow of FCoE traffic when necessary, while allowing TCP/IP traffic to continue flowing.

When such a change is made to the behavior of Ethernet, there is a strong desire to do it with minimum impact to Ethernet networks already deployed. An examination of the IEEE 802.1Q VLAN header standard reveals that a three-bit field referred to as the Priority Code Point (PCP) could be used to differentiate between eight different traffic priority levels and therefore distinguish eight different types of traffic on the network.



In addition, the Ethernet PAUSE command has a sufficient number of bytes available to allow an individual pause interval to be specified for each of the eight levels, or classes, of traffic. FCoE and TCP/IP traffic types can therefore be converged on the same link but placed into separate traffic classes. The FCoE traffic can be paused to guarantee the reliable delivery of frames, while TCP/IP frames continue to flow. Not only can different traffic types coexist, but best practices for each can be implemented in a non-intrusive manner.



From another perspective, consider that PFC attempts to emulate Virtual Channel (VC) technology widely deployed in current Brocade Fibre Channel SANs. While borrowing the lossless aspect of VCs, PFC retains the option of being configured as lossy or lossless. PFC is an enhancement to the current link level of Ethernet flow control mechanism defined in IEEE 802.3x (PAUSE). Current Ethernet protocols support the capability to assign different priorities to different applications, but the existing standard PAUSE mechanism ignores the priority information in the Ethernet frame.

Triggering the PAUSE command results in the link shutting down, which impacts all applications even when only a single application is causing congestion. The current PAUSE is not suitable for links in which storage FCoE and networking applications share the same link, because congestion caused by any one of applications shouldn't disrupt the rest of the application traffic.

IEEE 802.1Qb is tasked with enhancing the existing PAUSE protocol to include priority in the frames contributing to congestion. PFC establishes eight priorities using the priority code point field in the IEEE 802.1Q tags, which enable the control of individual data flows, called flow control, based on the frame's priority. Using the priority information, the peer (server or switch) stops sending traffic for that specific application, or priority flow, while other applications data flows continue without disruption on the shared link.

The new PFC feature allows FC storage traffic encapsulated in FCoE frames to receive lossless service from a link shared with traditional LAN traffic, which is loss tolerant. In other words, separate data flows can share a common lossless Ethernet, while each is protected from flow control problems of the other flows—and everyone is happy. Note that LAN traffic priorities can be configured with PFC off, allowing for lossy or lossless LAN transmissions.

Enhanced Transmission Selection

With the use of Priority Flow Control, it is possible to combine eight different levels or classes of traffic onto the same converged link. Each of these classes can be paused individually if necessary without interfering with other classes. *PFC does not, however, specify how the bandwidth is to be allocated to separate classes of traffic.*

Suppose, for example, that a particular application happens to hit a hot spot that causes it to send a large number of TCP/IP messages. There is a good chance that the transmission of all these messages could interfere with the operating system's attempt to either retrieve or store block information from the storage network. Under these or similar circumstances, bandwidth starvation could cause either the application or the operating system to crash.

This situation doesn't occur if separate channels are used for storage and non-storage traffic. A Fibre Channel-attached server could access its block traffic independent of the messages traveling across an Ethernet TCP/IP connection. Competition for bandwidth occurs only when these two ordinarily independent streams share a common link.

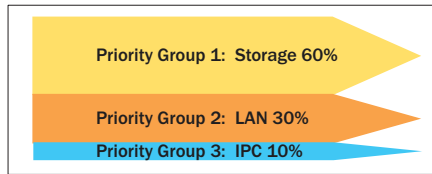
To ensure that all types of traffic are given the appropriate amount of bandwidth, a mechanism called Enhanced Transmission Selection (ETS) is used with Priority Flow Control. ETS establishes priorities and bandwidth limitations so that all types of traffic receive the priority and bandwidth they require for the proper operation of the server and all applications.

ETS establishes Priority Groups (PGs), or traffic class groups. A Priority Group is a collection of priorities as established in PFC. For example, all of the priorities associated with Inter-Process Communication (IPC) can be allocated to one Priority Group. All priorities assigned to FCoE can be assigned to a second group, and all IP traffic can be assigned to a third group.

Each Priority Group has an integer identifier called the Priority Group ID (PGID). The value of the PGID is either 15 or a number in the range of 0 through 7. If the PGID for a Priority Group is 15, all traffic in that group is handled on a strict priority basis. That is, if traffic becomes available, it's handled before traffic in all other Priority Groups without regard for the amount of bandwidth it takes. But be cautious. A PGID of 15 should be used only with protocols requiring either an extremely high priority or very low latency. Examples of traffic in this category include management traffic, IPC, or audio/video bridging (AVB).

The other traffic class groups with PGID identifiers between 0 and 7 are assigned a bandwidth allocation (PG%). The sum of all bandwidth allocations should equal 100%. The bandwidth allocation assigned to a traffic class group is the guaranteed minimum bandwidth for that group assuming high utilization of the link. For example, if the PG for the traffic class group containing all storage traffic is 60%, it is guaranteed that at least 60% of the bandwidth available after all PGID 15 traffic has been processed will be allocated to the storage traffic Priority Group.

The specification for ETS allows a traffic class group to take advantage of unused bandwidth available on the link. For example, if the storage traffic class group has been allocated 60% of the bandwidth and the IP traffic class group has been allocated 30%, the storage group can use more than 60% if the IP traffic class group does not require the entire 30%.



Data Center Bridge eXchange

In order for PFC and ETS to work as intended, both devices on a link must use the same rules. (PFC and ETS are implemented only in point-to-point, full-duplex topologies.) The role of Data Center Bridge eXchange (DCBX) is to allow two adjacent nodes to exchange information about themselves and what they support. If both nodes support PFC and ETS, then a lossless link can be established to support the requirements of FCoE.

As with other protocols, an implementation goal of DCBX is that it must be possible to add DCBX-equipped devices to legacy networks without breaking them. That's the only way to build a lossless Ethernet sub-cloud inside a larger Ethernet data center deployment. This is discussed in more detail in a following section.

Today, nearly all Ethernet devices are equipped to support the Link Layer Discovery Protocol (LLDP), a mechanism whereby each switch periodically broadcasts information about itself to all of its neighbors. It's a one-way protocol, meaning that there is no acknowledgement of any of the data transmitted. Broadcasted information includes a chassis identifier, a port identifier, a time-to-live (TTL) field, and other information about the state and configuration of the device.

Information in an LLDP data unit (LLDPDU) is encoded using a type-length-value (TLV) convention:

- Each unit of information in the LLDPDU starts with a type field, which tells the receiver what that information block contains.
- The next field, the length field, allows a receiver to determine where the next unit of information begins. By using this field, a receiver can skip over any TLVs that it either doesn't understand or doesn't want to process.
- The third element of the TLV is the value of that information unit.

The LLDP standard defines a number of required and optional TLVs. It also allows for a unique TLV type, which permits organizations to define their own TLVs as required. By taking advantage of this feature, DCBX can build on LLDP to allow two nodes to exchange information about their ability to support PFC and ETS. Nodes that do not support PFC and ETS are not negatively impacted by the inclusion of this information in the LLDPDU, and they can just skip over it. The absence of DCBX-specific information from an LLDPDU informs an adjacent node that it's not capable of supporting those protocols.

DCBX also enhances the capabilities of LLDP by including additional information that allow the two nodes to be better informed about what the other node has learned to keep the two nodes in sync. For example, the addition of sequence numbers in the DCBX TLV allows each of the two nodes to know that it has received the latest information from its peer and that its peer has received the latest information from it.

There are currently three different subclasses of information exchanged by DCBX:

- The first subclass is control traffic for the DCBX protocol itself. By using this subtype, state information and updates can be exchanged reliably between two peers.
- The second subtype allows the bandwidth for Traffic Class Groups to be exchanged. The first part of this data unit identifies the PGID for each of the seven message priorities. The second part of the data unit identifies the bandwidth allocation that is assigned to each of the PGIDs 0 through 7. Recall that PGID 15 is a special group that always gets priority over the others independent of any bandwidth allocation.
- The final part of the subtype allows a node to identify how many traffic classes it supports on this port. You can think of a traffic class as a collection of different types of traffic that are handled collectively. The limitation on the number of traffic classes supported by a port may depend on physical characteristics such as the number message queues available or the capabilities of the communication processors.

Because of the grouping of message priorities into traffic classes, it's not necessary for a communications port to support as many traffic classes as there are priorities. To support PFS and ETS, a communications port only needs to be able to handle three different traffic classes:

- One for PGID 15 high priority traffic
- One for those classes of traffic that require PFC support for protocols such as FCoE
- One for traffic that does not require the lossless behavior of PFC such as TCP/IP.

By exchanging the number of traffic classes supported, a node can figure out if the allocation of additional Priority Groups is possible on the peer node.

The third subtype exchanged in DCBX indicates two characteristics of the sender. First, it identifies which of the message priorities should have PFC turned on. For consistency, all of the priorities in a particular Priority Group should either require PFC or not. If those requiring PFC are mixed up with those that do not, buffer space will be wasted and traffic may be delayed. The second piece of information in this subtype indicates how many traffic classes in the sender can support PFC traffic. Because the demands for PFC-enabled traffic classes are greater than those classes of traffic that do not require lossless behavior, the number of traffic classes supporting PFC may be less than the total number of traffic classes supported on a port. By combining the information in this subtype with that in the previous subtype, a node can determine the number of PFC-enabled and non-PFC-enabled traffic classes supported by a peer.

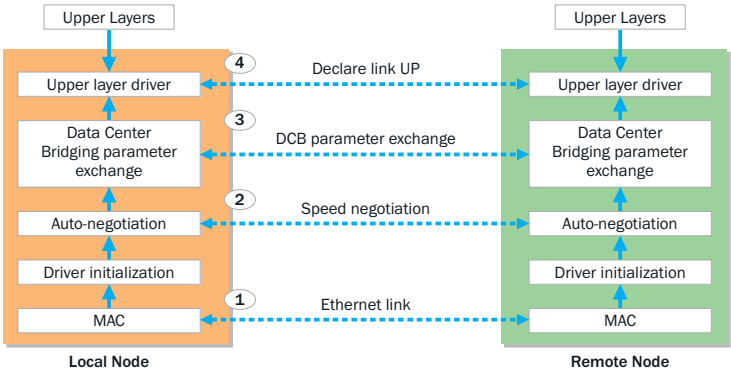
Building the DCB Cloud

The FC-BB-5 specification requires that FCoE run over lossless Ethernet. One of the first tasks of an Ethernet environment intended to run FCoE must be to determine which nodes can participate in FCoE and which links can be used to connect to those nodes. PFC and ETS must be supported on all links carrying this traffic, and the boundaries between lossy and lossless must be established.

To do this, switches and devices capable of supporting FCoE will broadcast their LLDP messages as they are initialized. These messages will include the DCBX extensions to identify themselves as being PFS and ETS capable. If a node receives an LLDP message from a peer and that message does not contain DCBX information, then the link will not be used for lossless traffic.

When a node receives a DCBX-extended LLDP message, it will examine the values of the parameters for compatibility. The peer must be capable of supporting the required protocols and must have like configurations for PGs and PGIDs. At first, this may sound like a daunting problem, but most experts agree that just as there are best practices for other protocols, there will be best practices for determining PGs, PGIDs, and PG%s. There will be mechanisms for customizing these values for special situations, but the default configuration values will be those generally agreed upon by the industry.

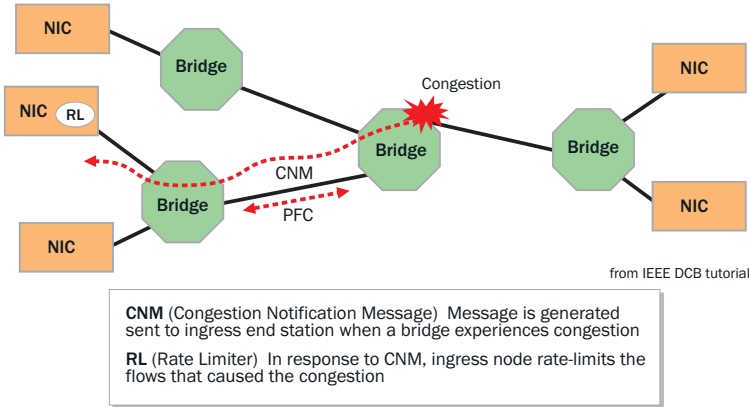
As devices participating in the LLDP process establish which links will be used for lossless Ethernet traffic, a natural boundary will form. Within this boundary, FCoE traffic will be allowed to move between nodes and switches. TCP/IP traffic will be allowed to travel within, across, and beyond this boundary. But to minimize the impact of TCP/IP on storage paths, a best practice will be to direct all IP traffic out of the cloud as quickly as possible toward nodes not within the lossless boundary.



Congestion Notification

802.1Qau: Congestion Notification (QCN)

An end-to-end congestion management mechanism enables throttling of traffic at the end nodes in the network in the event of traffic congestion. When a device is congested, it sends a congestion notification message to the end node to reduce its transmission. End nodes discover when congestion eases so that they may resume transmissions at higher rates.



But don't forget that QCN is a separate protocol independent of ETS and PFC. While ETS and PFC are dependent on each other, they do not depend on or require QCN to function or be implemented in systems.

Summary

- Most data centers in large organizations use Ethernet for front-end LANs linking users to enterprise servers and Fibre Channel for back-end SAN links between server and storage.
- The DCB effort undertaken by the IEEE 802.1 work group adds new extensions to bridging and Ethernet, so that it becomes capable of converging LAN and storage traffic on a single link.
- The combination of DCB and FCoE technologies provides a solution for the challenges of physical infrastructure reduction and cabling simplicity
- The following standards organizations are working on enhancements to the Ethernet standards to enable DCB and FCoE:
 - IEEE for DCB (PFC, ETS, and DCBX)
- The idea of Priority-based Flow Control (PFC) is to divide Ethernet traffic into different streams or priorities.

- Enhanced Transmission Selection (ETS) is used with Priority Flow Control. ETS establishes priorities and bandwidth limitations so that all types of traffic receive the priority and bandwidth they require for the proper operation of the server and all applications.
- The role of Data Center Bridge eXchange (DCBX) is to allow two adjacent nodes to exchange information about themselves and what they support.
 - An implementation goal of DCBX is that it must be possible to add DCBX-equipped devices to legacy networks without breaking them.

NOTE: There are no review questions and answers for this chapter.

TRILL—Adding Multi-Pathing to Layer 2 Networks

The ever-increasing adoption of virtual environments in the data center necessitates a more resilient Layer 2 network. Efficient and reliable L2 infrastructure is needed to support the I/O demands of virtual applications, especially when applications are migrated across servers or even different data centers. Today's STP-based networks limit the available network bandwidth and fail to maintain reliable complex networks architectures. IETF is developing a new shortest path frame L2 routing protocol for multi-hop environments.

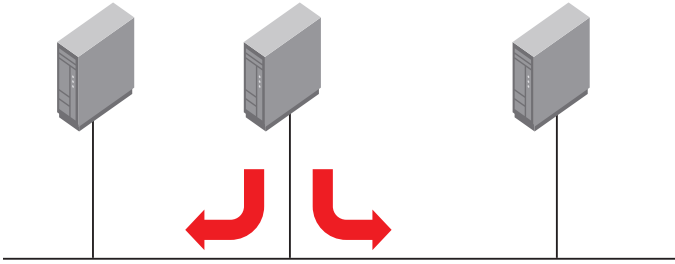
The new protocol is called Transparent Interconnection of Lots of Links, or TRILL, and it is expected to be completed in the second half of 2010. TRILL will enable multi-pathing for L2 networks and remove the restrictions placed on data center environments by STP (single path) networks. Data centers with converged networks will also benefit from the multi-hop capabilities of TRILL Routing Bridges (RBridges) as they also enable multi-hop FCoE solutions.

Why do we need it?

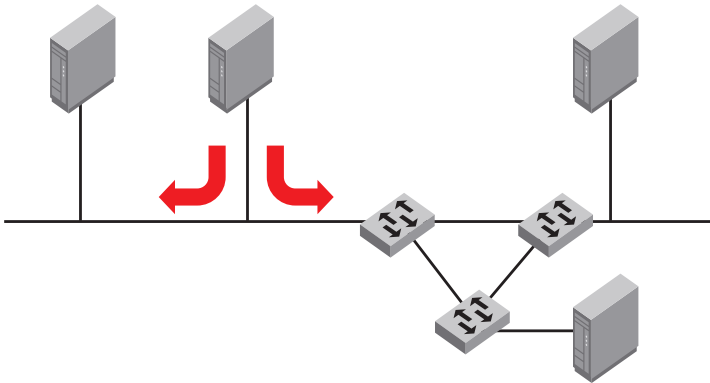
Resiliency in a network is always good. In the event of a failure in any one component, it's important to have sufficient resources to allow the network to continue functioning around the problem areas. This is especially important in converged networks with storage and IP resources, where the interruption of service may result in the disastrous failure of one or more server platforms.

And to reap the greatest advantage from a network, an administrator would like to be able to use all of the available capacity of the network for moving traffic. If there are multiple paths of lowest equal cost through a network from Point A to Point B, in a best-case scenario all of those paths would be used. Unfortunately, the evolution of Ethernet has introduced restrictions that do not always allow for either the maximum resiliency or the highest capacity in a network.

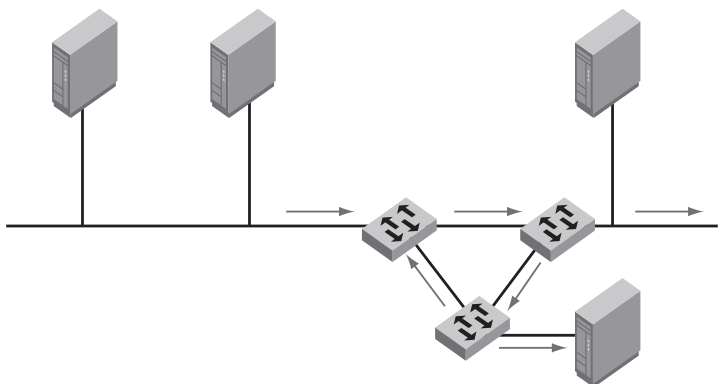
Because Ethernet was originally a flat point-to-point topology across a single segment of shared media, you didn't need to be concerned about multiple paths through the network. Each node was logically connected directly to each of its peers with no intermediary devices along the way. That meant that the Ethernet protocol could ignore cases in which multiple paths from a source to a destination were available. As a result counters and headers for management metrics such as hop count and time-out values were unnecessary and were not included in the standard Ethernet frame.



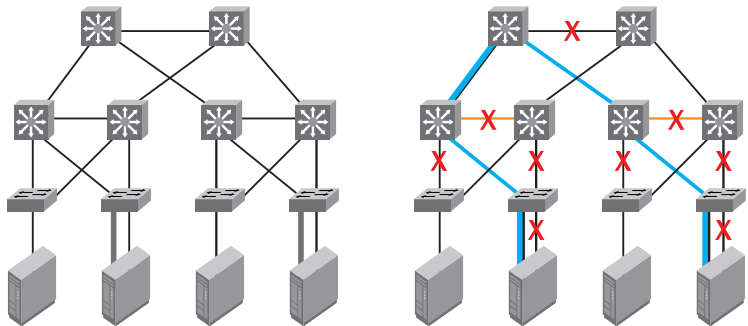
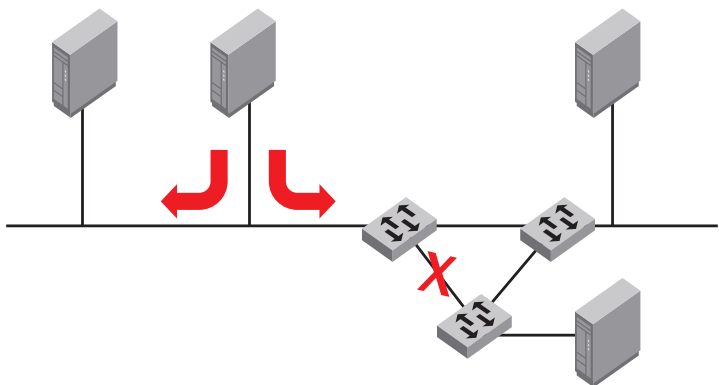
As Ethernet deployments became more common, new devices were introduced into the infrastructure to enable larger networks. Analog repeaters and digital bridges began to appear, and as they did, new complexities began to surface. For example, with these devices, you could design a network where there was more than one physical path from any one source to a destination.



The problem was that a network device receiving a frame on a port didn't know if it had seen that frame before. This introduced the possibility of a single frame circulating throughout the network indefinitely. Left unchecked, a network would soon be saturated with frames that couldn't be removed because they couldn't be identified or limited.



To address this problem, a logical topological restriction in the form of a spanning tree was placed on Ethernet networks. The STP protocol meant that: *although there may be many physical paths through the network at any given time, all traffic will flow along paths that have been defined by a spanning tree that includes all network devices and nodes.* By restricting traffic to this tree, loops in the logical topology are prevented at the expense of blocking alternative network paths.



While STP solves the problem of traffic loops, it prevents network capacity from being fully used. Algorithms that calculate this spanning tree may take a lot of time to converge. During that time, the regular flow of traffic must be halted to prevent the type of network saturation described above. Even if multiple simultaneous spanning trees are used for separate VLANs to better distribute the traffic, traffic in any one VLAN will still suffer from the same disadvantage of not being able to use all of the available capacity in the network.

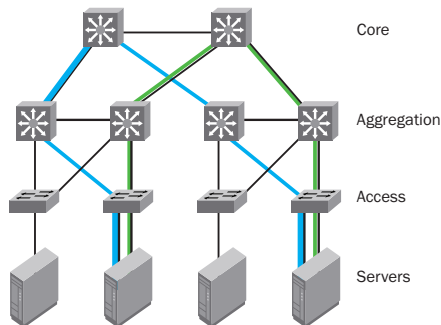
Introducing TRILL

To eliminate the restriction of a single path through the network, the IETF formed a working group to study this problem. The official documentation states the goal of the group this way: "The TRILL WG will design a solution for shortest-path frame routing in multi-hop IEEE 802.1-compliant Ethernet networks with arbitrary topologies, using an existing link-state routing protocol technology."

In simpler terms, the group was charged with developing a solution that:

- Uses shortest path routing
- Works at Layer 2
- Supports multi-hopping environments
- Works with an arbitrary topology
- Uses an existing link-state routing protocol
- Remains compatible with IEEE 802.1 Ethernet networks that use STP

The result was a protocol called TRILL. Although routing is ordinarily done at Layer 3 of the ISO protocol stack, by making Layer 2 a routing layer, protocols other than IP, such as FCoE, can take advantage of this increased functionality. Multi-hopping allows specifying multiple paths through the network. By working in an arbitrary topology, links that otherwise would have been blocked are usable for traffic. Finally, if the network can use an existing link-state protocol, solution providers can use protocols that have already been developed, hardened, and optimized. This reduces the amount of work that must be done to deploy TRILL.



Just as important is what TRILL doesn't do. Although TRILL can serve as an alternative to STP, it doesn't require that STP be removed from an Ethernet infrastructure. Most networking administrators can't just "rip and replace" their current deployments simply for the sake of implementing TRILL. So hybrid solutions that use both STP and TRILL are not only possible but will most likely be the norm at least in the near term. TRILL will also not automatically eliminate the risk of a single point of failure, especially in a hybrid architecture. The goals of TRILL are restricted to those explicitly listed above. Some unrealistic expectations and misrepresentations have been made about this technology, so it's important to keep in mind the relatively narrow range of problems that TRILL can solve.

Simply put, TRILL enables only two things:

- Multi-pathing for L2 networks
- Multi-hopping that can benefit FCoE

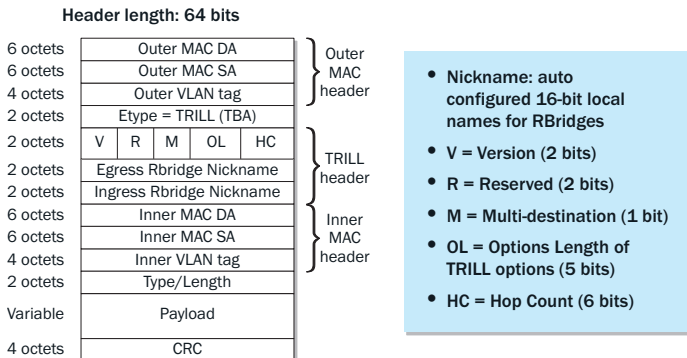
Anything else that you get with a TRILL deployment is a bonus!

The TRILL Protocol

Like other protocols, TRILL solutions will have three components; a data plane, a control plane, and devices that implement the protocol.

TRILL Encapsulation

TRILL encapsulation turns Ethernet frames into TRILL frames by adding a TRILL header to the frame. The new TRILL header is in exactly the same format as a legacy Ethernet header. This allows bridges (switches) that are not aware of TRILL to continue forwarding frames according to the rules they've always used. The source address used in the TRILL header is the address of the RBridge adding the header. The destination address is determined by consulting tables built by the link-state routing protocol. A new EtherType is assigned to TRILL. Note also the HC (hop count) field, a 6-bit field that allows for 64 hops. The HC field is used to prevent the formation of loops on the VLAN or the premature discarding of frames.



Link-State Protocols

As noted earlier, TRILL will use link-state protocols to form the control plane of TRILL. The purpose of the control plane is to distribute the VLAN configuration to all the R Bridges on the VLAN. Link-state protocols also continuously monitor the VLAN configuration and adjust the configuration data base in the event of changes. The control plane also provides the algorithms used to calculate the shortest path between any two R Bridges on the VLAN. Considering the fact that TRILL will be used in converged environments where storage and TCP/IP networks are deployed, you can expect that link-state protocols from both worlds will be utilized by TRILL.

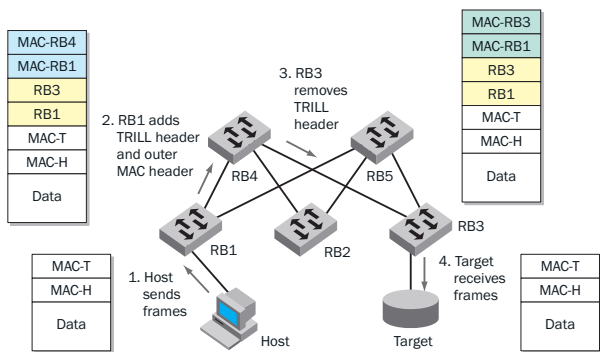
Routing Bridges

Routing bridges are a new type of L2 device that implement the TRILL protocol, perform L2 forwarding, and require little or no configurations. Using the configuration information distributed by the link-state protocol, R Bridges discover each other and calculate the shortest path to all other R Bridges on the VLAN. The combination of all calculated shortest paths make up the R Bridge routing table. It is important to note that all R Bridges maintain a copy of the configuration database, which helps reduce convergence time. When they discover each other, R Bridges select a designated bridge (DRB), which in turn assigns a designation for the VLAN and selects an appointed forwarder (AF) for the VLAN. Although the DRB can select itself as an AF, there can only be a single AF per VLAN. The AF handles native frames on the VLAN.

Moving TRILL Data

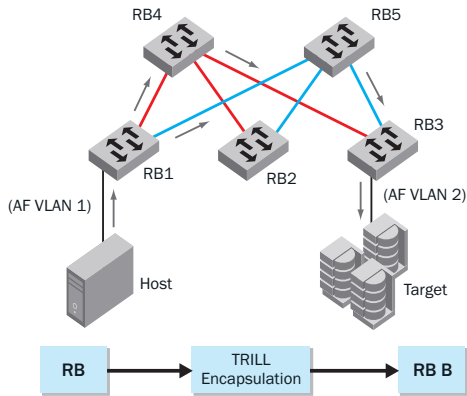
TRILL works by adding a new header to the beginning of an Ethernet frame. This new header is added to the frame when the first R Bridge encounters it. Note that this can happen if a host is directly connected to an R Bridge or if a non-routing Ethernet segment hands the frame off to an R Bridge—either way. If the frame can remain in a non-routing Ethernet segment without ever touching an R Bridge, then no header is added and it isn't really necessary.

Using the original destination address as a key, a list of eligible next-hop R Bridges is determined. This list contains the R Bridges that could be the next step along all least-cost paths moving to the final destination. If more than one R Bridge is in the list, a hash is used to distribute the traffic load and guarantee that all traffic in a single stream stays on the same path to avoid reordering overhead. The R Bridge that results from this is placed in the TRILL header and the frame is sent on.



The new TRILL header is in exactly the same format as a legacy Ethernet header. This allows bridges (switches) that are not aware of TRILL to continue forwarding frames according to the rules they've always used. The source address used in the TRILL header is the address of the RBridge adding the header. The destination address is determined by consulting the tables built by the link-state routing protocol.

When a frame with a TRILL header is received by an RBridge, the RBridge removes the header and examines the original source and destination addresses. It then creates a new TRILL header using the method described above and forwards the frame. The last RBridge receiving a frame prior to the delivery of the frame to either the destination or the local segment that connects to the destination removes the TRILL header and forwards the frame.



Summary

- Transparent Interconnection of Lots of Links (TRILL) is a new draft standard being created by IETF.
- The goal of TRILL is to create a Layer 2 shortest path routing protocol to replace STP and enable Layer 2 multi-pathing capability.
- The more resilient L2 will fulfill the needs of virtualized applications and data migration.
- It will also enable multi-hop capabilities for FCoE that will drive the expanded adoption of the new technology in converged network environments.

NOTE: There are no review questions and answers for this chapter.

Load Balancing Basics

With the rise of the Internet, load balancing is more popular than ever. It allows you to share the load of your traffic across many different servers. It also provides redundancy to your infrastructure. In this chapter, we'll go over many of the basic aspects of load balancing. In Section IV, we'll go over the specifics of how to configure them.

The Virtual IP (VIP) Address

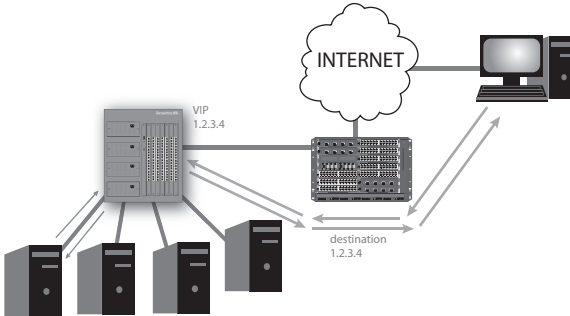
When you configure a device to be on a TCP/IP network, you will configure an IP address for the device. Whenever that device communicates on the network, it uses its IP address as the source IP address in the IP header of the packet. Whenever other devices on the network want to communicate to this device, they will use the device's IP address as the destination IP address in the IP header of the packet.

What if we have five servers that all provide the same function? We want all five to be used by our clients, but we don't want our clients to have to remember five different IP addresses. We also don't want them to have to use trial and error to make sure that the one server they chose is actually active. Here is where we would deploy load balancing, and the first element needed is a virtual IP address (VIP).

A VIP is an IP address that is not assigned to a specific NIC. In fact, it's not assigned to a specific host. It is an address that the clients would communicate with. Typically, the actual device that answers for the VIP is the load balancer. But traffic coming to the VIP will actually be handed off to one of the servers being load balanced. The server will actually do the processing of the data.

In our previous example, where we wanted our clients to use all five of our servers, the clients would now communicate with a VIP. The VIP would be hosted by the load balancer. When a request is sent by the clients, the load balancer will forward the request to one of the five servers. It will also be the load balancer's job to keep track of which servers may or may not be responding. This is so the client won't have to. That way, if a server goes down, the client may never know.

Most load balancers (including Brocade ServerIrons) can host many VIPs, regardless of how many or how few physical servers it is load balancing.

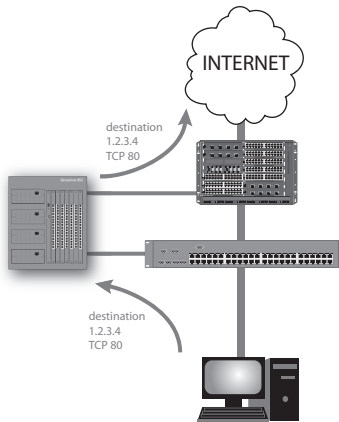


Load Balancing Methods

There are several different ways to load balance, and there are advantages and disadvantages to each method.

Proxy

A load balancer using a proxy method is usually a server, not a switch. The idea is that the client will actually terminate its session with the proxy server, and the server will initiate a new session to the server it chooses to load balance to.

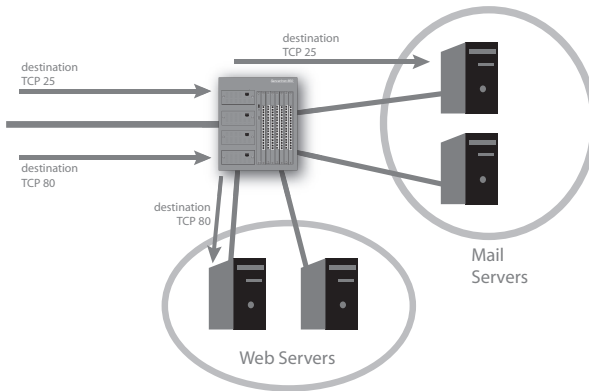


The proxy does have its advantages. For one, it allows for very granular control over the packets that pass through it. This control comes at a price: speed. Typically, a proxy load balancer is not going to perform as efficiently as others. In this book, we really won't be doing too much with the proxy load balancing method, but you should be aware of how it functions.

Layer 4 Switching

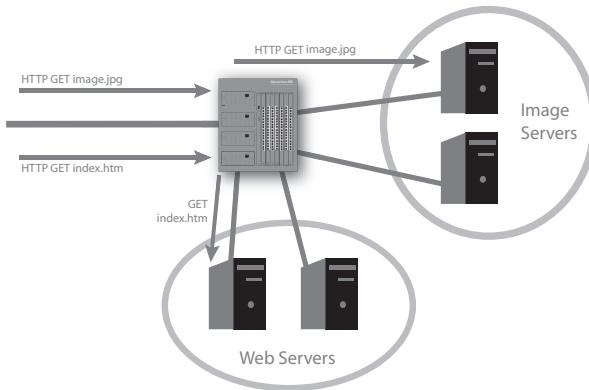
With pure Layer 4 switching, a load balancer looks at the segment header (TCP/UDP). This tells the load balancer which server or groups of servers to forward the packet to. For example, you could configure a load balancer to hand off to a group of web servers if the incoming destination port to the VIP is TCP 80 (HTTP). You could have the same load balancer forward to a different group of servers (mail servers) if the incoming destination port is TCP 25 (SMTP).

The point of Layer 4 switching is that the decision is based on the destination port in the Layer 4 header. This will tell the load balancer which actual servers it should be forwarding the packet to.



Layer 7 Switching

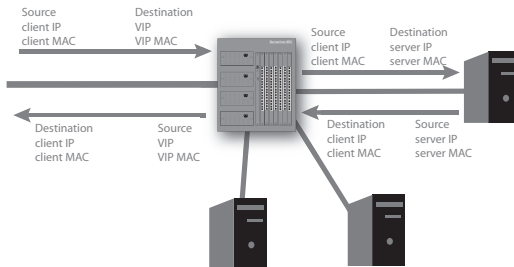
Layer 7 switching is a little more sophisticated. Here, the decision is made by looking at information in the data portion of the received frame. This is most commonly used with web servers, though it could be used with any Layer 7 protocol. For example, a client makes a request for the file “index.html.” The load balancer receives that request, and chooses a specific server. When it receives the next request asking for the file “sales.html,” the load balancer may be configured to hand that off to another specific server (or pool of servers).



Again, the idea is that the load balancer chooses which server to forward the packet to based on its Layer 7 data. This process provides for ultimate flexibility.

Server Load Balancing (SLB)

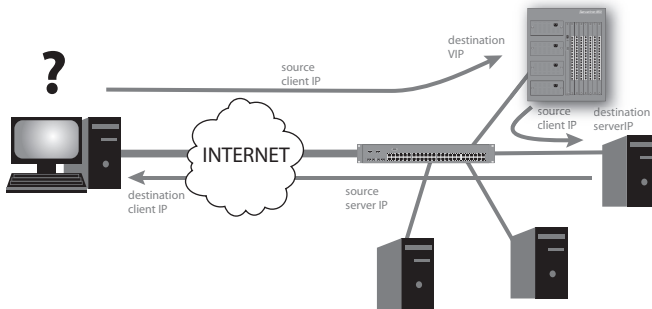
This is the most common method for load balancing. For SLB to function, all of the servers being load balanced must be physically connected to the load balancer, or the network must be engineered in such a way that all traffic going to or coming from the real servers passes through the load balancer. It employs Layer 4 switching or Layer 7 switching to make its decision. Once the decision is made, the load balancer changes the destination IP address and destination MAC address of the packet and forwards it to the server it has chosen. All return traffic must travel through the load balancer. This gives the load balancer the opportunity to intercept the reply, substitute the VIP's IP address and MAC address for the source address fields in the packet, and forward the packet back to the client.



As all traffic passes through the load balancer, this also allows the load balancer to make very intelligent decisions based on the number of sessions being processed on each server, the amount of data being passed, etc. SLB acts as a middleman, but it's a middleman that knows its servers intimately.

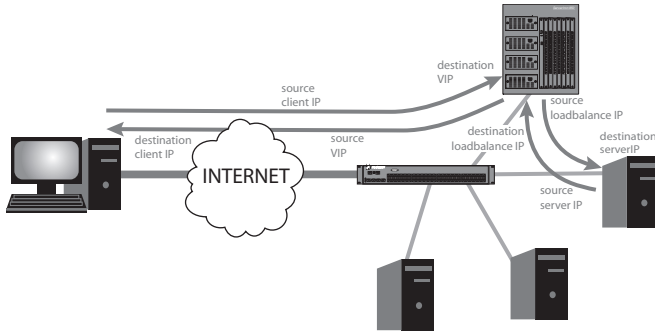
Source NAT

SLB is all well and good, but what if you have so many servers that you don't have enough ports to house them in your load balancer? What if the servers need to be physically attached elsewhere in the infrastructure? What if it's not practical to route all traffic to and from your real servers through the load balancer? If you still used SLB, the load balancer could still alter the incoming packet, and it would reach one of the servers. The problem would be in the reply. The reply would come from, not the VIP address as the client was expecting, but from the actual IP address of the server that received the request. This would confuse the client, and it would drop the packet.



Remember that for SLB to function, all traffic had to travel through the load balancer. While the incoming traffic would definitely be traveling through the load balancer, the return traffic would not. How do we make the return traffic travel back through the load balancer? We implement Source NAT.

Source NAT takes the incoming packet, and substitutes the destination addresses (IP and MAC) accordingly to the load-balanced server it has chosen. But then, the process also substitutes the *source* addresses (IP and MAC) with *its own address*. To the server, it will look like the packet originated from the load balancer. The server will process the request and send its reply back to the source, which it believes to be the load balancer. The load balancer will then take the reply, substitute the VIP address as the source, and the real client's IP address as the destination, and forward the packet back to the client.



Source NAT still allows the load balancer to keep an intimate record of data passing, active sessions, and so on, because all traffic is still coming into and going out of the load balancer.

Direct Server Return (DSR)

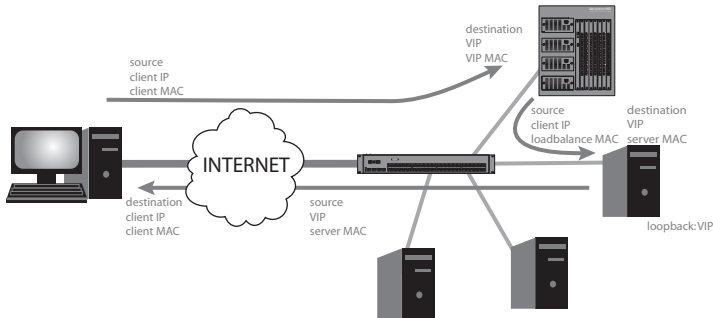
Okay, I've got a ton of servers (too many to physically connect to my load balancer), but what if I'm passing a really large amount of traffic? My load balancer, for example, only has a 100 Mbps connection to my core switch. That means that if all traffic is coming into and out of the load balancer (using Source NAT), I'm only going to get 100 Mbps worth of throughput. If I have 1,000 servers, I'm still ultimately limited by that 100 Mbps throughput. I need much more than that for my environment! Enter Direct Server Return (DSR).

Remember that if we use SLB (when your servers are not physically connected to the load balancer, and the network path does not route all traffic into and out of the load balancer) that it is the reply that is the problem. One way to solve this problem is with Source NAT. The disadvantage there is that all traffic is still going through the load balancer. If our load balancer's throughput to the core infrastructure is low (as in our example), this would be undesirable.

DSR presents another solution to the problem. The load balancer receives the incoming request, but this time, it only changes the destination MAC address, not the IP address, and forwards the packet to the chosen server.

But why would the server accept the packet? It's not addressed to it. Or is it? For DSR to work, a *loopback address* must be configured on all load-balanced servers. This address is not assigned to a specific NIC. Think of the loopback address as the address of the server itself. You may be familiar with the default loopback address of 127.0.0.1. Each server must be configured with an additional loopback address, which will be the same address as the VIP. Special precautions must be made so that the server configured with this loopback address will not advertise this address (in ARP messages), nor will it answer ARP broadcasts (otherwise, it would conflict with the load balancer). Consult Brocade's website (<http://www.brocade.com/>) for further details on configuring a DSR loopback IP address to your specific operating system.

With the loopback in place the server will believe that the packet belongs to it, process the request, and send the reply directly to the client. Because of the loopback address, it will use the VIP address as the source IP address of the reply. To the client, it would appear that it has had a communication with the VIP, as the response would appear to have come from the VIP.



With Direct Server Return, the request comes into the load balancer, gets forwarded to the server, and the server replies *directly* to the client (not through the load balancer). This allows for the load balancer's bandwidth to the core infrastructure to be only used for incoming traffic (which is typically the smaller amount of data), and the servers can use their individual bandwidth for the reply (the greater amount of traffic).

The Health Check

If the load balancer is going to share incoming traffic with several servers, it needs to be aware of which servers may or may not be active. If a server locks up, loses power, or has an application crash, the load balancer needs to know. If it doesn't, it could end up continuing to hand off traffic to a dead server. How does the load balancer know? It knows because of the health check.

A health check is a way to check on a server to see if it's functioning. It's kind of like the load balancer routinely saying, "Hey server, are you there? Are you okay? Do you feel alright? Could you handle any traffic if I were to send some your way?" It's a *check to verify the health* of the server. The load balancer must check the health of every server it is load balancing.

Some documentation refers to an "active health check." This is so named, as the load balancer is actively (not passively) checking on the health of its servers. It's configured to verify the servers' health above and beyond simply defining the servers' existence. An active health check can be categorized in one of two ways: *initialization* and *periodic*. An initialization health check is performed on a host when it is configured in the load balancer. Periodic health checks are those that are performed at repeating intervals to verify the status of the server.

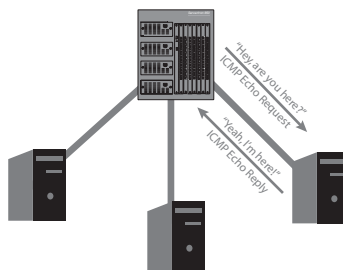
A load balancer will want to check on each server frequently, so it will know as soon as a server becomes unavailable. You don't want to check too often, as this could generate a lot of traffic and load on the servers. You don't want to check too little, because the longer you wait to check the health of the server, the longer it may take for the load balancer to realize a server has become inactive. A common interval would be about 5 seconds. For some applications, you may want to make the interval 30-60 seconds. This would minimize health check traffic and reduce health check induced load on the server. Remember that with a 60-second interval, a server could be dead for 59 seconds before your load balancer knows. That may be acceptable for some applications, but it may not be for others. The interval is adjustable to the needs of the VIP.

There are three general types of health checks that are typically performed. They are Layer 3, Layer 4, and Layer 7.

Layer 3

The Layer 3 (or Network Layer) health check verifies the first three layers of the server's configuration. It makes sure that the IP address of the server is reachable from the load balancer. To do this, the load balancer uses two Layer 3 protocols: ARP and ICMP. In fact, specifically, it uses the ICMP echo and echo-reply protocol, better known as *ping*.

For the Layer 3 health check, the load balancer first sends an ARP request to the server's MAC address. This is just to verify Layer 3 connectivity. Regardless of the response (or lack of response) it gets, it will still try and send an ICMP echo request to the server. The server (assuming all is well on Layers 1, 2, and 3) will send an ICMP echo-reply back to the load balancer. The load balancer will then register the server as being active, and will pass traffic to it when it receives it. If the load balancer does not receive an ICMP echo-reply after the request's *time-to-live (TTL)*, it will consider the server inactive, and it will not pass any traffic to it. At the next interval (on the ServerIron, the default is two seconds), the load balancer will try another health check. After all, it needs to know when the server has become active again. With a ServerIron, the load balancer will try to ping the host four times before it declares it inactive. This is configurable.



Layer 3 health checks are good, but they're not great. If the server loses power or network connectivity, the load balancer will certainly detect that with a Layer 3 health check. But what if the server is active (on Layer 3), but the needed service is not running. For example, what if the server is a web server? The server runs a program on it to listen on TCP port 80 (HTTP). What if that program isn't running? The load balancer will ping the server, and the server will respond (after all, Layer 3 is okay). The load balancer will deem the server active, and pass traffic to it. When the traffic arrives to the server, the server will drop it. It's not listening on the requested TCP port. What can be done?

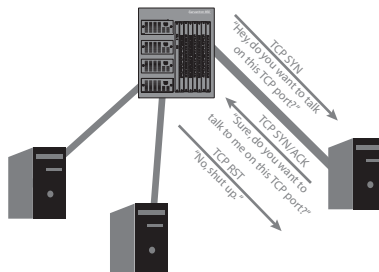
Layer 4

The Layer 4 (or Transport Layer) health check verifies that the first four layers of the server's configuration are functional. In the Brocade ServerIron, if a Layer 4 health check is specified, a Layer 3 health check will also be performed. How the Layer 4 health check is performed depends upon the protocol.

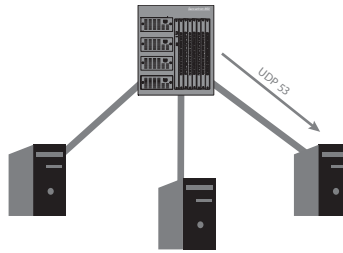
If the protocol defined is TCP, the load balancer will attempt a three-way handshake on the port specified. Specifically, the load balancer will send a TCP SYN segment to the real server (on the destination port specified). If the server responds with a TCP SYN/ACK (as it should), the load balancer will know that the port on the server is actively listening. Seeing as the load balancer knows all that it needs, it will send a TCP RST (reset) message to the server to abruptly terminate the session. This may seem rude, but the load balancer has no need to continue the creation of the session. And it's best if the server's resources aren't tied up any more than they have to be.

If the load balancer sends a TCP SYN and gets no response from the server after a short period of time (five seconds, by default on the ServerIron), or if it receives a "Connection Refused" response (meaning that the server is not listening on that port), it will consider the server inactive, and it will try the health check again later. I always picture the conversation representing a successful TCP health check going this way:

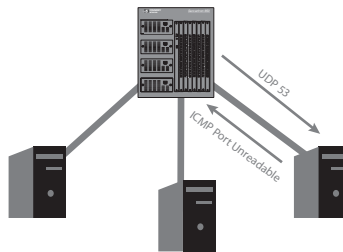
Load Balancer: "Hey, do you want to talk on this TCP port?"
 Server: "Sure, do you want to talk to me on this TCP port?"
 Load Balancer: "No, shut up."



If the protocol defined is UDP, the load balancer has to use a different method. UDP is connectionless. Many UDP protocols do not require replies. In this case, the load balancer will send a UDP segment to the server on the destination port specified. Here, if the load balancer gets no reply (after a short, specified period of time), the load balancer deems the server *active*. If the server is not listening on the destination port specified, it will reply with an ICMP “Port Unreachable” message. If the load balancer sees this, it will mark the server *inactive*. It will try again at the next health check interval.



UDP health check PASSED



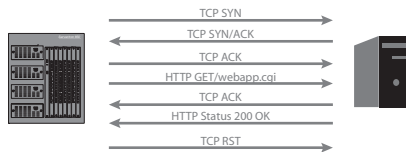
UDP health check FAILED

Combining Layer 3 and Layer 4 health checks is definitely more reliable than just using Layer 3. With Layer 3 and 4, you now know that your server is configured correctly, and it is listening on the TCP or UDP port you're expecting it to. But what if the program is locked? Normally, you'd think if the program were locked, the server would stop listening on the TCP or UDP port it's supposed to, right? Unfortunately, in real life, it doesn't work that way. For example, let's say you are running a web server. You want the customer to be able to use a special web application that is accessed through the web service. The program listening on HTTP (TCP port 80) is one process. When the customer makes a request of your web application, the program listening on HTTP will hand the request to your web application. It's possible (and expected) that the program listening to HTTP will keep responding to TCP port 80 requests, regardless of whether or not your web application is functioning (or even present). Well, that's no good. How will the load balancer know if the server's really active and capable of accepting requests?

Layer 7

Now, we're getting sophisticated. Layer 7 uses Layer 3 and Layer 4, but adds a customized element to verify that all seven layers are functioning as expected. The load balancer can be configured to perform a rudimentary Layer 7 task to verify that the server will respond in a desirable way. This will be different depending on the Layer 7 protocol you're load balancing (i.e., HTTP, DNS, SMTP, etc.).

If we go back to our HTTP example, we've done the Layer 3 check, so the server's responded to a ping. We've done the Layer 4 check, so we know the server is listening on TCP port 80 (HTTP). Now, we'll do the Layer 7 check. This starts out similarly to the Layer 4 check. The load balancer sends a TCP SYN. The server replies with a TCP SYN/ACK. This time, the load balancer finishes the handshake, and sends a TCP ACK. Now the load balancer will make an HTTP request. Let's say the web application is called "webapp.cgi." The load balancer will make an "HTTP GET /webapp.cgi" request of the server. The server will send a TCP ACK (acknowledging the request). The server will then send an "HTTP Status 200 OK" message. This tells the requester that it understood the request, and it is ready and able to reply. Usually before the actual requested data is sent, the load balancer will send a TCP RST (reset) message to the server. It does this because it already knows that the server will comply. It knows that it's operating at Layer 7.



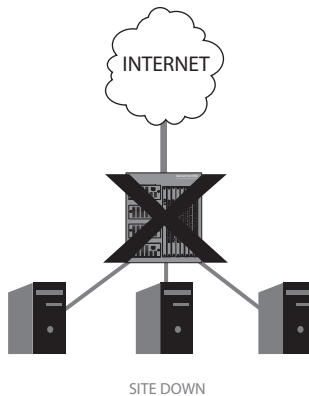
What if the "webapp.cgi" isn't there (for whatever reason)? The load balancer will establish a three-way handshake. It will send an "HTTP GET /webapp.cgi" request to the server. The server will send a TCP ACK, but this time, the server will reply with an "HTTP Status 404 File Not Found" message. The load balancer recognizes the HTTP status codes. It knows that if it does not receive an HTTP 200 status code, something is wrong, and it will deem that server inactive.

But what if you're not load balancing HTTP? No problem. The load balancer can handle all kinds of different Layer 7 protocols, and it will perform its Layer 7 check accordingly. If you're load balancing DNS, it will attempt a name lookup. If you're load balancing FTP, it will make certain it receives an initial greeting with a proper 220 status code. What if you're using a Layer 7 protocol that the load balancer doesn't know? No problem. You can customize your Layer 7 health check to look for a specific text in its response. In other words, you can teach the load balancer what it should expect to see when trying to communicate with the protocol.

Now, we've verified all layers. The Layer 7 health check is the best judge of whether or not a server is really active. However, a Layer 7 health check is not required in all situations. Some applications get easily bogged down if they're constantly queried with these Layer 7 checks. You may find a Layer 4 health check suits your purpose. You may find that even a Layer 3 health check suits your purpose. But if you really want reliability, I'd recommend Layer 7. It uses the reliability of all three checks to make absolutely certain your server is ready to receive requests. And the beauty of the Layer 7 check is that it's as reliable as you make it. It's all up to you.

Redundancy

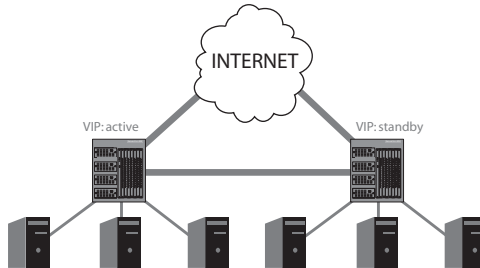
In the world of infrastructure design, there is a term that designers seek to eliminate: *single point of failure*. This means that one device's failure would result in a major loss (or complete loss) of the infrastructure's function. For example, let's say you had a load balancer with a VIP configured and load balancing to several web servers. Your users get to your web site by using the VIP address. All of the sudden, your load balancer loses power. Now, your site is unavailable to your users. The load balancer is a single point of failure.



What's the cure for single point of failure? Redundancy. We need to plan for the inevitable failures in the infrastructure. In our previous example, we need another load balancer. We can configure the load balancers in one of two ways to make them redundant: *Active/Standby* and *Active/Active*.

With *Active/Standby*, you would configure one load balancer with the VIP (as you would normally). You would then configure a second load balancer with the same VIP, but with a twist. You specifically configure this second load balancer not to answer for the VIP. Instead, it watches the other load balancer and performs its own type of health check to verify that the active load balancer is, well, active. When the active load balancer fails, the standby load balancer will start answering for the VIP. The standby load balancer will continue to check the health of the previously active load balancer. If the previously active load

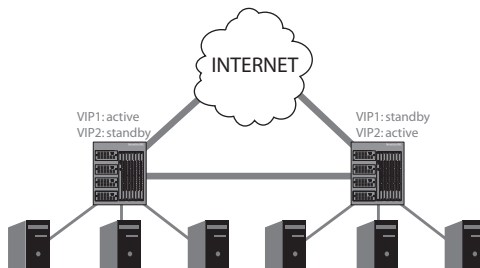
balancer becomes active again, the standby load balancer will stop answering for the VIP, and the now active load balancer will resume answering for the VIP again.



Active/Standby is a good solution, but it does have the drawback of expense. I mean, you paid for two load balancers, but only one is actually being used at any given time. Wouldn't it be great if you could actually use both of them at the same time? Enter Active/Active.

Active/Active can be achieved in one of two ways. One method involves creating two alternating Active/Standby VIPs. The idea is that instead of using one VIP, you would use two. Your users would use both VIPs simultaneously. This is commonly done using a DNS name to resolve to two IP addresses, instead of one. DNS inherently load balances which IP address will resolve to the user, and approximately 50% of your traffic will go to one VIP, and 50% of your traffic will go to the other. It's never precisely that, but it's close.

For VIP1, you would configure Load Balancer A to be active, and Load Balancer B to be standby. For VIP2, you would configure Load Balancer B to be active, and Load Balancer A to be standby. This way, both load balancers would receive traffic. Both load balancers would be actively functioning. They would also both be keeping an eye on each other. If one became unavailable, the other would answer for both VIPs (until the unavailable load balancer became active again).



The second method for achieving Active/Active redundancy involves two load balancers and just one VIP. In this case, one load balancer actively listens for incoming traffic for the VIP, but offloads some of the incoming traffic to the

second load balancer. If the active load balancer becomes unavailable, the second load balancer answers for the VIP. This method is sometimes referred to as “true Active/Active.”

The disadvantage with true Active/Active is that your bandwidth is still directed through only one load balancer. This may not be a critical component for all sites, but it's something to consider for larger, busier sites.

We've gone over examples where you would use two load balancers, but you can use three, four, ten, as many as you need. Each can be configured with their own order of failover. For example, with three load balancers, you could have Load Balancer A be the active, Load Balancer B as standby to A, and Load Balancer C as standby to B. The more load balancers you have acting redundantly, the less likely your site will ever become unavailable. There comes a point, of course, where the expense and complexity outweighs the risk. It all depends on your environment.

Summary

- A Virtual IP (VIP) address is a virtual address that the load balancer uses to share load over several physical devices
- Proxy load balancing terminates the Layer 4 session and initiates a new session to the physical machines it is load balancing
- Layer 4 Switching load balances based on the Layer 4 protocol (TCP/UDP) and destination port
- Layer 7 Switching is based on the Application-layer protocol; switching decisions are made depending on a set of rules defined by the user
- Server Load Balancing (SLB) changes the destination IP and MAC address of an incoming packet to the IP and MAC address of the desired physical server
 - SLB changes the source IP and MAC address of an outgoing packet to the IP and MAC address of the VIP, so that the end user believes that the session was only with the VIP
 - SLB requires all physical servers to be physically connected to the load balancer (or network engineered in such a way to route all traffic through the load balancer); all physical server traffic must flow into and out of the load balancer
- Source NAT changes an incoming packet's source addresses (IP and MAC) to those of the load balancer and the destination addresses (IP and MAC) to those of the desired physical server
 - Source NAT changes an outgoing packet's source addresses (IP and MAC) to those of the VIP, and the destination addresses (IP and MAC) to those of the original client
 - Source NAT does not require the physical servers to be physically attached directly to the load balancer

- Direct Server Return (DSR) changes only the destination MAC address of the incoming packet; it does not see the outgoing packet
 - DSR requires all physical servers to have a loopback address configured with the same IP address as the VIP
 - In DSR, the outgoing packet (the reply from the server) travels directly from the server to the original client; the load balancer does not see the reply
- Health check is the method the load balancer uses to make sure a physical server may receive incoming traffic
 - Active health checks are categorized as either initialization or periodic
 - A Layer 3 health check uses ARP and ICMP echo (“ping”) to see if a host is active
 - A Layer 4 health check for TCP uses a modified three-way handshake - SYN - SYN/ACK - RST
 - A Layer 4 health check for UDP sends a segment, and looks for an ICMP “Port unreachable” message; if it doesn't see this message, the server is active; if it does see the message, the server is inactive
 - A Layer 7 health check uses a user-defined Application-layer request to see if the server responds properly
 - If a Layer 4 health check is defined, a Layer 3 health check will be performed also
 - If a Layer 7 health check is defined, a Layer 3 and Layer 4 health check will be performed also
- Active/Standby Redundancy requires two or more load balancers; only one load balancer will actively listen for the VIP at any time.
- Active/Active Redundancy requires two or more load balancers; one Active/Active method involves multiple Active/Standby VIPs, requiring two or more VIPs; “true Active/Active” requires only one VIP, but only one load balancer actively listens for the VIP.

Chapter Review Questions

1. Which load balancing method will give your end users the greatest bandwidth and number of sessions?
 - a. SLB
 - b. DSR
 - c. Source NAT
 - d. Proxy

2. You have only two load balancers and one VIP. What kind of redundancy could you use?
 - a. Active/Standby
 - b. Active/Active
 - c. Both A and B
 - d. Standby/Active
3. For an incoming packet, a load balancer configured for Source NAT changes:
 - a. The Destination MAC address only
 - b. The Destination IP and MAC address only
 - c. The Source MAC address and the Destination IP and MAC address only
 - d. The Source IP and MAC address and the Destination IP and MAC address
4. In a packet capture, I see a TCP SYN, a TCP SYN/ACK, and a TCP RST. What kind of health check did I witness?
 - a. Layer 4
 - b. Layer 3
 - c. Layer 7
 - d. Layer 2
5. For an incoming packet, a load balancer configured for SLB changes:
 - a. The Destination MAC address only
 - b. The Destination IP and MAC address only
 - c. The Source IP address and the Destination IP and MAC address only
 - d. The Source IP and MAC address and the Destination IP and MAC address
6. You have two load balancers and two VIPs. What kind of redundancy could you use?
 - a. Layer 3
 - b. Active/Active
 - c. Standby/Active
 - d. Both B and C

7. Your load balancer believes that one of your physical servers is down. In a packet capture, you see the following:
- ICMP Echo
 - ICMP Echo Reply
 - UDP packet
 - ICMP Port Unreachable
- What's the problem?
- a. Your physical server failed Layer 3 health check
 - b. Your physical server failed Layer 7 health check
 - c. Your physical server is up, but is not listening on the UDP port specified by the health check
 - d. Your physical server is off or has been disconnected from the network
8. For an incoming packet, a load balancer configured for DSR changes:
- a. The Destination MAC address only
 - b. The Destination IP and MAC address only
 - c. The Source MAC address and the Destination IP and MAC address only
 - d. The Source IP and MAC address and the Destination IP and MAC address
9. In a packet capture, you see the following:
- ICMP Echo
 - ICMP Echo Reply
 - TCP SYN
 - TCP SYN/ACK
 - TCP ACK
 - HTTP GET /index.html
 - TCP ACK
 - HTTP STATUS 200 OK
 - TCP RST

- What type of health check have you just witnessed?
- Layer 3
 - Layer 4
 - Layer 7
 - All of the above
10. Which load balancing method requires the physical servers to be configured with a loopback?
- DSR
 - SLB
 - Source NAT
 - Proxy

Answers to Review Questions

- b. DSR uses the bandwidth of the individual server, not the load balancer. With SLB, Source NAT, and Proxy, all traffic travels into and out of the load balancer.
- c. With one VIP, you could either configure Active/Standby, or true Active/Active. In either case, only one load balancer actively listens for the single VIP.
- d. Source NAT changes the incoming packet so that it appears to come from the load balancer to the physical server (rather than the client to the VIP). It has to appear to come from the load balancer, so that the return traffic will come back through the load balancer as well.
- a. This is a TCP Layer 4 health check. The load balancer sends a TCP SYN. The physical server sends a TCP SYN/ACK. Then, the load balancer finishes the discussion by sending a TCP RST (reset).
- b. The load balancer changes the destination IP and MAC address to be the addresses of the physical server it has chosen (as opposed to the VIP).
- d. You could use Active/Standby, but if you have two VIPs, you would be well-served to configure Active/Active.
- c. This is the way the load balancer will health check UDP. If it receives an ICMP Port Unreachable message, it knows that the physical server is not listening on the specified UDP port, and will mark the server inactive. For the UDP Layer 4 health check to succeed, the load balancer must send the UDP segment, and not receive any reply.
- a. In DSR, the load balancer only changes the destination MAC address to be the MAC address of the physical server it has chosen.

9. d. You get partial credit if you answered C. You see a ping, a TCP three-way handshake, and a Layer 7 request and response. All three health check methods are represented. This occurs when you have configured a Layer 7 health check.
10. a. In DSR, the destination MAC address is the only thing that's changed on the incoming packet. If the physical server is not configured with a loopback (that is the same IP address as the VIP), it will not accept the incoming packet (it will assume it is meant for some other host).

Part Two: Switching

The following chapters are included in Part Two:

- [“Chapter 6: The Brocade CLI”](#) starting on page 119
- [“Chapter 7: Link Aggregation \(Trunks\)”](#) starting on page 159
- [“Chapter 8: Virtual Local Area Network \(VLAN\)”](#) starting on page 169
- [“Chapter 9: Spanning Tree Protocol”](#) starting on page 193



The Brocade CLI

We've gone over the basics. Now, it's time to start getting our hands dirty, so to speak. In this section, we're going to start at Layer 2 (Data Link) and work our way up again. But this time, we're going to delve into more specifics. We're going to start with the Brocade Command Line Interface (Brocade CLI). This is the text-based environment that allows you to program a Brocade switch.

Although, the Brocade CLI may appear to be cryptic and intimidating at first, I think you'll find that it quickly becomes very natural. Brocade has done their best to make the CLI very easy to use.

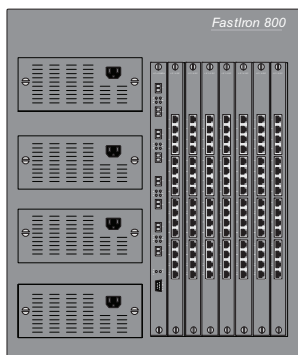
Let's start with a review of the switches themselves.

Switch Layout

As we talked about in Chapter 1, many Brocade switches come in two varieties: *chassis* and *stackable*.

Chassis

The chassis will have at least one management blade, and any number of interface blades (depending on how many slots the particular chassis has).



Notice that the management blade is the only blade that contains a DB-9 serial console port. This is the blade that we will interface with when we configure the switch. Traditionally, the management blade will be installed in the first

available slot. This is not required. The chassis also supports a second, or failover, management blade. This blade will act as an active/standby failover blade to the active management blade. This second management blade has its own serial port, and can actually be accessed independently of the active management blade. Because the two blades are automatically synchronized, there's little reason to access the second blade...unless it becomes active (due to a failure of the primary management blade).

Stackable

The stackable has a DB-9 serial console port as well, but it's typically part of the body. The RJ45 copper ports are also fixed, but you can see that there are often ports to house mini-GBICs. These ports are usually shared with an embedded copper port. For example, you'll see the copper labeled as Port 1 and the mini-GBIC port labeled as Port 1F. You can use one or the other, but you can't use both at the same time. In this example, you can use Port 1 or you can use Port 1F, but not both simultaneously. By default, you'll be able to use the copper port. When a mini-GBIC is installed, it's assumed that you want to use the fiber port.



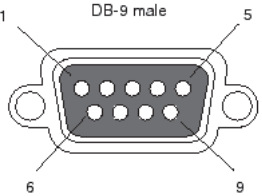
Accessing the CLI

Knowing the CLI is fine, but if you can't actually access it, this knowledge does you very little good. The CLI can be accessed in one of two main ways: via *serial* and via *network*.

Serial

With serial, the communication happens using the DB-9 serial console port on the switch or management blade. The most basic way to access this port is to connect a serial straight-through cable (not a null-modem cable) from your PC or laptop's serial port to the serial console port on the switch. This is the physical setup, but now you'll need a *terminal emulator* program running on your workstation or laptop. Common terminal emulators are HyperTerminal, ZTerm, and, my favorite, minicom. There are many others. The program will need the following configurations:

- Terminal Type — VT-100 [other terminals will work, but you're safest using VT-100]
- Data Rate — 9600 baud
- Data Bits — 8
- Parity — None
- Stop Bits — 1
- Flow Control — None

Pin Assignment	Pin Number	Switch Signal
	1	Reserved
	2	TXD (output)
	3	RXD (input)
	4	Reserved
	5	GND
	6	Reserved
	7	CTS (input)
	8	RTS (output)
	9	Reserved

This is not the only way to use the serial port. For example, you can connect the serial port to a modem. This will allow you to dial into the switch using a PC with a modem. To dial in, you will also need a terminal emulator program of some kind. You can connect the serial port to a serial switch. This would allow you to remotely connect to the switch, and then tunnel your communication through the serial port. You could also connect the serial port to an actual terminal. You have many, many options.

Why use the serial port at all? This allows you to communicate with the CLI without depending on the network configuration being correct (or even functional). This is referred to as *out-of-band* access. Think of this as simply an alternative way to communicate to a device. If the network was down, for whatever reason, you could still communicate with the switch using the serial port. The serial port becomes critical on a brand new switch (as there's no configuration).

Network

For network communication, there are currently two options: Telnet and SSH. Telnet is enabled by default. It's a protocol that uses TCP port 23. It's very easy to use, and most popular operating systems include a Telnet client. The downside is that it's insecure. All of the data (including any username or password information) in the communication is passed in clear text. Anyone capturing packets would be able to see all the data in the session.

The second option is Secure SHell (SSH). This is not enabled by default. SSH provides the same kinds of access as Telnet, but it is much more secure. All the data in the session is encrypted. To enable SSH, you need to do the following:

- Configure at least one username and password
- Configure an authentication mechanism for the switch (more on this later)
- Configure a DNS domain for the switch (this is used as a seed to generate the SSH keys)
- Generate the SSH keys

This needs to be done only once. More usernames and passwords can be added or deleted later.

Online Help

Whichever method you use to access the CLI, when you finally do, it will look something like this:

```
Switch>
```

The “Switch” portion is the name of the device. The “>” tells you that you are in *User mode* (more on this later).

Now what? Well, fortunately, the Brocade switch will help you answer that question. The Brocade CLI provides built-in online help. You access the help by using the question mark (“?”). To display a list of available commands or command options, enter ? or press Tab. If you have not entered part of a command at the command prompt, all the commands supported at the current CLI level are listed. If you enter part of a command, then enter ? or press Tab, the CLI lists the options you can enter at the point in the command string. Look what happens when you enter ? at the prompt:

```
Switch>?  
enable          Enter Privileged mode  
ping            Ping IP node  
show            Display system information  
stop-traceroute Stop current TraceRoute  
traceroute      TraceRoute to IP node  
Switch>
```

The CLI is showing you a list of legitimate commands. This list is pretty short, but that's the nature of User mode. The important thing to notice is that it gives you the command, followed by a description of what the command does.

You can issue some commands by themselves, as a single word. Other commands require additional information. For example, the **ping** command will issue an ICMP echo message to whatever address you want to send it. But to do this, it has to know the address. How do you know if a command needs more information? Remember the good old question mark. You can use the question mark with an individual command to see all available options for that command or to check context. To view possible **ping** command options, enter the following:

```
Switch>ping ?  
ASCII string  Host name  
A.B.C.D       Host IP address  
Switch>ping
```

Notice that there's a space between the “ping” and the “?” The CLI explains what it needs next. In this case, it needs a host's name (in the form of an ASCII string) or an IP address. Notice that it kept the **ping** command waiting for you at the next line.

For your convenience, you can abbreviate commands. The CLI will let you abbreviate up to the next distinct letter. You could do this:

```
Switch>pi ?
ASCII string      Host name
A.B.C.D           Host IP address
Switch>pi
```

Or even this:

```
Switch>p ?
ASCII string      Host name
A.B.C.D           Host IP address
Switch>p
```

The switch knows that when you issue the command **p**, you really mean **ping**. It knows this because there are no other available commands that begin with the letter “p.” What about the letter “s?”

```
Switch>s ?
Ambiguous input -> s ?
Switch>
```

Ambiguous input? What does that mean? It means the CLI can't distinguish which command you meant by just typing the letter “s.” Let's see what our options are:

```
Switch>s?
show                Display system information
stop-traceroute     Stop current TraceRoute
Switch>
```

Notice that there is no space between the **s** and the **?** this time. That's important. You're not asking to see if a command needs more information. You're asking to see all of the “s” commands (or commands that start with the letter “s”). You can see that we have two commands to choose from. So, what if we tried it again, giving it a little more information (one more letter, to be precise):

```
Switch>st ?
<cr>
Switch>st
```

It didn't give us the “Ambiguous input” alert because it knew which command we meant. We added a “t,” and there's only one available command that starts with “st.” Now, what does <cr> mean? It means <carriage return>. Well, what does that mean? It means there's no further information you need to feed the command. You can simply type **st** and press Enter. That's the end of the line.

For convenience, Brocade also keeps a command history. The CLI keeps track of all of the commands you've issued in a given session. To refer to a command that you previously typed, press the up-arrow key. You can press the up-arrow key multiple times to go further back in the command history. If you overshoot the command you were looking for, the down-arrow key will sift through the command history in the opposite direction (e.g., more recent commands).

Configuring by Context

We talked a little bit about the different *modes* you can be in when you're using the CLI. The first mode we saw was User mode. This mode has very few commands. It's meant to provide access for people who may need to be able to ping and traceroute, but should not be allowed to change the configuration of the switch.

The next mode is called *Privileged mode*. In this mode, you have all the same commands as User mode along with a whole bunch of other commands. This is the administrator's mode. This mode will allow you to change the switch's configuration, reboot the switch, perform troubleshooting, etc. For those familiar with UNIX-based systems, this would be very similar to logging into a system as the *root* user.

You get to Privileged mode by typing the **enable** command. When you type the **enable** command, you will most likely be prompted for a password. If this is a brand new switch, it will not ask for a password. Given that this mode will allow you to fully administer your switch, it is always a very good idea to require a password to get into Privileged mode (more on this later in the chapter):

```
Switch>enable
Password:
Switch#
```

Notice that the prompt has changed. You're no longer seeing the ">" after the switch's name. It's been replaced by a "#". The "#" tells you that you are in Privileged mode. You might also notice that you do not see any characters representing the password. The switch does receive your password. It just doesn't display it while you're typing it.

The next mode is called the *Global config mode*. Here, you are telling the switch that you'd like to change something about its configuration. Maybe you want to change its hostname. Maybe you want to set up usernames or change passwords. There are many possibilities. Much of what we will do in the remaining chapters of this book will be in the Global config mode. This mode is where the most significant changes are made.

To get into the Global config mode, you must first be in Privileged mode. Then, you may issue the command **configure terminal**:

```
Switch#configure terminal
Switch(config)#
```

Don't think of this command as telling the switch that you want to configure the terminal. You want to enter the terminal that allows you to configure. Notice that the prompt changed again. Instead of just the "#" character, you see "(config)#." This is to tell you that you're in Global config mode.

Now, there are very few, if any, experienced Network Engineers that will actually take the time to type **configure terminal**.

Probably the most recognized abbreviation is as follows:

```
Switch#conf t
Switch(config)#
```

You may hear engineers referring to “conf t”. This is their abbreviated way of saying “enter the configuration terminal” or Global config mode.

Global config mode has several sub modes, and we'll see many of them throughout this book. There are modes to configure individual interfaces. There are modes to configure specifications for a VLAN. There are modes to define the details of a routing protocol, and many, many more. Just remember the “(config)”. If you see that anywhere in your prompt, you're in Global config mode, or one of its sub modes.

You can escape any mode (Global config mode, Privileged mode, User mode, etc.) by typing the command **exit**:

```
Switch(config)#exit
Switch#exit
Switch>exit
```

With that last **exit** command, you'll be disconnected from the CLI.

The config

The config is the configuration file for the switch. This is a very simple, text file. It is saved to the switch's flash memory. To see the config, use the following command:

```
Switch#show config
!
Startup-config data location is flash memory
!
Startup configuration:
!
ver 03.0.01cT3e1
!
!
!
!
!
!
!
!
!
hostname Switch
!
!
!
!
!
end
```


This is a very simple config. The **show config** command tells the switch to display the configuration file as it exists in flash memory. The config is simply a list of settings that change the switch's behavior.

In the above example, there really is only one setting that will change the switch's behavior. The "!" are blank lines. They simply space out the lines of the configuration to make it easier to read. The first text line tells us that the configuration file is stored in flash. Then, the startup configuration starts. The line "ver 03.0.01cT3e1" tells you what version of the Brocade OS you're currently running. Now, we finally get to the one setting that changes this switch's behavior: "hostname Switch." The **hostname** command sets the name of the switch. This is purely for your own reference. Finally, you see the "end" of the config.

Again, this is a very simple config. The more configuration options you change from the default, the more items will appear in your config file. You make changes to your config file by entering the Global config mode.

But there are actually two versions of a config on a running switch. As I mentioned, the config is stored in the switch's flash memory. When the switch boots, it copies that config file from flash memory into RAM. Like a computer, RAM is memory that the switch actively uses while it has power (while it's "on"). It's operational memory. When you go into the Global config mode, you are making changes to the config that's in RAM, not the config that's in the switch's flash memory. If you were to make a change to the config, and then pull the plug (causing the switch to lose power), and then boot the switch up, you would notice that your change would not be there. RAM is very fast memory, but its contents only exist when there's electricity applied to it. When that electricity is gone, so are the contents of the RAM.

To make a permanent change, you need to copy the config file in RAM to overwrite the config file in flash:

```
Switch#write memory
```

Or, even shorter:

```
Switch#write mem
```

You may often hear experienced network engineers refer to performing a **write mem**. You're simply saving your changes to the config from volatile memory (RAM) to more permanent memory (flash). On most switches, the shortest abbreviation would be **wr m**. It accomplishes the same thing.

To see the config in RAM, you can use:

```
Switch#show running-config
```

Or, for short:

```
Switch#show run
```

And this one will also work:

```
Switch#write terminal
```

All three of these commands do the same thing. They simply display the config file that currently resides in RAM. They display the “active” config.

What if you want to just start over? You want a clean slate. Here's the command to use with caution:

```
Switch#erase startup-config
```

This will erase the config file stored in flash. This will not erase the running-config file (the config file in RAM). To finish the job, you will need to reload the switch. This is like a computer's restart or reboot function. It forces the switch to boot. This time, when it loads the config file from flash to RAM, it will be empty.

Let's recap what we've learned:

- show running-config — Displays the config file in RAM (or “active” config)
- show configuration — Displays the config file in flash (or “startup” config)
- write memory — Copies the config file in RAM to flash
- erase startup-config — Clears the config file in flash

Software (and where it's stored)

Brocade offers three different versions of software to run on most switches:

- Switch Code
- Base Layer-3 Code
- Full Router Code

To find out which code your switch is running, you can issue this command:

```
Switch#show version
SW: Version 03.3.01eTc1 Copyright (c) 1996-2004 Foundry
Networks, Inc.
Compiled on May 24 2005 at 10:53:13 labeled as FES03301e
(2059596 bytes) from Secondary FES03301e.bin
Boot Monitor: Version 03.1.02Tc4
HW: Stackable FES4802
=====
Serial #:
330 MHz Power PC processor 8245 (version 129/1014) 66 MHz bus
512 KB boot flash memory
16384 KB code flash memory
128 MB DRAM
The system uptime is 370 days 8 hours 25 minutes 16 seconds
The system : started=warm start    reloaded=by "reload"
Switch#
```

This switch, as the command reveals, is a Brocade FastIron Edge Switch (FES) 4802. It is running the Brocade OS (for the FES) version 03.3.01e. The “Tc1” following the version number is for tracking within Brocade. The “c” indicates a hardware type, and the “1” indicates a software platform. For the end user, you need only concern yourself with the version number.

Notice that the next line tells you that the compiled binary was labeled as “FES03301e.” Brocade names their software images deliberately. The first three letters (in this case “FES”) will describe the type of software compiled. In this example, “FES” should be interpreted as “FastIron Edge Switch” code, specifically “Switch Code.” Base Layer-3 code for this particular switch would be labeled “FEL,” and Full Router Code would be labeled as “FER.”

Not all models are this straight forward. A chassis, for instance, has its own (but consistent) naming schema for the first three letters. Switch code would be “B2S.” Base Layer-3 would be “BL3.” Full Router Code would be “B2R.”

As the product lines continue to evolve, your safest bet for determining the type of code can be found in the release notes for your particular device's software. These can be obtained from the Brocade web site (<http://www.brocade.com/>).

There is one more trick that you should be aware of. If a switch is running either Base Layer-3 or Full Router code, there will be an additional few letters to the prompt. For example, if you are running switch code, you would see this prompt:

```
Switch#
```

If you were to install and run Base Layer-3 or Full Router code, the prompt would look like this:

```
BR-Switch#
```

Notice the addition of the prefix “BR-” to the switch's name. This is an indicator to the user that they are using some form of Layer 3 code on the switch.

The **show version** command also reveals other useful bits of information. You can see the switch's serial number (the actual number was removed from the published output), the switch's uptime (how long it's been since the last boot), and the cause of the last boot (in this example, an administrator typed **reload**).

Now, where is the software stored? We already know that the switches contain flash memory, and that the config is stored there. This is also where the OS is stored.

Let's look at another command:

```
Switch#show flash
Compressed Pri Code size = 1888271, Version 03.1.02bTc1
(fes03102b.bin)
Compressed Sec Code size = 2059596, Version 03.3.01eTc1
(FES03301e.bin)
Compressed Boot-Monitor Image size = 45598, Version
03.1.02aTc4
Compressed Boot-Tftp Code size = 184112, Version 03.1.02aTc5
Default BootROM Contents Info:
    Boot Loader Rev: 1a
    Monitor Image Version: 01.00.00Tc4
    Boot-Tftp Image Version: 01.00.00Tc5
Code Flash Free Space = 12386304
```

Notice that this command refers to a “Pri Code” (or “Primary Code”) and a “Sec Code” (or “Secondary Code”). Each switch has two locations to store software code: primary flash and secondary flash. This is of crucial convenience.

Let's say I wanted to upgrade to a new version of code for my switch. I am currently using my old version, 1.2.0a, and I want to upgrade to 1.3.0d. The software binary file for 1.2.0a is stored in my switch's primary flash. I copy 1.3.0d into my switch's secondary flash, and I tell the switch to boot off of secondary flash. When I reboot, I discover that there's a problem with 1.3.0d that is making it incompatible with my environment. Instead of having to reload 1.2.0a, I can simply tell my switch to boot off of primary flash (the location of 1.2.0a), and all is restored back to normal. The idea of primary and secondary flash is that they allow you to move to new versions of code, but to keep the last known functioning version of code handy, just in case.

Perhaps we should take a moment to discuss the boot process of the switch. In hardware, there is a boot ROM. This contains very basic code that instructs the switch how to boot. The boot ROM calls the Boot-Monitor. This is stored in flash (in its own space, not primary or secondary). This is also a very basic structure, but it gives you the opportunity to perform basic diagnostic functions before loading the actual OS. It also allows you to configure your switch to boot off of primary flash, secondary flash, or even a TFTP server.

Next, the Boot-Monitor loads the config file from flash, and checks to see if there are any special instructions on whether to boot from the primary or secondary flash. Finally, the OS from either primary or secondary flash is loaded. The OS applies the settings in the config file, and the switch boot process is finished.

To get into the Boot-Monitor, you must be connected to the serial console on the switch. You cannot use the Boot-Monitor via Telnet or SSH. After turning on (or rebooting) the switch, within 2 seconds, press the “b” key. This will interrupt the boot process and put you into the Boot-Monitor. The monitor has very limited commands, and it is beyond the scope of this book to go into full detail of the available commands. To escape the Boot-Monitor and continue with the

boot process, simply type **boot**. If you want to boot from a different flash, you can use the command **boot system flash primary** or **boot system flash secondary**.

Let's say you're already booting from primary flash. You'd like to copy some new code to secondary flash from a TFTP server.

```
Switch#copy tftp flash 10.0.0.16 FES03601a.bin secondary
```

The **copy** command, in this example, specifies that you want to copy some data from a TFTP server and you want to copy it to flash. You don't specify which flash until the very end of the line. In this example, we are copying to secondary flash. The TFTP server's IP address is 10.0.0.16, and the software file that we are requesting is FES03601a.bin. Assuming that the switch is able to reach the TFTP server, the software will copy to the secondary flash.

When the copy is completed, you will want to tell the switch to boot from the code in secondary flash. You could use the serial port, enter the Boot-Monitor, and specify it to boot from secondary flash. This is a fair amount of work. Also, if the switch is rebooted or loses power, it will still boot from primary flash. The Boot-Monitor method is for a one-time change in the boot path.

The most effective way to accomplish our design is to change the config:

```
Switch#conf t
Switch(config)#boot system flash secondary
```

Be sure to issue a **write mem** command to save this change to flash. Now, you simply reload the switch. When the Boot-Monitor loads the config, it will see that you've requested to boot from secondary flash, and it will call the code in secondary flash.

But what if your switch is in use, and you can't reload it right away? No problem. If you'd like, you can actually schedule the switch to reload at a future time.

```
Switch#conf t
Switch(config)#reload at 01:00:00 12-21-10
```

This command is telling the switch to reload at 1:00 AM on December 21, 2010. Remember to **write mem** afterwards.

Remember that the config is used to change default settings to customized settings. The default setting in any switch is to boot from primary flash. There is no config command for **boot system flash primary**. If you change your mind, and you want to boot from primary flash again, you need to remove the **boot** command from the config. To remove almost any command from the config, simply put a **no** in front of it:

```
Switch#conf t
Switch(config)#no boot system flash secondary
```

This will remove the **boot system flash secondary** from the config, and, as long as you performed a **write mem** afterwards, the switch will now boot from primary flash (on the next reload).

The **copy** command can be used to back up software on the switch as well. Notice the difference in this command:

```
Switch#copy flash tftp 10.0.0.16 FES03601a.bin secondary
```

This copies the software in secondary flash to the TFTP server. By the way, on some TFTP servers, you need to create an empty file to write to before copying something to the server. Consult your server's documentation.

You can even copy software from one flash into the other. This command copies the contents of primary flash into secondary:

```
Switch#copy flash flash secondary
```

And this copies the contents of secondary flash into primary:

```
Switch#copy flash flash primary
```

The important thing to remember is that you're specifying the destination, not the source. The source is assumed. If you get confused, you always have contextual help to guide you.

The **copy** command can also be used to back up or restore the config file:

```
Switch#copy running-config tftp 10.0.0.16 running-cfg.txt
```

```
Switch#copy startup-config tftp 10.0.0.16 startup-cfg.txt
```

These commands will back up the running and startup config. To restore, just reverse the process:

```
Switch#copy tftp running-config 10.0.0.16 running-cfg.txt
```

```
Switch#copy tftp startup-config 10.0.0.16 startup-cfg.txt
```

I don't recommend copying from a TFTP server directly to the running-config. It's much safer to copy it to the startup-config, and reload the switch.

show commands

We've already seen a few of **show** commands. There are many (too many to put in this book). Here's a bunch that may come in handy:

- **show version** — Displays software version, uptime, the reason for the last reboot, the device's serial number and more
- **show interface** — Displays interface statistics, including whether an interface is up or down, speed of the interface, errors, throughput, etc.
- **show statistics** — Displays throughput and error rates for each interface in an easy-to-read table
- **show ip** — Displays IP configuration information
- **show ip route** — Displays the IP routing table (if you're running Base Layer-3 or Full Routing code)
- **show ipx route** — Displays the IPX routing table (if you're running Base Layer-3 or Full Routing code)
- **show span** — Displays spanning-tree information

- `show mac-address` — Displays the switch's MAC address table
- `show mac-address stat` — Displays a table that shows the number of MAC addresses the switch has learned on each interface
- `show media` — Displays the physical media type of each interface of the switch; “C” means an integrated copper Ethernet port; “M-SX” means this interface has a mini-GBIC that is multimode (SX) fiber; “M-LX” means this interface has a mini-GBIC that is single mode (LX) fiber; “M-TX” means this interface has a mini-GBIC that is a copper Ethernet interface
- `show arp` — Displays the ARP table; this will show all of the IP address/MAC address matchings that the switch currently knows about; it will match each entry with the physical interface it was learned on
- `show logging` — Displays the switches log
- `show flash` — Displays the file contents of primary, secondary, and boot-monitor flash
- `show vlan` — Displays information pertaining to configured VLANs
- `show telnet` — Displays active Telnet sessions into the switch, including the source IP address of each session
- `show trunk` — Displays information about configured trunks (see [Chapter 7](#))
- `show tech-support` — Displays most information a Brocade TAC engineer would need to troubleshoot problems on your switch

This last one deserves a little more discussion. You might remember reading about it in Chapter 1. It is more commonly referred to as “show tech.” This is actually a combination of several “show” commands. It is a culmination of all relevant information that a Brocade TAC engineer would need to help you solve your problem (or at least start the diagnosis). For many troubleshooting cases, you may want to collect the output of this command before you even open the case with Brocade TAC. Odds are they will request it anyway. To collect the output, you can either save the log of your session (see the documentation of your terminal program), or you may be able to copy and paste the output into a blank text file. Either way, the end result should be a text file containing the output of the command. It is this file that Brocade TAC will be interested in. You can upload this file to your case using the Brocade web site (<http://www.brocade.com/>).

There's one more show command that deserves a little extra explanation as well. This can be one of the handiest troubleshooting tools in the “show” arsenal:

```
Switch#show interface brief
Port Link State Dupl Speed Trunk Tag Prior MAC Name
1 Up Forward Full 1G 1 Yes level0 000c.db6d.4480
2 Up Forward Full 1G 1 Yes level0 000c.db6d.4480
3 Down None None None None No level0 000c.db6d.4482
4 Down None None None None No level0 000c.db6d.4483
5 Up Forward Full 1G None No level0 000c.db6d.4484 MailServer
6 Up Forward Full 1G None No level0 000c.db6d.4485 DNS Server
```

Port. The interface number. On a chassis, this will be in the form of “slot/port”; for example, “1/3” would mean “slot 1 interface 3”

Link. This is the Physical Layer status of the interface. Is it up? Is it down?

State. This is the Spanning Tree state (more on this in Chapter 7)

Dupl. Short for “Duplex”. Here you should see “Full” or “Half”

Speed. The interface's speed. Possible values include “10M” (10 Mbps), “100M” (100 Mbps), “1G” (1 Gbps), and more.

Trunk. Is this interface a member of a trunk? If so, what is the trunk number that it belongs to? Notice that ports 1 and 2, in this example, are members of trunk number 1.

Tag. Is this interface 802.1q-tagged or not? More on this in Chapter 6.

Priority. What is the “Quality of Service” (QoS) priority level of this interface? More on this in Chapter 11.

MAC. The MAC address of the port. This should not be confused with MAC addresses learned on the port. Each switch port has its own MAC address. Notice that ports 1 and 2 share a MAC address, as they are both members of the same static trunk.

Name. This is an arbitrary name that you assign to an interface. It is purely for your reference and convenience.

clear commands

The **clear** command is generally used to clear counters and tables of old information. Here are a few of the more handy clear commands:

- **clear arp** — Empties the switch's ARP table; this can be used to make certain the switch's ARP information is current; when it is cleared, the switch will simply begin populating the ARP table again
- **clear mac-address** — Empties the switch's MAC table; this is used to make certain that the switch's information is current; this information will also repopulate (and typically very quickly)

- clear statistics — This empties the interface statistics (throughput, errors, etc.); this can be used to clear statistics for all interfaces, or for a specific interface; this is very useful if you want to quickly determine whether errors shown are old or new
- clear logging — Empties the switch's log
- clear ip route — This is only be available if you're running Base Layer-3 or Full Routing code; this command empties out the IP routing table, allowing the switch to repopulate it with current information

Greetings

Every now and then, you'll find it useful to display a greeting to the end user. In the Brocade OS, these are called *banners*. There are three different types of banners: motd, exec, and incoming.

The “motd” (“message of the day”) banner is a message that displays whenever someone logs in to the CLI remotely (e.g., via Telnet or SSH). Often, network administrators use this banner to issue a warning to those who may have logged in accidentally:

```
Warning: All changes to this switch must be preapproved by the
Director of Network Engineering. Termination may result if
this warning is unheeded.
SSH@Switch#
```

The “exec” banner is a message that displays whenever the user is in the Privileged EXEC mode. This space is commonly used for warnings, but you can put anything you like here.

The “incoming” banner is unique. This is a message that is displayed to the serial console, and is triggered by someone logging in via Telnet or SSH. This is a simple way to keep track of when (and from where) someone logs in. When this banner is configured it automatically displays the incoming IP address that initiated the session. It also displays the protocol used (Telnet or SSH). This is followed by whatever text you put in the incoming banner (e.g., “Incoming Remote Session! Alert the Network Engineer!”).

The banners are all configured in a very similar way. From the global config prompt, you use the **banner** command. This is followed by the type of banner (“motd”, “exec” or “incoming”; if no banner type is specified, it assumes “motd”). Finally, you would give it a character to tell it that you've finished your message. The message can be multiple lines long, and can be up to 2,048 characters.

Here's an example to correspond with our previously demonstrated motd:

```
Switch#conf t
Switch(config)#banner motd $
Enter TEXT message, End with the character '$'.
Warning: All changes to this switch must be preapproved by the
Director of Network Engineering. Termination may result if
this warning is unheeded.
$
Switch(config)#
```

I chose the “\$” character as my delimiting character. You could just as easily choose “#” or “.” or even the letter “C.” The trick is to not choose a character that you really need in the message. My message didn't have any “\$” in it, so that was a safe character to use.

To remove the banner, employ the “no” rule:

```
Switch#conf t
Switch(config)#no banner motd
```

Configuring the Interface

Every interface on a switch has its own configuration. You can specify speed, duplex, an interface name, and, if you're running Base Layer-3 or Full Router code, you can even specify an IP address. To configure an interface, you must be in *interface configuration* mode:

```
Switch#conf t
Switch(config)#interface e 1
Switch(config-if-e1000-1)#
```

Notice that the prompt has changed. You're still in config mode, and now you're in a sub-config mode: interface config mode. The “if” in the prompt tells you that you're in interface config mode. The “e1000” tells you that you're configuring a gigabit Ethernet port. Finally, the prompt tells you which interface you're configuring (“1,” in this example).

Here's an example of how you might use this mode:

```
Switch(config-if-e1000-1)#speed-duplex 100-full
Switch(config-if-e1000-1)#interface e 5
Switch(config-if-e1000-5)#port-name MailServer
Switch(config-if-e1000-5)#
```

Notice that you've forced the speed and duplex of interface 1 to be 100 Mbps Full Duplex. The port is capable of doing gigabit full-duplex, but it's possible that you may be dealing with a device that may not negotiate properly. Interface 1 is now no longer autonegotiating. Any device plugged into this port must be set to 100 Mbps Full Duplex.

The next thing you need to do is to change to interface e 5. Now you're configuring a port-name to e 5. Call it “MailServer.” This would presume that you have a mail server plugged into e 5, but that's up to you.

You can also configure some settings to multiple interfaces at the same time:

```
Switch(config)#int e 3 to 48
Switch(config-mif-3-48)#no spanning-tree
Switch(config-mif-3-48)#
```

In this example, I've disabled spanning tree on interfaces 3 through 48. Spanning tree is still enabled on ports 1 and 2, but under the interface of every port from 3 to 48, the command **no spanning-tree** will be seen. Notice the change in prompt, too. It uses "mif" for "multiple interface config mode." This is followed by the range of ports you're configuring.

Ping

Very often, you will find it handy to make sure you can reach a certain IP address from a switch. For example, it's a good idea when you are copying data to or from a TFTP server to first make sure that you can reach that TFTP server. If you are running Base Layer-3 or Full Routing code, you may need to troubleshoot routing problems. This is where the **ping** command comes in handy.

In its simplest form, you simply issue the **ping** command and tell it which IP address you want to reach:

```
Switch#ping 10.10.10.1
Sending 1, 16-byte ICMP Echo to 10.10.10.1, timeout 5000 msec,
TTL 64
Type Control-c to abort
Reply from 10.10.10.1      : bytes=16 time<1ms TTL=64
Success rate is 100 percent (1/1), round-trip min/avg/max=0/0/
0 ms.
Switch#
```

If the destination does not reply, you'll see output like this:

```
Switch#ping 10.10.10.2
Sending 1, 16-byte ICMP Echo to 10.10.10.2, timeout 5000 msec,
TTL 64
Type Control-c to abort
Request timed out.
No reply from remote host.
Switch#
```

The **ping** command is very configurable. To see your options, use the **?**:

```
Switch#ping 10.10.10.1 ?
ping <ip addr> [source <ip addr>] [count <num>] [timeout
<msec>] [ttl <num>] [verify] [no-fragment] [quiet] [data <1-
to-4 byte hex#, e.g. abcdef00>] [numeric] [size <byte>] [brief
[max-print-per-sec <num 0-2047>]]
Switch#ping 10.10.10.1
```

Source <ip addr>. Particularly if you are running routing code, your switch may have many different IP addresses. This option gives you the ability to specify which of your switch's IP addresses you want to use as the source IP of the ping.

Count <num>. This allows you to specify how many pings you want to send. By default, it is set to 1, but it can be as many as 4,294,967,296.

Timeout <msec>. This specifies how long the switch will wait for the echo-reply packet of the ping. The default is 5,000 milliseconds (5 seconds). It can be set as high as 4,294,967,296 milliseconds.

TTL <num>. This allows you to specify the maximum number of router hops the ping packet will take to get to its destination. This prevents excessive looping. The default is 64. It can be set as high as 255.

Verify. This command verifies that the data in the echo-reply packet is the same data that was sent in the echo-request packet. In other words, it makes sure that this reply belongs to the appropriate request. By default, it does not verify.

No-fragment. This turns on the “don't fragment” bit in the IP header of the ping packet. By default, this bit is not turned on.

Quiet. This option deletes the first two lines of the output (showing all of the default parameters, etc.). The display will only show whether or not the ping received a reply.

Data <1-to-4 byte hex#, e.g. abcdef00>. This allows you to specify a pattern in the data section of the ping packet. By default, this is “abcd” (hex - 61626364). The defined pattern will repeat itself throughout the payload portion of the ping packet.

Size <byte>. This specifies the size of the ICMP data portion of the ping packet. This is just for the data payload, and does not include the size of the header. The default is 16 bytes. You can specify a number from 0 to 4,000.

Brief [max-print-per-sec <num 0-2047>] This will display an individual character representing the results of each ping.

- '!' means a reply was received
- '.' means a timeout; the switch did not receive a reply before the timeout value
- 'U' means that the switch received an ICMP “Destination Unreachable” message
- 'I' means that the user interrupted the ping (e.g., by pressing Ctrl-C)

The “max-print-per-sec” setting dictates how many characters will be printed in a given second.

You don't necessarily have to know the IP address of the host you're trying to ping. You can also specify a host by its name. To do this, you will need to configure a DNS "Resolver" for your switch. This is a DNS server that will look up and "resolve" the host's name to its IP address. You can configure this on your switch with the following command:

```
Switch#conf t
Switch(config)#ip dns server-address 10.10.10.1
```

You may also configure multiple DNS server addresses, in case the first does not respond.

Filters and Regular Expressions

Show commands can give you a lot of information. Sometimes, they can give you too much information. Luckily, Brocade has made it easier to sift through this information. Most show commands will allow you to follow the command with a pipe ("|") character. This feeds the output of the show command into a filter that we define. For those of you who are familiar with UNIX-based operating systems, think of this as being able to "pipe to grep" (though not as sophisticated).

Let's say we want to know how long the switch has been up, but we don't want to sift through all of the output normally associated with the **show version** command.

Using filters, we could do this:

```
Switch#show version | include uptime
The system uptime is 370 days 8 hours 25 minutes 16 seconds
Switch#
```

Notice the use of the word "include." Brocade gives us three commands we can use to filter the show command's output:

- **include** — Meaning "show me any line containing the following text"
- **exclude** — Meaning "don't show me any line containing the following text"
- **begin** — Meaning "as soon as you see the following text, show me the rest of the output"

You can only use one filter per command. In UNIX-based operating systems, for example, you can pipe from one command to another (e.g., "command | command2 | command3 | command4", etc.). With the Brocade CLI, you can only "pipe" once per command.

If the output is longer than your terminal screen, the Brocade CLI has got you covered. When a show command displays output that's more than a screen-size long (depending on your terminal settings), it will display this line at the end (or near the end) of the screen-size:

```
--More--, next page: Space, next line: Return key, quit:
Control-c
```

This allows you to continue at your own pace. You must press the Space bar to continue to the next page of data, or you could press the Return key to continue line by line (rather than page by page). If you don't want to see any more data, press Control-C. The Space bar and the Return key are the options that are displayed by the prompt.

Brocade, however, has made available some additional options. When you're at the "More" prompt, you can use these additional filters to further customize your display:

- The '+' (plus sign) acts as the "include" option; type the text of the lines you want to include after the '+', and press Return
- The '-' (minus sign, or hyphen) acts as the "exclude" option; type the text of the lines you want to exclude after the '-', and press Return
- The '/' (forward-slash) acts as the "begin" option; following the '/', type the text of the line you want to begin at, and press Return

To help you filter further, Brocade includes the use of *regular expressions*. If you're a UNIX programmer, you'll feel right at home here. For the rest of us, the idea of regular expressions is to give us a way to search within text. For example, a "." (period), in regular expressions, represents any single character. If I issue this command: **show running-config | include 1.3**, this displays all lines that contain "123," "113," "1A3," "1/3," in fact any character sandwiched between a "1" and a "3."

Here are some other characters that are useful:

Table 5. CLI Characters: Regular Expressions

Character	Purpose	Examples
.	Period. Any single character.	"1.3" matches "123," "1A3," "1/3"
^	Caret. The beginning of the line.	"^int" matches "interface e 3" but not "ip interface"
\$	Dollar sign. The end of the line.	"3/1\$" matches "interface e 3/1" but not "untagged e 3/1 to 3/2"
_	Underscore. Any characters used to separate a word (e.g., space, comma, parenthesis, etc.)	"list_" matches "access-list" but not "web-management list-menu"; "_ip_" matches "ip address" and "ip route" but not "router rip"
[]	Brackets. Used to specify a range, or a set of characters.	"[135]" matches "1," "3," or "5"; "[1-3]" matches "1," "2," or "3"
[^]	A caret as the first character inside the brackets is an inverse (e.g., match every character except these).	"[^135]" matches any character as long as it is not "1," "3," or "5"; "[^1-3]" matches any character as long as it is not "1," "2," or "3"
	Vertical bar or "pipe". A choice (e.g., boolean "OR")	"123 132" matches either "123" or "132"; "_ip_ _ipx_" matches the whole word "ip" or "ipx"
()	Parenthesis. Used to enclose a pattern.	Used in conjunction with "?" "*" and "+"
?	Question mark. Matches zero or one instance of the preceding character or pattern.	"123?" matches "12" and "123" but not "1233"; "(321)?" matches "321" and "321321" but not "321321321"
*	Asterisk. Matches zero or more instances of the preceding character or pattern.	"123*" matches "12," "123," "1233," "1233333," etc.; "(321)*" matches ""(nothing)," "321," "321321," "321321321," etc.; ".*" matches any number of any characters
+	Plus sign. Matches one or more instances of the preceding character or pattern.	"123+" matches "123," "1233," "123333," etc. but not "12"; "(321)+" matches "321," "321321," etc.

User Access Control

On most switches, it is a good idea to have at least some form of password protection. After all, you don't want just anyone making changes to your switches, right?

Username and Passwords

Simply put, a username and password are a combination that you can define to allow certain individuals to access your switch. A username can be anything. A password can be anything. But the combination of the two is what permits you to access the switch. Typically, you'll want your passwords to be cryptic (difficult to guess). A combination of letters, numbers, and special characters (periods, commas, etc.) is a good idea.

The Three “Enable” Modes

To start with, every switch should have an “enable” password. As we've talked about earlier in this chapter, this is the password that will get you into “Privileged EXEC mode.” This gives you full access to the switch to make changes, read tables, etc.

But what if you want to give someone Privileged EXEC mode access, but you want to limit what they can do? Brocade provides three “enable” passwords that you can define:

Super-User. This is the king. With this password, you have full control to execute any commands on the switch. Typically, this should be your most cryptic password, and should be used only by Administrators.

Port-Configuration. This allows a user access to most “show” commands, and it allows the user to make changes to individual interfaces. It does not allow the user to make global changes to the switch (i.e., Global config mode).

Read-Only. This allows a user to access most “show” commands, but it does not allow any changes to be made. This is typically used for other non-administrative personnel who may need to troubleshoot network throughput, but should not be allowed to make changes.

To configure these passwords, you can use the following commands:

```
Switch#conf t
Switch(config)#enable super-user-password s0m3th1ngh4rd
Switch(config)#enable port-config-password admins-0nly
Switch(config)#enable read-only-password EasyPassword
Switch(config)#
```


You access all three of these passwords the same way: type **enable** at the User level prompt. The password you enter will determine which of the three modes you enter into. Here's an example of a read-only login:

```
Switch>enable
Password:      <typed "EasyPassword">
Switch#config t
Invalid input -> config t
Type ? for a list
Switch#
```

The prompt shows that you are in Privileged EXEC mode, but you used the “read-only” password. This means that you can't change the configuration, but you still have access to most “show” commands.

It's important to remember that when you define passwords, they appear in plain text when you enter them. Afterwards, however, they are encrypted. For example, if we were to issue a **show run** command on the switch we just configured, we would see this:

```
Switch#show run
Current configuration:
!
ver 03.0.01cT3e1
!
!
!
!
!
!
!
!
enable super-user-password .....
enable port-config-password .....
enable read-only-password .....
hostname Switch
!
!
!
!
!
!
!
end
```

Notice that the passwords have been replaced with “.....” characters. Also notice that, no matter how long or short the password is, it will always be replaced (in the **show run** command) with five periods.

If you were to copy this config to a TFTP server, you would not see the five periods, but you would see the encrypted hash. It would look something like this:

```
enable super-user-password 8 $1$eqT62$3cKeWJKxb3ISFO
enable port-config-password 7 $4jl,sL%alsEN&
enable read-only-password 7 $1$eqjajidfiolKLjs.$
```

Why not show this information in the **show run** output? Well, as safe and sound as this appears, it is possible, though time-intensive, to reverse-engineer the hash. For example, if an attacker obtains access to this information, given enough time, he could use the hash to discover the actual passwords. Brocade makes it harder for attackers by hiding the hash and the passwords from anyone with read-only access to the switch.

You can actually show the passwords in plain text, by adding this command to the Global config:

```
Switch#conf t
Switch(config)#no service password-encryption
```

Your passwords will still show as five periods (“.....”) when you view either the startup or running config on the switch. This time, if you were to copy the config to a TFTP server, the passwords would be shown in plain text. There are very few scenarios in which turning off password-encryption is a good idea. The safest thing is to leave this setting at its default.

If you absolutely must show the password hash in the startup and running config, you may enter this command:

```
Switch#conf t
Switch(config)#enable password-display
```

Brocade recommends against this (and so does the author). Default settings will provide the greatest security.

Recovering Passwords

The feared day has come. You've forgotten your Super-User password, and you need to make an important change to the switch's config. You do have one last hope, but to perform this procedure, you must have serial access to the switch. This procedure *cannot* be performed remotely.

1. Reload the switch

The switch must be reloaded. If you cannot log in to issue the “reload” command, you must power off the switch, and power it back on.

2. Enter Boot-Monitor mode by pressing the “b” key when prompted. You must press the “b” key within two seconds of the switch receiving power.

You should see a prompt that looks like this:

```
BOOT MONITOR>
```

3. Temporarily bypass the system password check function. This is done by entering the following command:

```
BOOT MONITOR>no password
```

This command cannot be abbreviated. It must be entered exactly as shown.

4. Boot the switch.

```
BOOT MONITOR>boot system flash primary
```

Or alternatively:

```
BOOT MONITOR>boot
```

5. Enter Privileged EXEC mode.

```
Switch>enable
```

```
No password has been assigned yet...
```

```
Switch#
```

At this point, you can do any number of things:

- Configure a new super-user password
- Disable password-encryption and copy the config to a TFTP server
- Remove all passwords from the config and start over again

The options are completely up to you. Just remember, the override of the Super-User password is only temporary. The original config is still unchanged. If the switch were to be reloaded again, everything would be back the way it was.

Username

Now, what if you have 15 different administrators? Should you give all of them the “Super-User” password? That would mean that each time an administrator left the company, you’d have to change the Super-User password. Plus, it would increase the possibility that someone may discover the Super-User password (for example, one of your 15 administrators decides to write the password on a sticky note and put it on his/her monitor). Depending on your organization, it may be a better idea to give each administrator their own username and password and to set their privilege level accordingly.

To configure a username:

```
Switch#conf t
```

```
Switch(config)#username jimbob password SomethingEasy
```

```
Switch(config)#
```

Now, when user “jimbob” logs in with his password, he will automatically be in Privileged EXEC mode. The added bonus is that any time “jimbob” logs in, it will be noted in the switch’s log, and any changes that jimbob makes will be tied to his name. This provides accountability among your administrators. If they all used the same Super-User password, you’d never be sure which of them actually made a change.

By default, a newly-created user is given “Super-User” privilege. If you don't want a particular user to have Super-User privileges, you can specify the privilege in the command when you create the user:

```
Switch#conf t
Switch(config)#username beckysue privilege 5 password test
Switch(config)#
```

You can give someone the same privileges we listed in the different “enable” modes earlier in this section:

- 0 = Super-User
- 4 = Port-Configuration
- 5 = Read-Only

We created a username for “beckysue” and we gave her Read-Only privileges (“privilege 5”).

You can even create a username without a password:

```
Switch#conf t
Switch(config)#username anybody privilege 5 nopassword
Switch(config)#
```

Be certain you want to do this. Understand that this particular example will be able to log in to Privileged EXEC mode with read-only privileges without a password. You don't have to specify a privilege level when you're assigning a “nopassword” username, but it is a very, very good idea. It would be very risky to set up a username with no password that had Super-User privileges.

Telnet/SSH

SSH is a protocol that requires you to submit a username when initiating a connection. If you plan to use SSH to remotely connect to your switches (a good idea, by the way), you must have usernames defined. They don't necessarily need to be defined on the switch (more on that later in the next section), but they do have to be defined.

Telnet, on the other hand, does not require a username. In fact, by default, a switch will allow a user to Telnet into a switch and enter User mode without even a password! To configure a Telnet password:

```
Switch#conf t
Switch(config)#enable telnet password ThisIsMyTelnetPass
Switch(config)#
```

Now, even to enter User mode, a Telnet client will be prompted for a password (not a username, just a password).

Authentication, Authorization, and Accounting (AAA)

No, we're not planning a travel destination. “AAA” allows us to define how you want the switch to provide “Authentication” (user accounts), what someone is “Authorized” to do (privilege levels), and how to keep an “Accounting” of switch access (logging).

We mentioned earlier that you need to specify a username for SSH to function. The second part is that you need to tell the switch where to find the username database. If you've defined the usernames on the switch (as we specified earlier), you would issue a command similar to this:

```
Switch#conf t
Switch(config)#aaa authentication login default local
Switch(config)#
```

In this command, you're specifying an "authentication" method. It's specific for "login," which means Telnet and SSH. You could also use the key-word "web-server" to specify authentication for the Web UI.

The last field ("local," in our example) specifies where the switch should look for the defined usernames and passwords. The keyword "local" tells the switch that the usernames and passwords are defined in the switch's local config. Other possibilities are:

- line — Use the defined Telnet password
- enable — Use the defined enable passwords
- none — Turn off all authentication
- radius — Use a RADIUS server for usernames and passwords
- tacacs — Use a TACACS server for authentication
- tacacs+ — Use a TACACS+ server for authentication

Let's talk about those last three for a moment. RADIUS (Remote Authentication Dial In User Service) is an open standard for providing a centralized storage database for usernames and passwords (among other functions). This prevents you from having to define all of your usernames on all of your switches. With RADIUS, you can define the users once on the server, and simply configure all of your switches to authenticate using the RADIUS server. This requires two steps in the config:

```
Switch#conf t
Switch(config)#aaa authentication login default radius
Switch(config)#radius-server host 10.10.10.1
```

You can also configure a custom authentication port, should you have one defined on your RADIUS server (default is TCP and UDP 1645):

```
Switch(config)#radius-server host 10.10.10.1 auth-port 7932
```

TACACS (Terminal Access Controller Access-Control System) is a similar protocol to RADIUS. Here again, you define a centralized TACACS server to keep all of your usernames and passwords, and tell your switches to use it for their authentication:

```
Switch#conf t
Switch(config)#aaa authentication login default tacacs
Switch(config)#tacacs-server host 10.10.10.1
```

You can also configure a custom authentication port, should you have one defined on your TACACS server (default is UDP 49):

```
Switch(config)#tacacs-server host 10.10.10.1 auth-port 9723
```

TACACS+ is a proprietary enhancement to TACACS. It adds encryption and uses TCP. If you wish to use TACACS+ instead of TACACS, you specify this in the **aaa** command:

```
Switch(config)#aaa authentication login default tacacs+
```

A remote server makes it handy to centralize your usernames and passwords. But what if the server goes down? With the **aaa** command, you can define a series of “fail-back” methods (as many as you have). For example:

```
Switch(config)#aaa authen login default tacacs+ radius local
```

This command tells the switch:

1. Authenticate the incoming user to the TACACS+ server
2. If the TACACS+ server is not defined in the switch's config, or if it is down, authenticate the incoming user to the RADIUS server
3. If the RADIUS server is not defined in the switch's config, or if it is down, authenticate the incoming user to the locally defined usernames in the switch's config

If the process gets all the way to step 3, and the username isn't defined locally, the authentication will fail. You can also configure multiple TACACS and RADIUS servers for redundancy.

RADIUS, TACACS, and TACACS+ may be used to authenticate Telnet, SSH, and Web UI access. Other access methods, such as SNMP, have their own authentication mechanisms.

Simple Network Management Protocol (SNMP)

SNMP is an open standard defined in RFC 1157. It is beyond the scope of this book to go into any great detail about this protocol. There are many books written on the subject. For this section, we will discuss how to configure a switch to use SNMP.

SNMP version 1 uses a community string as an authentication mechanism. This string is defined on the switch. There are two types of community strings defined: read-only and read-write. The read-only string is only used for gathering information. This community string is like having a “read-only” enable password. The read-write string is used to change the configuration of the switch. This is similar to the “super-user” enable password.

SNMP clients use a **GET** command when they are gathering information. If they send the **GET** command to the switch accompanied by the defined read-only community string, the switch will give them any information they ask for (e.g., the running config, CPU statistics, interface statistics, etc.). Likewise, SNMP clients use a **SET** command to send a change to the switch. If they send the **SET** command to the switch accompanied by the defined read-write commu-

nity string, the switch will receive the new change and apply it (e.g., adding a static route, changing the switch's name, configuring an interface's speed, etc.).

SNMP is frowned on by some administrators. There are more than a few security concerns about what I've just described. The only thing keeping an attacker from seeing information you don't want them to see (or worse, changing information that you don't want them to change) is knowing the community string. And here's more bad news. In SNMP version 1, the community string is sent in clear text. Anyone generating a packet capture of SNMP traffic can very easily see both your read-only and read-write community strings.

There is a happy ending to this story. In RFC's 3411-3418, the IETF has defined SNMP version 3. This version provides authentication, privacy, and access control. SNMP version 3 is supported on all Brocade switches. But for the scope of this book, we're just going to focus on SNMP version 1.

Let's see how we configure the community strings:

```
Switch#conf t
Switch(config)#snmp-server community public ro
Switch(config)#snmp-server community private rw
```

The words “public” and “private” are arbitrary choices for the community string name. We could have just as easily have chosen:

```
Switch#conf t
Switch(config)#snmp-server community frank ro
Switch(config)#snmp-server community alice rw
```

The strings themselves can be anything, and it may be a good idea to make them fairly cryptic (especially, the read-write). In the above example, the read-only community string is “frank.” The read-write community string is “alice.”

When you look at your running config after configuring the strings, you'll notice that they are replaced by five periods, just like your passwords:

```
Switch#show run | include snmp
snmp-server community ..... ro
snmp-server community ..... rw
Switch#
```

Like the passwords, if you were to copy the config to a TFTP server, you would see an encrypted hash where the community string is supposed to be. You have the option not to encrypt the community string if you prefer. You indicate this when you create the string, using a “0” (zero) to specify no encryption, or a “1” (one) to specify encryption:

```
Switch(config)#snmp-server community 0 frank ro
Switch(config)#snmp-server community 1 alice rw
```

In this example, “frank” will not be encrypted, but “alice” will be. Again, the community strings are encrypted by default, so the “1” is assumed, if it is not specified.

SNMP also allows you to define other bits of information that are helpful when information gathering:

```
Switch(config)#snmp-server contact "Jeff Albertson"  
Switch(config)#snmp-server location "The switch closet"
```

These settings are purely for the convenience of the person who is configuring them. It can help organize your network and the data you receive.

You can also configure what is called an SNMP-trap receiver. SNMP can be used to gather information, but it can also be configured to receive information initiated from the switch. For the switch to be able to send SNMP information, it needs to know where to send it:

```
Switch(config)#snmp-server host 10.10.10.1
```

In this example, you configure the server at 10.10.10.1 to receive SNMP traps. A server running IronView can be used in this capacity, but there are other software packages that can perform this service as well. A favorite of mine is OpenNMS (<http://www.opennms.org/>).

Remote Monitoring (RMON)

You may also use a Remote Monitoring (RMON) client on your workstation to gather SNMP data. RMON (RFC 1757) uses a local data file called a Management Information Base (MIB) to match the cryptic numbers used by SNMP with more English-readable names. RMON software can help make it easier to find information that you want about a switch, such as CPU statistics, individual interface performance statistics, or whatever you may want to watch.

On the switch, RMON information is divided into four groups:

- Statistics (RMON Group 1)
- History (RMON Group 2)
- Alarms (RMON Group 3)
- Events (RMON Group 9)

The Statistics group will give you information per interface on multicast and broadcast packets, collisions, CRC errors, total packets sent, and more. Think about the information you see when you type **show interface**. Most of that same data is available through this RMON group. To see the information that the switch will give the RMON client, you can type **show rmon statistics**.

While the Statistics group gives you more real-time information, the History group preserves samples of the information in the Statistics group. By default, the History group will store two samplings: one at 30 second intervals, and one at 30 minute intervals. The time intervals are configurable. But remember, the switch doesn't have a hard drive. It's better to offload that data onto a monitoring station than to keep too much information on the switch itself. To see what is currently being stored in the History group, you can type **show rmon history**.

The Alarms group logs events of certain items you wish to monitor. For example, you could configure rmon to watch CPU utilization, and alert when the CPU passes a certain threshold. The alert will add a message to the Alarms group. To see the entries in the Alarms group, type **show rmon alarm**.

Finally, the Events group is divided into two sub-elements: the event control table and the event log table. The event control table has lists all of the action items pertaining to items being monitored. Typical actions will be to log an event, send an SNMP trap on an event, or to log and send an SNMP trap on an event. The event log table contains the event log. You will see the same data when you issue a **show log** command. To see the information in the Events group, type **show rmon event**.

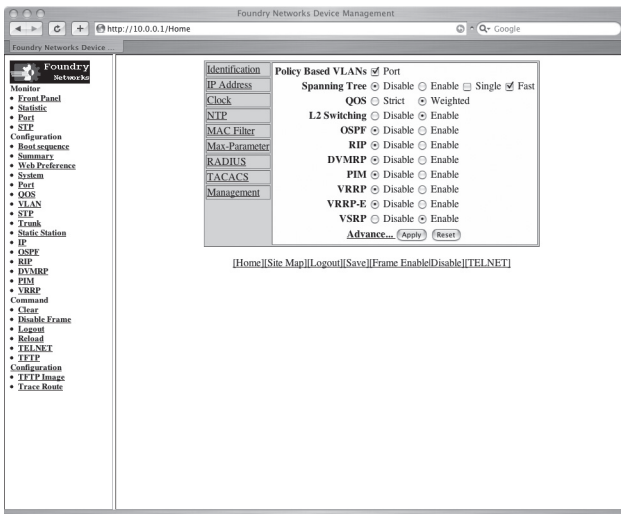
The Web UI

So, you don't really like the Command Line Interface so much? You feel more comfortable with menus and graphic images? Well, Brocade's got you covered. By default on every switch, a Web-based user interface (Web UI) is enabled.

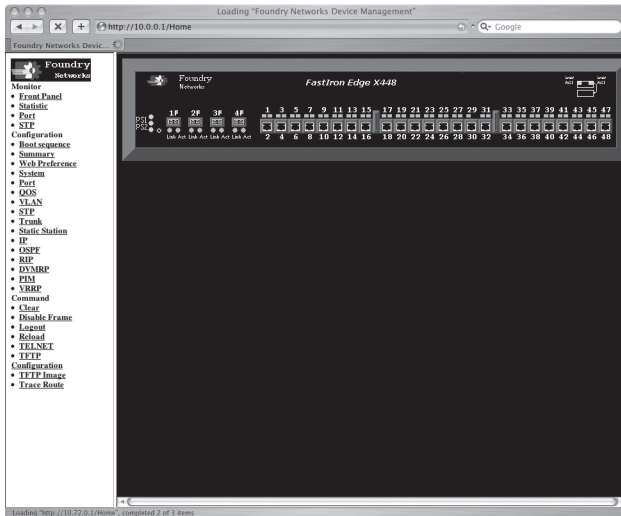
To access this Web UI, all you need is a Web browser, and the IP address of your switch. The first page that you see will prompt you to login. If you have not configured “aaa” for the “web-server” or any local usernames and passwords, you can access a read-only version of the Web UI using the username “get” and the password “public.” This is tied to SNMP. By default, a read-only community string of “public” is enabled on all switches.

It's important to remember that the Web UI is tied to SNMP. If you have defined a read-write community string, you may login to the Web UI using a username “set” and the defined read-write community string as the password.

When you login, you will see this:



You can see that you have many menus and links to permit you to configure your switch. There are also many helpful troubleshooting tools. For example, the “Front Panel” link represents what you would see if you were physically looking at your switch:



The Web UI has many configuration options. For example, to disable all authentication for the Web UI:

```
Switch(config)#web-management allow-no-password
```

This is insecure, and I wouldn't recommend it. However, there may be circumstances in which this may best fit your needs.

You can specify the use of HTTPS (SSL/TLS) for the Web UI, by using:

```
Switch(config)#web-management https
```

If you prefer, you can even change the TCP port that the Web UI listens on. By default, it would listen on TCP 80 (and TCP 443, if HTTPS was enabled). To change it:

```
Switch(config)#web-management tcp-port 1234
```

Now, whenever accessing the Web UI, you would need to specify TCP port 1234 in the URL.

Restricting or Disabling Access

All of this access is great, but too much of it can leave a Network Administrator feeling pretty vulnerable. Not to worry. You can actually restrict Telnet, Web UI, and even SNMP access to a single address with these commands:

```
Switch(config)#telnet client 10.10.10.1
Switch(config)#web client 10.10.10.1
Switch(config)#snmp-client 10.10.10.1
```

In these examples, the host whose IP address is 10.10.10.1 is the only device that may Telnet to the switch, access the switch's Web UI, or send SNMP GET or SET messages. If any other IP address attempts to Telnet to the switch, the TCP connection will be reset. Likewise, if any other IP address attempts to access the Web UI, the TCP connection will be reset. If any other IP address attempts to send SNMP messages, they will be dropped and ignored.

If you want to be really restrictive, you can even specify a specific MAC address in the client line as well:

```
Switch(config)#telnet client 10.10.10.1 1234.5678.9abc
```

This prevents someone from spoofing or assuming your IP address. This requires a match of both the IP address and the MAC address for access.

In this example, we've only configured a single "client" IP address. You can configure many IP addresses, depending on your need. However, you can't configure a range using the "client" commands.

If you need to restrict access to a range of IP addresses, you can use the **access-group** command. Create an Access Control List (see [Chapter 14](#) for more detail on this) that defines which IP addresses you want to have access. Let's say, for example, that you created an Access Control List named "Admins." You want to make sure that only the IP addresses defined in "Admins" have the right to Telnet, SSH, or access the Web UI of the switch.

```
Switch(config)#telnet access-group Admins
Switch(config)#ssh access-group Admins
Switch(config)#web access-group Admins
```

Notice that I've added SSH. SSH does not have a "client" command. It only uses "access-group." Likewise, notice that I left out SNMP. SNMP only has an "snmp-client" command. It has no "access-group." You typically want to heavily restrict the clients who are able to access SNMP (even when using version 3). Your average administrator is typically not going to use SNMP commands to interface with the switch. This is more commonly done through network management software, like IronView.

But what if you want to restrict SSH access to only one IP address? Access Control Lists can be as restrictive or as open as you define them to be. You would simply define an Access Control List that only permits one IP address, and apply that ACL to SSH with the **access-group** command.

But what if you want to disable all access? What if you don't want Telnet access to your switch at all? What if you don't want users to be able to access the Web UI? Take a look at these commands:

```
Switch(config)#no telnet server
Switch(config)#no web-management
Switch(config)#no snmp-server
```

The first command turns off Telnet. The switch will no longer listen on TCP 23 at all, regardless of the IP address. The second command turns off the Web UI. The third command disables SNMP functions.

Notice that I did not include a line for SSH. There isn't a way to disable SSH on the switch. To do this, you need to define an ACL that allows no traffic, and apply this ACL to SSH using the **access-group** command.

Summary

- A stackable switch is self-contained; it provides low-cost, but is not easily scalable
- A chassis switch provides greater scalability through a management blade, and any number of interface blades
- You can access a switch from its serial console or, if it is configured with an IP address, through the network, using Telnet, SSH, HTTP, or SSL
- At any stage in the configuration or command process, you can use the “?” to display contextual help
 - Using a question mark immediately following a word (or letters) will show you all the possible commands available that start with those letters
 - Using a question mark after a word (with a space between the word and the question mark) will show you what is needed next in the command
 - Using a question mark by itself will show you all of the available commands in your mode
- There are several modes in the CLI:
 - User mode allows very basic commands; does not change the configuration of the device
 - Privileged mode allows you to run all commands available to the device, including **configure terminal** which allows you to change the configuration of the device
 - Global config mode is the mode you enter when you enter **configure terminal** in Privileged mode; this is where you change settings that affect the entire switch

- Interface config mode can be entered by typing **interface e**, followed by the interface number you wish to configure; settings here affect only that interface
- There are many more sub-configuration modes to the Global config
- The running config is the configuration for the switch that is active; it is the configuration in RAM that affects the real-time settings of the switch
- The startup config is the stored configuration; it is stored in the switch's flash memory
- The switch runs an operating system that is stored in flash memory
- Flash memory is split into a primary and a secondary; the switch may be booted from either one
- “show” commands are used to display data about the switch, its configurations, its interfaces, and more
- “clear” commands are used to reset statistical counters, clear log entries, and more
- A “message of the day” (motd) can be configured; it displays a message to all CLI users when they log in
- The **ping** command uses ICMP echo request and echo reply packets to determine if a Layer 3 device is reachable
- You can use regular expressions and filters to trim needed data
- Users and passwords can be stored locally in the switch's config, or remotely using centralized servers (RADIUS, TACACS, TACACS+, etc.)
- Authentication Authorization and Accounting (AAA) is used to define how a switch is accessed
- SNMP and RMON can be used to gather statistics about (and even to configure) your switch remotely

Chapter Review Questions

1. Which command displays the running config?
 - a. show run
 - b. show config
 - c. show flash
 - d. show arp

2. Which RMON group gives you real-time information about total packets, collisions, and errors on each interface?
 - a. Events
 - b. Alarms
 - c. Statistics
 - d. History
3. Which command saves the running config to the switch's flash (to become the startup config)?
 - a. `copy tftp flash 10.1.1.1 config.txt`
 - b. `write terminal`
 - c. `copy flash flash primary`
 - d. `write memory`
4. Which command will show you a brief table of information about all interfaces?
 - a. `show interface`
 - b. `show interface brief`
 - c. `show logging`
 - d. `show run`
5. Which command will configure an SNMP read/write community string that is encrypted?
 - a. `snmp-server community 0 public ro`
 - b. `snmp-server community 1 private ro`
 - c. `snmp-server community 1 private rw`
 - d. `snmp-server community 0 private rw`
6. Which command displays the MAC address table?
 - a. `show arp`
 - b. `show mac`
 - c. `show interface`
 - d. `show config`

7. Which command will configure the switch to boot from secondary flash on reboot?
 - a. boot secondary flash
 - b. boot flash:2
 - c. boot system flash secondary
 - d. flash boot secondary
8. Which command will reset the interface counters?
 - a. clear statistics
 - b. clear interface statistics
 - c. clear counters
 - d. clear interface counters
9. Which command will copy a new software image from a TFTP server and save it to secondary flash?
 - a. copy tftp secondary 10.1.1.1 fes3601a.bin
 - b. copy secondary tftp 10.1.1.1 fes3601a.bin
 - c. copy flash tftp 10.1.1.1 fes3601a.bin secondary
 - d. copy tftp flash 10.1.1.1 fes3601a.bin secondary
10. Which command will show you the uptime of the switch, and how it was last booted?
 - a. show log
 - b. show flash
 - c. show version
 - d. show config

Answers to Review Questions

1. a. **show run** shows the running config. **show config** shows the startup config.
2. c. The History group shows statistics, but they're not real-time. The correct answer is Statistics.
3. d. **write memory** or **write mem** or even **wr me**. All of these copy the contents of the config in RAM (the running config) and write it flash to become the startup config.
4. b. **show interface brief** is the only command listed that will present information in a brief table. **show interface** will display the same information, but it won't be in table form, and it won't be brief.

5. c. This was a tricky one. A and B are read-only strings (not read/write). D is a read/write string, but it's unencrypted (uses a "0"). C is the only one that is both a read/write string and uses a "1" to indicate encryption of the string in the config.
6. b. **show mac** will display all of the MAC addresses the switch knows and which ports it learned them on. **show arp** will display the ARP table, matching IP addresses with MAC addresses and ports.
7. c. **boot system flash secondary** will tell the switch, when it is rebooted, to boot using the software stored in secondary flash. All of the other answers here were made up.
8. a. **clear statistics** will reset the counters.
9. d. You need to specify that you're copying from tftp to flash first. Then, you can specify the TFTP server and file, and finally, you can specify which flash you wish to copy to.
10. c. This is a tricky one. You would normally use **show version** to display the version of the currently running software. It also has a very handy function in which it displays how long the router has been running since the last reboot, as well as how the switch was last rebooted.

Link Aggregation (Trunks)

In Section 1, I talked about a “single point of failure.” This is when one component failure causes the whole (or a large portion) of the infrastructure to fail. Let's start with an example:



If this single link between the two switches were to fail, communication between the two switches would cease.



One way we can solve this problem is by defining an *aggregate link* or *trunk*. A trunk is two or more links combined together to become one virtual link. Going back to our example:



The circle around the two links designates a trunk defined on both switches. With this in place, if one link goes down, the other will continue to pass traffic. Even better, while both links are active, traffic will pass through both links. This provides greater bandwidth between the two switches.

A trunk link load balances the traffic that goes through it. The method that it uses to load balance the traffic depends on the type of traffic, and what type of trunk it is.

In this chapter, we will talk about two different types of trunks: *static* and *dynamic*.

Static Trunks

Static trunks are trunks that are configured once, and then, essentially, left alone. They don't change. For example, let's say you've configured a two-port trunk from one switch to another. Later on, you decide that you actually need to make it a four-port trunk. To create this four port trunk, you must first remove the two-port trunk, and then rebuild the trunk as a four-port trunk. There's no way to simply "expand" the existing trunk.

Static trunks come in two varieties: *switch* and *server*.

Switch Trunks

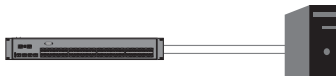
Switch trunks are designed for communication between two switches. All of the examples you've seen so far are switch trunks. These are designed to provide redundancy and greater throughput between switches.

On JetCore switches, the trunk load balances traffic based on the destination address. If the traffic is pure Layer 2 traffic, it is load balanced by the destination MAC address of the traffic. If the traffic is Layer 3 (or higher) traffic, it will be load balanced by the destination Layer 3 address (e.g., IP address, IPX address, etc.). The initial choice (for the first packet or frame) is decided round-robin (e.g., first link 1, then link 2, then link 1, etc.). Once a destination address has been tied to a particular link in the trunk, the switch will keep this information in a table, and future traffic will travel down the same link.

Newer switches provide the same load balancing options as server trunks.

Server Trunks

Server trunks are designed for communication between a single host (or server) and a switch. For this, you'll need to have a server with multiple NICs. The server will also need to have a link aggregation driver (this driver will vary depending on the manufacturer of the NICs and the operating system of the server). These are designed to provide link redundancy and increased throughput for the server.



With server trunks, the traffic is load balanced by source and destination addresses. If the traffic is pure Layer 2 traffic, it is load balanced by the source and destination MAC address of the frame. If the traffic is pure Layer 3 traffic, it is load balanced by the source and destination Layer 3 address (e.g., IP address, IPX address, etc.). If the traffic is TCP/IP, it is load balanced using the source and destination IP address and the source and destination TCP/UDP port number. This adds a lot more granularity. Here, it is possible that a single source going to the same destination engaging more than one conversation will likely use different trunk links. This maximizes the bandwidth of the trunk.

Brocade has also added an extra load balancing option that is unique to server trunk groups: “per packet” load balancing. The “per packet” load balancing method fixes an age-old problem with link aggregation. Load balancing based on source and destination addresses is okay if your traffic is diverse. If you want to maximize traffic between *one* source and *one* destination copying a large amount of data, however, you're out of luck. The switch will make its decision as to which trunk link the traffic will go across, and that's the only link that will be utilized. As you can see, when configuring a four- or an eight-port trunk link, that's quite a waste of bandwidth. This problem is commonly found when dealing with server backups.



The answer? “Per packet” load balancing. This uses pure round-robin. In other words, one packet is sent out trunk link 1, the next is sent out trunk link 2, and so forth. This is the best load balancing method to get the absolute maximum bandwidth out of your trunk.



Dynamic Link Aggregation (IEEE 802.3ad)

This is a dynamic trunk. A dynamic trunk can have ports added to it (and subtracted from it) at any time without having to redefine the trunk. Using this trunk method provides a much greater amount of flexibility, and it seems clear that this will become the trunking standard.

IEEE 802.3ad defines a protocol called Link Aggregation Control Protocol (LACP). This protocol sends frames out its defined ports. These frames are called Link Aggregation Control Protocol Data Units (LACPDU). Whether you're creating a dynamic trunk from a switch to a server, or from a switch to another switch, these frames are shared to provide information regarding the current state of the trunk.

LACP provides better redundancy. In a static trunk, the switch knows that if a link goes down, it will no longer pass traffic along that link. But what if a port's ASIC (hardware) fails? The physical connection may still show as being up, but the port may be unable to pass traffic. In a static trunk, traffic will still be sent on that “sick” link, and frames will be lost. With LACP, it would detect the failed link, because the port would no longer be sending LACPDUs. Yes, it may still show Layer 1 (Physical) connectivity, but if it's not sending LACPDU frames, the link is considered down. No traffic will be sent on that link.

A port defined as LACP can be either *active* or *passive*. If a port is active, it can send and receive LACPDUs with the other side of the link. This enables the switch to negotiate and configure the trunk. If a port is in passive mode, the port may receive and respond to LACPDUs, but it may not initiate the negotiation process.

Configuring Trunks

There are a few rules to be aware of when configuring trunks.

Rule #1. Generally speaking, if you are dealing with a stackable switch, you can only configure a two- or four-port trunk. On chassis switches, you can configure a two-, four-, or eight-port trunk. This rule is not hard and fast. Consult the Brocade web site (<http://www.brocade.com/>) for your specific model.

Rule #2. Your trunk links must be contiguous. You cannot create a trunk containing, say, ethernet 1, ethernet 6, ethernet 11, and ethernet 23. Given this example, if you wanted to configure a four-port trunk starting with port 1, you must configure the trunk to include ethernet 1, ethernet 2, ethernet 3, and ethernet 4.

Rule #3. For a two-port trunk, you must start your trunk on an odd-numbered port. For example, you can trunk ethernet 1 and ethernet 2, but you cannot trunk ethernet 2 and ethernet 3.

Rule #4. For a four-port trunk, you must start your trunk on ethernet 1, 5, 9, 13, 17, etc. You cannot start your trunk on, say, ethernet 3. It's an odd number, but it's not within the four-port divisor (starting with port 1). If you're still confused by this rule, a good rule to remember is to take the last port of your proposed four-port trunk—if you can divide that port number by four evenly, it is a valid trunk.

Rule #5. Similar to Rule #4, except now we're talking about eight-port trunks. These trunks must start on ethernet 1, 9, 17, etc. A similar rule applies (as in Rule #4). If you can divide the last port number in your proposed trunk by eight evenly, it is a valid trunk.

Rule #6. The ports in the trunk must have the same tag type (tagged/untagged), they must be in the same VLAN, they must be the same speed and duplex, and they must be the same Quality of Service (QoS) priority.

Rule #7. For chassis switches, you can create a four- or eight-port trunk that spans neighboring blades.

For this to work, the ports on either blade have to align. For example, e 1/1 to 1/4 and e 2/1 to 2/4 may be trunked in an eight-port trunk. You cannot trunk e 1/1 to 1/4 and e 2/5 to 2/8 in a single eight-port trunk. Notice also that I mentioned neighboring blades. You cannot span the trunk from slot 1 to, say, slot 4. It must be slot 1 to slot 2. This brings up another good point, you must start the multi-blade trunk on an odd-numbered slot. You can trunk from slot 1 to slot 2, but you cannot trunk from slot 2 to slot 3.

Switch Trunks

You define static trunks in the Global config:

```
Switch#conf t
Switch(config)#trunk switch e 1 to 4
Trunk will be created in next trunk deploy.
Switch(config)#
```

The CLI is reminding you that this trunk will not be active until you deploy the trunk. So, how do you deploy the trunk?

```
Switch#trunk deploy
```

Server Trunks

Similar to static switch trunks, static server trunks are also configured in the Global config:

```
Switch#conf t
Switch(config)#trunk server e 5 to 8
Trunk will be created in next trunk deploy.
Switch(config)#
```

Again, to actually activate the trunk:

```
Switch#trunk deploy
```

If you want to configure “per packet” load-balancing, you would configure this on the interface. You would configure the first interface in the trunk like this:

```
Switch#conf t
Switch(config)#int e 5
Switch(config-if-e1000-5)#serv-trunk-per-pkt-lb
```

IEEE 802.3ad (LACP) Trunks

These trunks are not configured at the Global config level. They are configured at the interface level:

```
Switch#conf t
Switch(config)#int e 9
Switch(config-if-e1000-9)#link-aggregate active
Switch(config-if-e1000-9)#int e 10
Switch(config-if-e1000-10)#link-aggregate active
Switch(config-if-e1000-10)#
```

Or, a more efficient way to do it would be:

```
Switch#conf t
Switch(config)#int e 9 to 10
Switch(config-mif-9-10)#link-aggregate active
```

If you would like to configure your LACP interface to be passive:

```
Switch#conf t
Switch(config)#int e 9
Switch(config-if-e1000-9)#link-aggregate passive
```

LACP gives you several other parameters to configure, if you like. All of these are configured on the individual interface:

System Priority. This sets the switch's priority in relation to the other device you're connecting the trunk to. This can be a number from 0-65,535 (16-bit). By default, it is 1. The lower the number, the higher the priority. When you're working with switches from other manufacturers, sometimes it helps to set the Brocade switch's priority to a higher number (lower priority).

```
Switch(config-if-e1000-9)#link-aggregate configure system-
priority 10
```

Port Priority. This is also a number from 0-65,535. The default value is 1. The lower the number, the higher the priority. This number is set to define which port will become the default active port in the trunk.

```
Switch(config-if-e1000-9)#link-aggregate configure port-
priority 20
```

Link Type. This is where you would define if your LACP trunk is to a server or a switch. By default, it's a switch.

```
Switch(config-if-e1000-9)#link-aggregate configure type
server
```

Key. This is a number from 0-65,535. This helps you define a unique trunk on a switch that may have several LACP ports. If you're configuring a custom key, Brocade recommends that you choose a number greater than 10,000. If no key is defined, LACP creates dynamic keys. Choosing a number greater than 10,000 avoids potential conflicts with those keys.

```
Switch(config-if-e1000-9)#link-aggregate configure key 12345
```

show Commands

There are two commands to become familiar with here. For static or LACP trunks, you can use:

```
Switch#show trunk
```

Configured trunks:

```
Trunk ID: 1
Hw Trunk ID: 1
Ports_Configured: 4
Primary Port Monitored: Jointly
Ports          1          2          3          4
Port Names     none      none      none      none
Port_Status    enable   enable   enable   enable
Monitor        off       off       off       off
Rx Mirr Port   N/A      N/A      N/A      N/A
Tx Mirr Port   N/A      N/A      N/A      N/A
Monitor Dir    N/A      N/A      N/A      N/A
```

Operational trunks:

```

Trunk ID: 1
Hw Trunk ID: 1
Duplex: Full
Speed: 1G
Tag: Yes
Priority: level0
Active Ports: 4
Ports          1          2          3          4
Link_Status    active    active    active    active

```

This information is very straightforward. Here, you can check to see how many ports you've configured in the trunk, the speed, duplex, link status, etc.

For LACP, you can also use:

```

Switch#show link-aggregation
System ID: 000c.abcd.1200
Long timeout: 120, default: 120
Short timeout: 3, default: 3
Port  [Sys P] [Port P] [ Key ]
[Act] [Tio] [Agg] [Syn] [Col] [Dis] [Def] [Exp] [Ope]
9 1 1 482 Yes S Agg Syn Col Dis Def No Ina
10 1 1 482 Yes S Agg Syn No No Def Exp Ina

```

System ID. This is the base MAC address of the switch

Port. The port number that has been configured for LACP

Sys P. This is the System Priority

Port P. This is the Port Priority

Key. This is the LACP key

Act. This interface is either “Yes” (active) or “No” (passive)

Tio. This is the Time Out value for the port; this value is either “L” (Long), meaning a trunk has already been negotiated (and longer timeouts are being used), or “S” (Short), meaning that negotiation has just started

Agg. This is either “Agg” (link aggregation is enabled) or “No” (link aggregation is disabled)

Syn. Synchronization; this can be either “Syn” (synchronized with the remote port) or “No” (not synchronized with the remote port)

Col. Collection; this identifies whether or not the port is ready to send traffic; if it is “Col,” it is ready to send traffic; if it is “No,” it is not

Dis. Distribution; this identifies whether or not the port is ready to receive traffic; if it is “Dis,” it is ready to receive traffic; if it is “No,” it is not

Def. Default; if the value is “Def,” this means no parameters were received from the remote device, so the interface will use default link aggregation settings; if the value is “No,” this means that parameters were negotiated with the remote device and those parameters will be used

Exp. Expired; this indicates whether or not the negotiated link aggregation settings have expired; these settings expire when the interface does not receive an LACPDU before the timer runs out; this can be either “Exp” (meaning that the settings have expired, and the interface has changed to default), or “No” (meaning that the settings have not expired)

Ope. Operational; this can be either “Ope” (everything is normal), “Ina” (Inactive; the link is down or has stopped sending LACPDUs), or “Blo” (Blocked; the remote port is either not configured with LACP or cannot join a trunk group; be sure that both interfaces are configured with the same key, if a custom key is used)

Summary

- A trunk can bind two or more links together to provide redundancy and greater bandwidth
- A static trunk is configured once; if changes are necessary, the trunk must be removed and recreated
- A dynamic trunk allows the trunk to be resized (ports added or taken away) without removing and recreating
- Dynamic (LACP) trunks use LACPDUs (Layer 2 frames) to negotiate the trunk
- Layer 2 traffic is load balanced using the Destination MAC address
- Layer 3 traffic is load balanced using the Source IP address, Destination IP address, Source MAC address, Destination MAC address and protocol (IP, TCP, UDP, etc.)
- A switch trunk is a aggregated link between two switches
- A server trunk is a aggregated link from a switch to a server or host

Chapter Review Questions

1. On a stackable switch, what's the largest trunk you can create?
 - a. Two-port
 - b. Four-port
 - c. Eight-port
 - d. Six-port
2. I'm using a stackable switch, and I'd like to create a four-port trunk. Which port can I start with (without generating an error)?
 - a. 3
 - b. 6
 - c. 9
 - d. 2

3. Which type of trunking allows you to easily add or subtract ports to the trunk without removing and recreating the trunk?
 - a. Static Trunk
 - b. Switch Trunk
 - c. Server Trunk
 - d. Dynamic (LACP) Trunk
4. What are the frames that LACP uses to negotiate the trunk?
 - a. LACPDU
 - b. BPDU
 - c. LACPBdu
 - d. BPDU-LACP
5. When load balancing traffic that is strictly Layer 2 only, how will the switch decide which trunk interface the frame uses?
 - a. Source IP, Destination IP, Source MAC, Destination MAC, Protocol
 - b. Frame size
 - c. Destination MAC
 - d. Destination IP
6. When you create or remove a static trunk, what last command do you use to actually implement the change?
 - a. show trunk
 - b. trunk deploy
 - c. deploy trunk
 - d. write memory
7. Which type of server trunk load balancing will send frames round-robin across all of the ports in the trunk?
 - a. Static Trunk
 - b. Per-packet
 - c. Destination MAC
 - d. Destination IP
8. How can you distinguish separate LACP trunks on the same switch?
 - a. Static Trunks
 - b. Per-packet
 - c. Keys
 - d. MD5 hash

9. I have two chassis switches, each with multiple blades. May I set up a trunk between these two switches and have the trunk ports span multiple blades?
 - a. No, ports must be contiguous.
 - b. No, you can only do that with a Server Trunk.
 - c. Yes, as long as each blade houses two or four ports of the trunk.
 - d. Yes, because of LACP.
10. When a trunk is load balancing Layer 3 (IP) traffic, what criteria does it use to decide which trunk interface to send the traffic?
 - a. Source IP, Destination IP, Source MAC, Destination MAC, Protocol
 - b. Packet size
 - c. Destination MAC
 - d. Destination IP

Answers to Review Questions

1. b. For 8-port (and potentially larger), you need a chassis.
2. c. 9 is correct. Remember to divide the last interface number (12, in this case) by four. If it's a round number (no remainder or fraction), it's correct.
3. d. Dynamic trunks allow you to add or subtract interfaces from the trunk without removing the entire trunk.
4. a. LACPDU's are the frames that are passed between LACP links. BPDUs will be talked about in Chapter 8. The other two don't exist.
5. c. Destination MAC. If it's strictly Layer 2 traffic, it's not going to have any IP information. It will just have a Source and Destination MAC address. It load balances on the Destination MAC address.
6. b. trunk deploy. If you forget, the switch will remind you immediately following a command that requires it.
7. b. The server per-packet load balancing sends frames round-robin across all links.
8. c. Keys can be configured to distinguish one LACP trunk from another.
9. c. You can span multiple blades, but you've got to have at least a four-port trunk (two-ports per blade), and the blades must be sequential, and you must start on an odd numbered slot.
10. a. With Layer 3 traffic, it uses all of these criteria to decide which trunk interface to direct traffic through.

Virtual Local Area Network (VLAN)

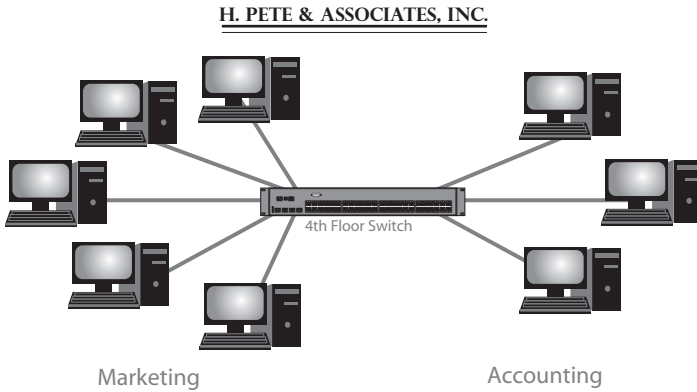
No discussion of Layer 2 switching would be complete without mentioning VLANs. VLANs have become the foundational blocks of network infrastructures all over the world.

What is a VLAN?

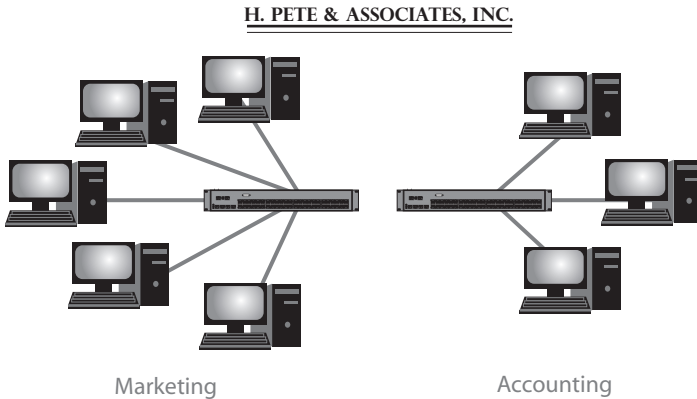
The acronym LAN stands for Local Area Network. This is the kind of network most people are familiar with. If you're using Ethernet, you are most likely doing so within a LAN. The key word here is *local*. This would be your network at home, the network that you plug your workstation into at the office, or it can even be the network between a group of buildings geographically close to each other. The main principle of a LAN is high-speed bandwidth and short physical distances. In contrast, a WAN (Wide Area Network) is geographically distant (networks between cities, countries, and even continents), and involves, typically, slower bandwidth (often involving leased long-distance circuits from telecommunications companies). The Internet is the largest and most well-known WAN.

Okay, so what is a VLAN? The V stands for *Virtual*. This is a Virtual Local Area Network. "Great," I hear you cry, "so what does that mean?!" Think of a VLAN as the ability to subdivide a LAN. It can break up a LAN into, essentially, smaller LANs. These "smaller LANs" are virtual or logical LANs within a larger, physical LAN.

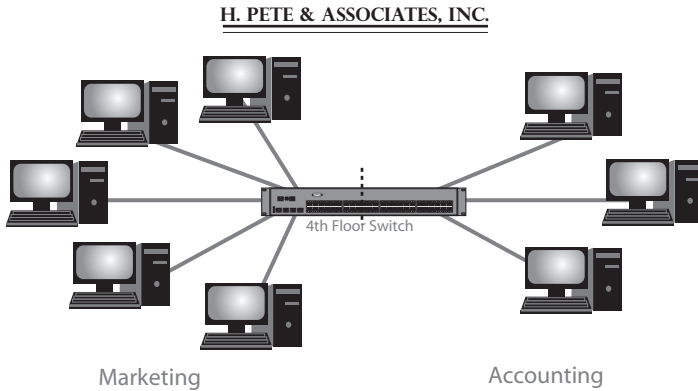
Let's take a look at an example. Say, you have a LAN for the fictional company H. Pete & Associates, Inc. On the fourth floor of their building, we have the Marketing department and the Accounting department. All of the workstations and servers in these two departments connect to the same switch on the fourth floor.



The Accounting department does most of its communicating within itself, and rarely has a need to communicate to other departments. It also contains the most sensitive information about the company. There's growing concern within the company that these two departments should be more separated. One way to do it is to purchase another switch and move the workstations of, say, the Marketing department to the new switch.

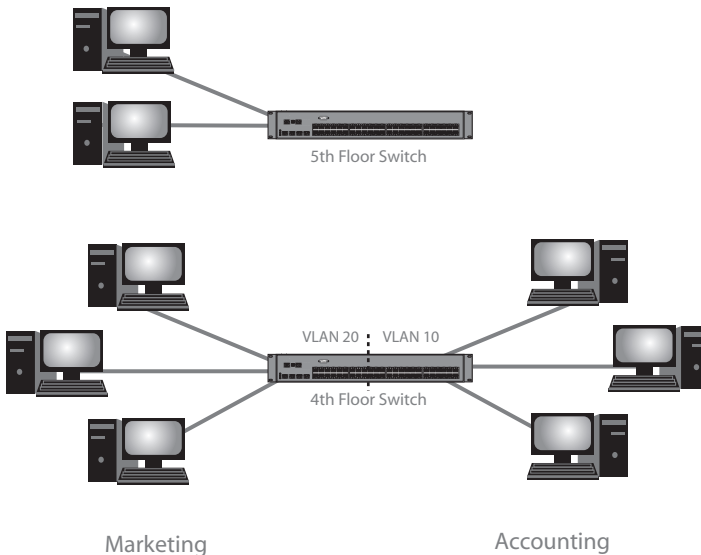


There's a less expensive way. You could, instead, create two VLANs on the switch. One VLAN for Accounting, and another one for Marketing.

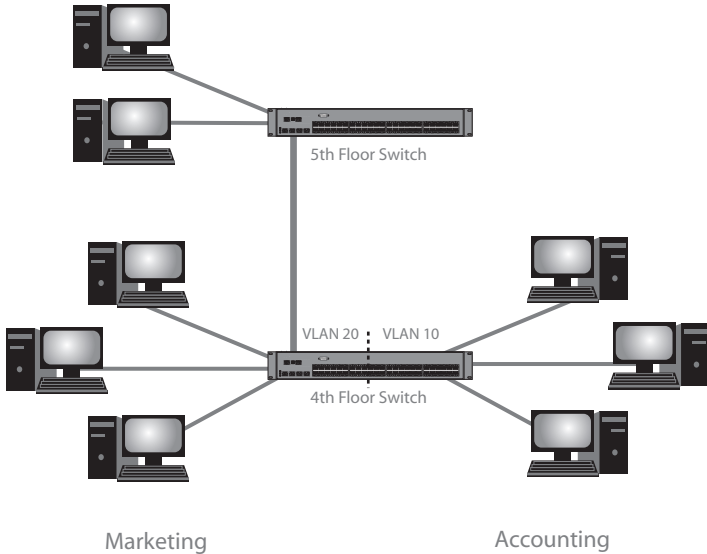


Notice that all the workstations are still plugged into the same single switch. But now, they are in two different *broadcast domains*. In other words, when a workstation in Marketing sends a broadcast (ARP, DHCP, etc.), Accounting will never see it, even if the two departments are in the same subnet. The VLAN behaves exactly like the solution in which you bought another switch. It's as if the one switch has become two smaller switches (that are not connected to each other).

Now, let's say that the Marketing department increased in size, and now they've moved some of their personnel up to the fifth floor. The workstations on the fifth floor plug into a different switch, which is plugged into the switch on the fourth floor.



We want the fifth floor Marketing workstations to be able to communicate in the same broadcast domain as the Marketing workstations on the fourth floor. No problem. VLANs can (and commonly do) extend to multiple switches.



Now the entire Marketing department on both floors is able to communicate amongst themselves, but their communication is still completely segregated from the Accounting department.

802.1q Tagging

IEEE 802.1q defines the VLAN concept. When you create a VLAN on a switch, you need to be able to tell the switch which of its ports belong to that VLAN. There are two types of membership in a VLAN: *tagged* and *untagged*.

If a switch port is an untagged member of a VLAN, the switch simply keeps track that this port is a member of this broadcast domain (VLAN) and only this broadcast domain. If a port is an untagged member of a VLAN, it can only be a member of that one VLAN.

But what if you want a port to be a member of two (or more) VLANs? This requires the switch port to be tagged. By “tagged,” it means that the switch actually adds an extra four bytes to the Ethernet frame. These additional four bytes are called the *802.1q header*. These four bytes are squeezed in between the Source Address field, and the Length/Type Field.

Preamble 8 bytes	Destination Address 6 bytes	Source Address 6 bytes	TPID 2 bytes	Priority 3 bits	CFI 1 bit	VLAN ID 12 bits	Length Type 2 bytes	Data	FCS 4 bytes
---------------------	-----------------------------------	------------------------------	-----------------	--------------------	--------------	-----------------------	---------------------------	------	----------------

The 802.1q header has four parts:

Tag Protocol Identifier (TPID). This is a 2-byte (16-bit) field, and it equals 0x8100. In Chapter 1, we talked about the Length/Type Field, and how it can be used to identify the upper-layer protocol that generated the frame (e.g., 0x0800 means the frame was generated by IP). The value “0x8100” tells any Ethernet device receiving it that it is receiving an IEEE 802.1q tagged frame. It will then know to expect and process the rest of the 802.1q header before looking for the actual Length/Type Field.

Priority (802.1p). This is a 3-bit field. It can be a number from 0 to 7. It defines the Quality of Service priority for the tagged frame. We'll talk more about this in Chapter 13.

Canonical Format Indicator (CFI). This is one bit. Given that, its value can be either 0 or 1. This field was largely defined to aid in compatibility between Ethernet and Token Ring networks. If you're using Ethernet, this value will be 0.

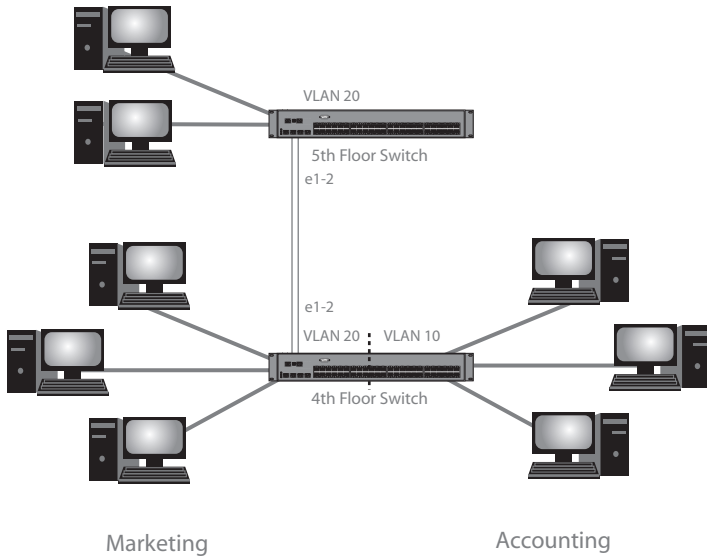
VLAN Identifier (VID). This is a 12-bit field. This means that it can be any number from 0 to 4,095. When you create a VLAN, you must give it a number. It can be any number between 1 and 4,095 (zero has been reserved to mean that the frame is not a member of any VLAN). A tagged port can be a member of more than one VLAN. This field identifies which VLAN this frame belongs to.

For tagging to work, *both* sides of the link must be tagged. This is most commonly used on links (or trunks) from switch to switch. If we are connecting Switch A to Switch B, and we want the link between the two switches to be members of more than one VLAN, the link's port on Switch A and the link's port on Switch B must be tagged.



Configuring VLANs

Well, this is all fascinating, but how or when would I ever use this? Let's go back to our H. Pete & Associates, Inc. example. We left off with the Marketing department on the fourth and fifth floor, and the Accounting department on the fourth floor. Let's say that we're dealing with 48-port switches, and let's say the Accounting department needs ports 3 to 24 (on the fourth floor switch). The Marketing department needs ports 25 to 48 on the fourth floor switch, and ports 3 to 48 on the fifth floor switch. Finally, let's say that the fourth floor switch has a two-port trunk configured using ports 1 and 2. This trunk is connected to the fifth floor switch that also has a trunk configured using ports 1 and 2.



How do we get this far? Well, first let's configure the trunks. We'll use LACP.

```
4thFloorSwitch#conf t
4thFloorSwitch(config)#int e 1
4thFloorSwitch(config-if-e1000-1)#link-aggregation active
4thFloorSwitch(config)#int e 2
4thFloorSwitch(config-if-e1000-2)#link-aggregation active
```

The same thing needs to be configured on the fifth floor switch:

```
5thFloorSwitch#conf t
5thFloorSwitch(config)#int e 1
5thFloorSwitch(config-if-e1000-1)#link-aggregation active
5thFloorSwitch(config)#int e 2
5thFloorSwitch(config-if-e1000-2)#link-aggregation active
```

Now we need to create the VLANs. Let's start with the fourth floor. Here, we need ports 3 to 24 to be untagged members of the Accounting VLAN. We need to give this VLAN a number, let's use 10:

```
4thFloorSwitch#conf t
4thFloorSwitch(config)#vlan 10 name Accounting
4thFloorSwitch(config-vlan-10)#untagged e 3 to 24
added untagged port ethe 3 to 24 to port-vlan 10
4thFloorSwitch(config-vlan-10)#
```

Notice that we gave the VLAN a name. This is completely optional. It's a feature you can use for your convenience.

Now, we need to set up the Marketing VLAN. Let's use VLAN 20 for that. Remember that this needs ports 25 to 48 to be untagged members:

```
4thFloorSwitch#conf t
4thFloorSwitch(config)#vlan 20 name Marketing
4thFloorSwitch(config-vlan-20)#untagged e 25 to 48
added untagged port ethe 25 to 48 to port-vlan 20
4thFloorSwitch(config-vlan-20)#
```

Now, before the expansion to the fifth floor, we would have been done. But we need to stretch the Marketing VLAN (VLAN 20) to the fifth floor. Now, we can do this in one of two ways. One way is that we could make ports 1 and 2 of the fourth floor switch untagged members of VLAN 20. Then, we'd need to make all of the ports on the fifth floor switch untagged members of VLAN 20, too. Now, this will work, but it reveals bad planning. Let's do it anyway, and I'll show you why:

```
4thFloorSwitch#conf t
4thFloorSwitch(config)#vlan 20
4thFloorSwitch(config-vlan-20)#untagged e 1 to 2
added untagged port ethe 1 to 2 to port-vlan 20
4thFloorSwitch(config-vlan-20)#
```

Notice that I don't have to refer to the VLAN's name every time I want to get into VLAN config mode. I only have to refer to it once, when I create it. In fact, if you want to give it a name after you've created it, you just have to go into VLAN config mode, and give it a name. This is how you would change its name as well. Now let's configure the fifth floor switch:

```
5thFloorSwitch#conf t
5thFloorSwitch(config)#vlan 20 name Marketing
5thFloorSwitch(config-vlan-20)#untagged e 1 to 48
added untagged port ethe 1 to 48 to port-vlan 20
5thFloorSwitch(config-vlan-20)#
```

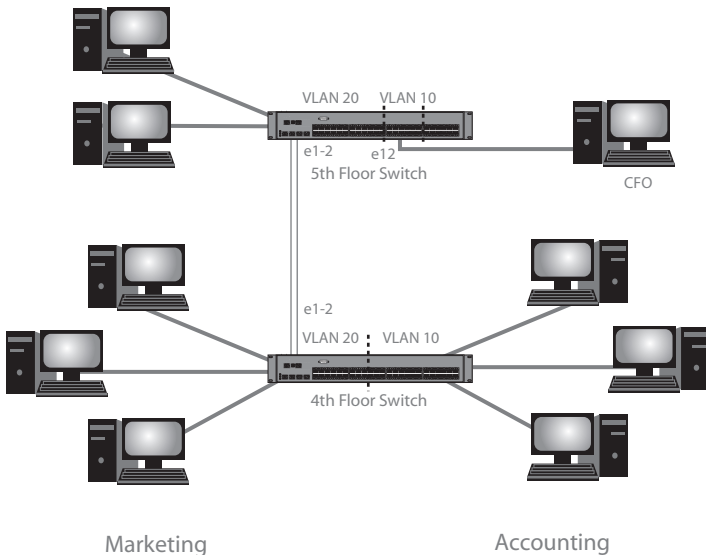
Okay, everything works now. Network communication is exactly the way we want it. But what about the future?

Let's say the CFO of H. Pete & Associates, Inc. has been looking at the office space on the fifth floor. He feels he needs to physically separate himself from the fourth floor Accounting department, and besides, the office he has his eye on is much bigger and has a better view. Now, his workstation still needs to be connected into the Accounting VLAN, and he'll be plugging into the fifth floor switch.

Let's also say the CFO's office uses port 12 of the fifth floor switch. We need to remove this port from the Marketing VLAN (an untagged port can only be a member of one VLAN), create the Accounting VLAN, and make port 12 an untagged member of it:

```
5thFloorSwitch#conf t
5thFloorSwitch(config)#vlan 20
5thFloorSwitch(config-vlan-20)#no untagged e 12
5thFloorSwitch(config-vlan-20)#vlan 10 name Accounting
5thFloorSwitch(config-vlan-10)#untagged e 12
added untagged port ethe 12 to port-vlan 10
5thFloorSwitch(config-vlan-10)#
```

Here's the updated picture:



Can the CFO's workstation speak to the rest of the Accounting department? No. There's no link for VLAN 10 between the fourth floor switch and the fifth floor switch. What about the two-port LACP trunk between them? That's not a member of VLAN 10. It is an untagged member of VLAN 20. What can we do to fix this? Well, we can physically run another cable between the fourth and fifth floor switches, but the building owners won't allow it. We could break the two-port trunk. Then we could make port 1 an untagged member of VLAN 10 and port 2 an *untagged* member of VLAN 20. This isn't very redundant. What if one line went down? Besides, rumor has it that the IT department wants to set up their own additional VLANs on the fourth and fifth floor.

What can be done? Well, the best solution would be to *tag* the two-port trunk. This would allow the trunk to be a member of both VLANs (and any future VLANs). The bad news is that we can't just "convert" an untagged member to be a tagged member. We actually have to remove the untagged port from the

VLAN, and add it back in as a tagged member. As you might have guessed, this will temporarily interrupt service between the two switches. Remember that for tagging to work, *both* sides of the link must be tagged. If one side is tagged and the other side is untagged, no frames will pass successfully.

Do you see why I called this bad planning? It would have been better to make the two-port trunk (ports 1 and 2) *tagged* members of VLAN 20 to begin with. Oh, sure, they would be only members of one VLAN, but they have the potential need to be members of many VLANs. Generally speaking, it's almost always a good idea to *tag* a switch-to-switch link. Even if you're only tagging it to one VLAN, it will save you a great deal of headache when changes are made later.

Okay, let's change the two-port trunk on both switches to be tagged:

```
4thFloorSwitch#conf t
4thFloorSwitch(config)#vlan 20
4thFloorSwitch(config-vlan-20)#no untagged e 1 to 2
4thFloorSwitch(config-vlan-20)#tagged e 1 to 2
added tagged port ethe 1 to 2 to port-vlan 20
4thFloorSwitch(config-vlan-20)#
```

The fourth and fifth floor switches are no longer communicating to each other right now. The fourth floor trunk ports are tagged, and the fifth floor trunk ports are still untagged. To fix this, we perform the same procedure on the fifth floor switch:

```
5thFloorSwitch#conf t
5thFloorSwitch(config)#vlan 20
5thFloorSwitch(config-vlan-20)#no untagged e 1 to 2
5thFloorSwitch(config-vlan-20)#tagged e 1 to 2
added tagged port ethe 1 to 2 to port-vlan 20
5thFloorSwitch(config-vlan-20)#
```

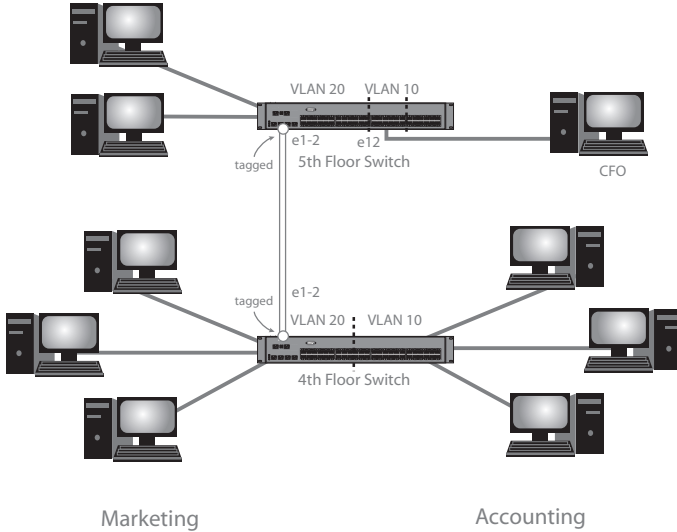
The executives have gotten angry with you because you caused an outage, but you've made your change now. The switches are communicating again. The trunk is now a *tagged* member of VLAN 20. Are we done? Well, the CFO's workstation still can't talk to the Accounting department. The two-port trunk is a tagged member of VLAN 20, but it's still not a member of VLAN 10. Let's fix that now:

```
4thFloorSwitch#conf t
4thFloorSwitch(config)#vlan 10
4thFloorSwitch(config-vlan-10)#tagged e 1 to 2
added tagged port ethe 1 to 2 to port-vlan 10
4thFloorSwitch(config-vlan-10)#
```

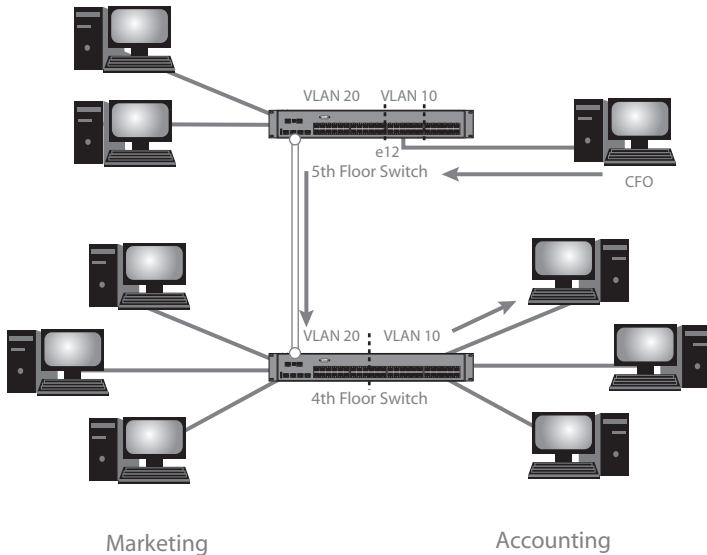
And on the same procedure on the fifth floor switch:

```
5thFloorSwitch#conf t
5thFloorSwitch(config)#vlan 10
5thFloorSwitch(config-vlan-10)#tagged e 1 to 2
added tagged port ethe 1 to 2 to port-vlan 10
5thFloorSwitch(config-vlan-10)#
```

And now, like magic, the CFO is able to communicate with the Accounting department, and all is well.



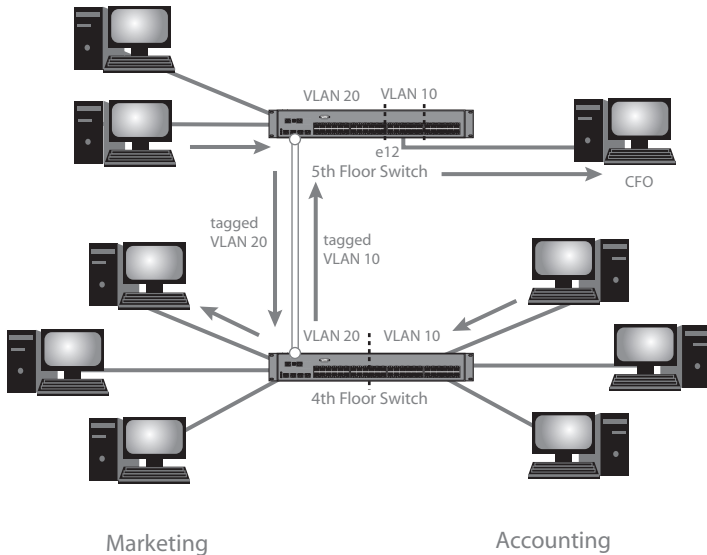
But wait a minute! How do the switches know which frames are meant for VLAN 10 and which frames are meant for VLAN 20? The two-port trunk is *tagged*. Remember those four extra bytes that are added? The frame is now a little bit larger, but it includes, in its header, which VLAN the frame belongs to. If we were able to capture frames on the two-port trunk (either side) and look at the VLAN ID portion of the frame (12-bit field), we would see either 0000 0000 1010 (in decimal, “10”) or 0000 0001 0100 (in decimal, “20”).



When traffic originates from the CFO's workstation, it travels to the fifth floor switch. The fifth floor switch knows it needs to pass the frames to the two-port trunk. It also knows that the two-port trunk is tagged, so it adds the 4-byte 802.1q header to the frame. Inside that header, it specifies that it received the traffic from a host who is a member of VLAN 10. It then sends the altered frames through the two-port trunk.

When the fourth floor switch receives the frames, it recognizes that they are tagged (good news, because the fourth floor switch's ports 1 and 2 are tagged as well). The switch looks at the VLAN ID portion of the 802.1q header, and realizes that the frames are meant for VLAN 10. It also knows that it will be passing the frames to an *untagged* port (the final destination; one of the workstations in the Accounting department). Because of this, it removes the 802.1q header from the frames, and passes the frames to their destination.

This same process is used (in reverse) when the Accounting department sends frames back to the CFO. This is the same process that is used when the Marketing department on the fourth floor wants to send frames to the Marketing department workstations on the fifth floor. This is how the switches know the boundaries of the defined VLAN.



The Default VLAN

Believe it or not, even if you've never configured a VLAN on your switch, there is a VLAN on your switch. It's the *default* VLAN, and every single port is an untagged member of it. On new switches, this is VLAN 1. VLAN 1 is often used for special purposes with other switch manufacturers, so it's considered good practice to change this number. In fact, it is commonly changed to the very last VLAN number: 4095. You can make this change in the switch's Global config:

```
Switch#conf t
Switch(config)#default-vlan-id 4095
```

Even though every switch port starts out as an untagged member of this VLAN, you don't have to do anything special to make a port a member of a new VLAN. Let's say a port is an untagged member of, oh, VLAN 10, and you wanted it to be an untagged member of VLAN 20. You must first remove the untagged port from VLAN 10. Then, you can add it as an untagged port in VLAN 20. With the default VLAN, you don't need to do this. That's one of the things that makes the default VLAN special. If you have a port that's a member of the default VLAN, and you want to make it an untagged member of VLAN 20, you can just untag it as a member. No preliminary steps required.

The other thing that makes this VLAN special is that all ports that have been removed from a VLAN are automatically untagged members of the default VLAN. If you have a port that's an untagged member of VLAN 10, and you remove it from VLAN 10:

```
Switch(config-vlan-10)#no untagged e 27
Switch(config-vlan-10)#
```

The port is now a member of the default VLAN.

The important thing to remember is that whether or not you have any VLANs configured on your switch, all Brocade switches have at least one VLAN.

Dual Port Mode

Feeling comfortable with VLANs now? Let's turn it up a notch then. Remember how I said that a single interface could either be a tagged or untagged member of a VLAN? Well, I left out one other possibility: it can be both. On a Brocade switch, a single port may be an untagged member of one VLAN, and a tagged member of one or more different VLANs.

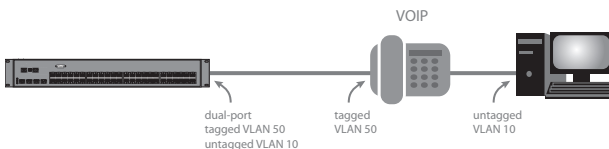
"Both?!?" I hear you cry, "Why in the world would any rationally thinking human being do such a thing?!?" Well, let's look at an example:



Here, we have a Voice-over-IP (VoIP) phone. Most models of these phones have an additional pass-through network port on them. This makes it easier for offices or cubicles that have only one network port. That one port can now house both the phone, and the workstation. The workstation will plug into the phone, and the phone will plug into the network port on the wall.

The problem? Well, it's common to have an entirely separate VLAN (often accompanied by an entirely separate network) for the VoIP phones. Let's say, the phone's VLAN is VLAN 50. Then the workstation needs to be in VLAN 10.

"What's the problem?" you say, "Just tag the switch port as members of both VLANs." Well, that is one solution. The problem? If the switch port is tagged, the connected devices have to be tagged, too. Now, a phone, as in almost any network appliance can be easily tagged. A workstation, on the other hand, is not nearly so simple. In fact, on most operating systems, it is not in the least bit easy, nor desirable, to 802.1q tag your workstation's NIC. This is particularly undesirable with a novice, or even an intermediate level, user. By default, and in the vast majority of configurations, the workstation is untagged. What do we do?



We configure the switch port to be in *dual port mode*. This way, it will be an untagged member of VLAN 10 (for the workstation), *and* it will be a tagged member of VLAN 50 (for the phone). We will need to configure the phone to be

tagged, too. This way, when the phone generates its traffic, it will tag its traffic to be a member of VLAN 50. Because the switch's port is a tagged member of VLAN 50, it will properly receive the tagging, and send the traffic along to the rest of VLAN 50. Likewise, the workstation will send its traffic untagged to the switch. Because the switch has received untagged traffic, it will know that it is meant for VLAN 10 (as the switch port is also an untagged member of VLAN 10).

How do you configure dual port mode? It's a little tricky, because you actually have to start out by tagging *all* of the interfaces involved.

```
Switch#conf t
Switch(config)#vlan 10
Switch(config-vlan-10)#tagged e 12
added tagged port ethe 12 to port-vlan 10
Switch(config-vlan-10)#vlan 50
Switch(config-vlan-50)#tagged e 12
added tagged port ethe 1 to 2 to port-vlan 10
Switch(config-vlan-50)#
```

The last part is where you make the change:

```
Switch(config-vlan-50)#interface e 12
Switch(config-if-e1000-12)#dual-mode 10
```

We used the **dual-mode** command in the interface config to explain that we wanted this interface to be in dual port mode. The “10” designates which VLAN we want to be untagged. Remember how we made the interface a tagged member of both VLANs? Because we configured the interface in dual port mode, and specified VLAN 10, the switch will now treat the interface as an untagged member of VLAN 10.

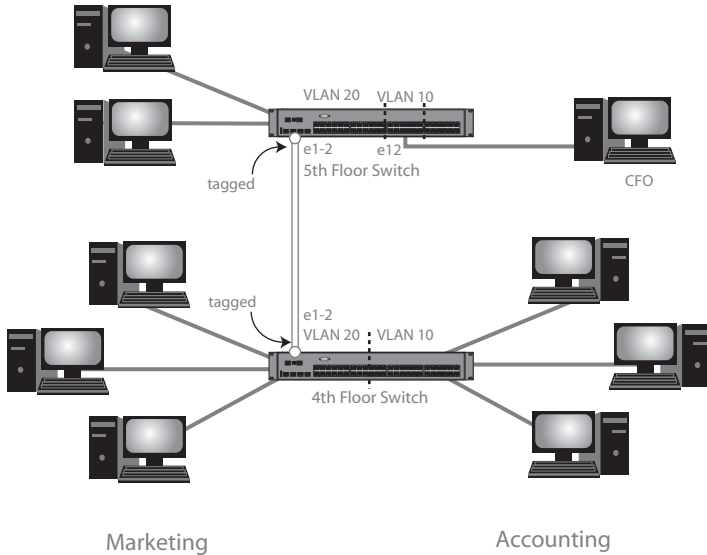
When you're configuring dual port mode, bear in mind that there are still rules to follow. A single interface can only be an untagged member of *one* VLAN. There's no exception to that. Think about it. If a port could be an untagged member of two VLANs, how would the switch know which traffic was meant for which VLAN? There's no VLAN information in an untagged frame. Also, a port cannot be an untagged and a tagged member of the same VLAN. If a port is made a member of a VLAN, you still have to decide whether it's going to be tagged or untagged. It can't be both, in the same VLAN.

The only rule that's really “broken” by dual port mode is that a switch port can be an untagged member of one VLAN, *and* a tagged member of one or multiple VLANs. It just provides you, the network engineer, with some additional freedom, should you need it.

Configuring a Layer 3 VLAN

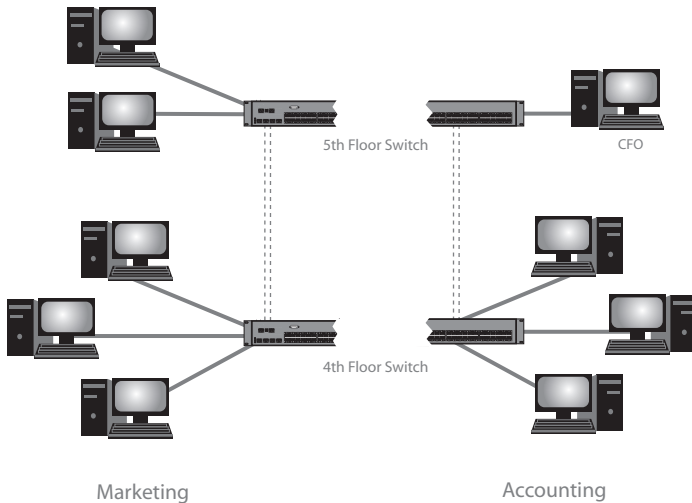
“What? I don’t think you typed that right. We’re talking about VLANs. They exist on Layer 2.” Now let’s twist your brain a little further.

Remember H. Pete & Associates? Let’s look at where we left off there:



On the fourth floor, we have the Accounting VLAN and the Marketing VLAN. On the fifth floor, we have more of the Marketing VLAN, and the CFO has his one workstation in the Accounting VLAN. You may remember that, in this design, the two VLANs cannot communicate to each other. They may look connected, but they are in separate VLANs. It’s just as if they were in entirely different non-connected switches.

In fact, if it helps, logically, it looks like this:

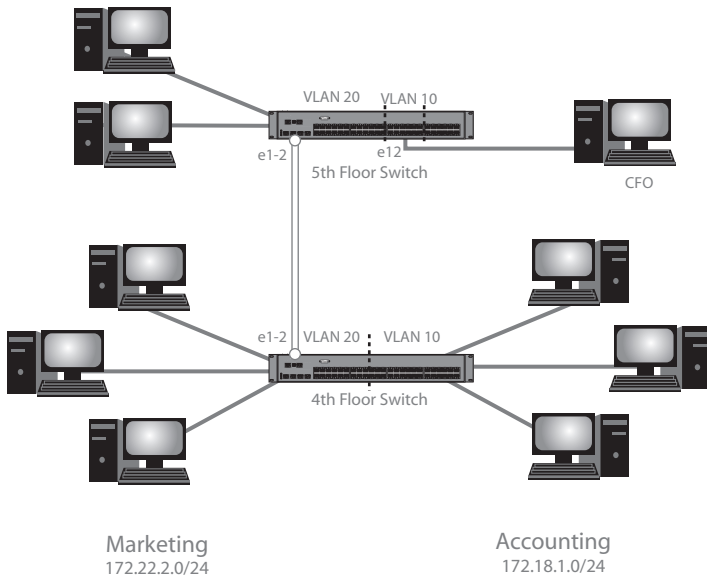


Now, the two-port trunk from the fourth floor to the fifth floor is actually shared, but it's shared privately. The VLAN 10 data never sees the VLAN 20 data, and vice versa.

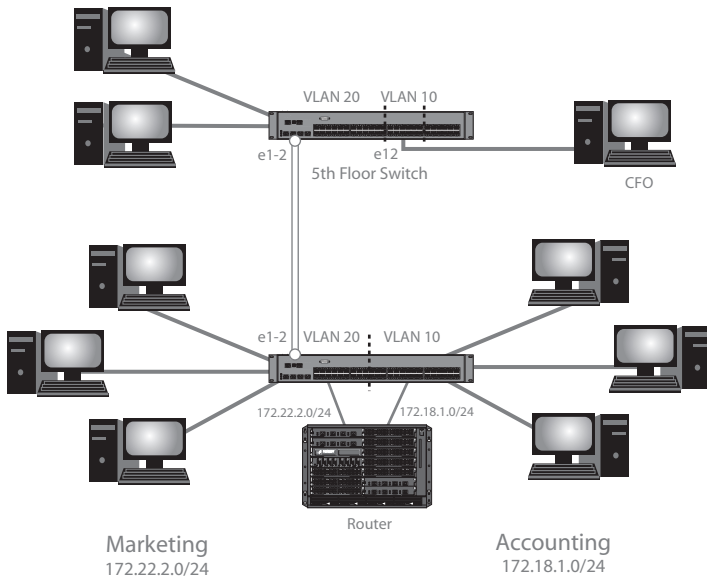
The CEO comes back from his 15-week vacation in the Bahamas, and he declares, "I want the Accounting department and the Marketing department to share information!" In other words, the executives have changed their mind about what's required. How can we handle this? Well, we could put all the machines in the same VLAN. That would make an even bigger broadcast domain. If either department were to grow, that would just make the problem worse. Plus, what if the executives decide next week that they don't want the two departments communicating to each other anymore?

Let's take the design one step further. Let's say that not only are the Accounting and Marketing departments in different VLANs, they're in different networks.

Let's get some IP addresses involved:

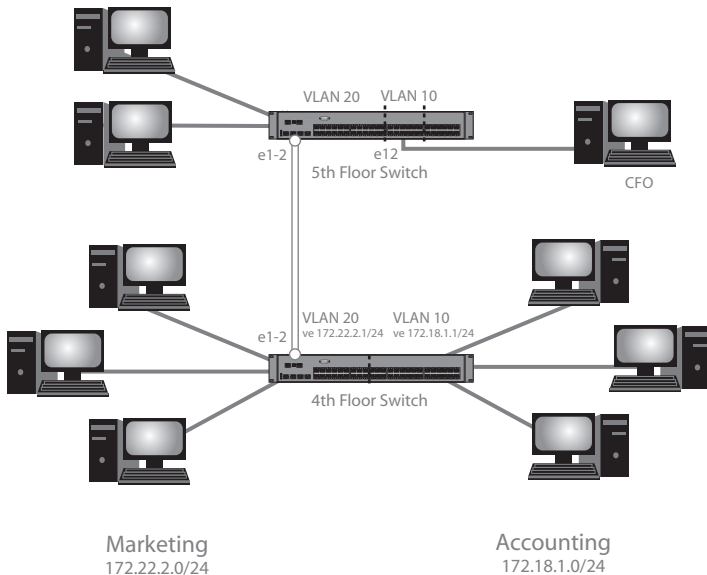


The Accounting department is using 172.18.1.0/24. The Marketing department is using 172.22.2.0/24. Now, even if you put them in the same VLAN, they still wouldn't be able to talk to each other. They're in different broadcast domains. What we need is a router.



Now, any traffic that needs to flow from 172.18.1.0/24 to 172.22.2.0/24 (and vice versa) can travel through the router. The router has one interface (172.18.1.1) in the Accounting VLAN, and one interface (172.22.2.1) in the Marketing VLAN. Problem solved, right? Well, that is, until the CFO finds out you want to spend several thousand dollars to solve the problem. Now, it doesn't look so hot. Wouldn't it be great if you could somehow squeeze the power of the router into your already existing switches? You can see where I'm going with this, can't you?

What makes a VLAN a Layer 3 VLAN? Two things, really. First, you must be running routing code (Base Layer 3 or Full Router). Second, you need to configure a *router-interface*. Think of the router-interface as the physical interface on the router we used in the previous example. This time, we'll configure it inside the fourth floor switch (after we install routing code on it, of course).



To configure the router-interfaces, we first need to declare the router-interfaces:

```
BR-4thFloorSwitch#conf t
BR-4thFloorSwitch(config)#vlan 10
BR-4thFloorSwitch(config-vlan-10)#router-interface ve 10
BR-4thFloorSwitch(config-vlan-10)#vlan 20
BR-4thFloorSwitch(config-vlan-20)#router-interface ve 20
```

When you declare a router-interface, you use a special interface type – *ve*. It stands for *virtual Ethernet*. The interface you're creating doesn't physically exist. Think of it as an interface sitting behind the VLAN, yet inside the switch.

You also have to assign the `ve` a number. This is an 8-bit number, so it can be any number from 1 to 255 (there is no `ve 0`). I chose a number that is the same number as the VLAN. This was purely my choice, and it is not required. I chose to do this, because it makes it easier for me to remember which `ve` is assigned to which VLAN.

Now that we've declared the interfaces, we need to assign them IP addresses. This is done in the same way you would configure IP addresses on any other interface:

```
BR-4thFloorSwitch#conf t
BR-4thFloorSwitch(config)#interface ve 10
BR-4thFloorSwitch(config-vif-10)#ip address 172.18.1.1/24
BR-4thFloorSwitch(config-vif-10)#interface ve 20
BR-4thFloorSwitch(config-vif-20)#ip address 172.22.2.1/24
```

Now, configure all of the workstations in the Accounting department to use 172.18.1.1 as their default gateway. Also, configure all of the workstations in the Marketing department to use 172.22.2.1 as their default gateway. And you're done. The fourth floor switch will now route traffic between the two Layer 3 VLANs.

A Layer 3 VLAN can only have one router-interface. Likewise, a router-interface may only belong to one VLAN. The interface can be addressed with multiple IP addresses. You would assign additional IP addresses the same way that you assigned the first. Just like a router, you can't have an interface with two IP addresses in the same subnet. They must be in different subnets.

VLAN Groups

Finally, let's say you've got a switch with a lot of VLANs. You've got a large chassis switch that you're using as a backbone switch to your entire infrastructure. All VLANs will pass through this switch. You'll find a couple of things. For one, you'll notice that most of your interfaces will be tagged (as they will be trunks coming from other switches). And two, you'll notice that, although you're configuring many different VLANs, each VLAN configuration is identical.

VLAN groups allow you to save config space (and configuration time) by allowing you to group multiple VLANs together. For example, let's say you were needing to configure one four-port trunk (ports 1/1 to 1/4 on your chassis) to be a member of VLANs 30 through 100. Now, you could define each VLAN (e.g., 30, 31, 32, etc.) and make your trunk port a tagged member in each. A more efficient way would be:

```
Switch#conf t
Switch(config)#vlan-group 1 vlan 30 to 100
Switch(config-vlan-group-1)#tagged e 1/1 to 1/4
```

Isn't that much prettier? And it took so much less time to do! But what about Layer 3 VLANs? We've got you covered. A VLAN group may have a virtual interface as well. Because it's a group, instead of a "ve," we use a "group-ve":

```
BR-Switch#conf t
BR-Switch(config)#vlan-group 1 vlan 30 to 100
BR-Switch(config-vlan-group-1)#tagged e 1/1 to 1/4
BR-Switch(config-vlan-group-1)#group-router-interface
BR-Switch(config-vlan-group-1)#interface group-ve 1
BR-Switch(config-vif-1)#ip address 172.18.1.1/24
```

This virtual interface, with its IP address, will be available to VLANs 30 through 100. What if some of the VLANs use different subnets? No problem. Assign the group-ve with multiple IP addresses:

```
BR-Switch(config)#interface group-ve 1
BR-Switch(config-vif-1)#ip address 172.18.1.1/24
BR-Switch(config-vif-1)#ip address 172.22.2.1/24
```

Here again, the router rule applies. A single interface cannot have two IP addresses in the same subnet.

Summary

- A VLAN subdivides a LAN; it breaks up a broadcast domain into smaller broadcast domains
- IEEE 802.1q Frame Tagging allows a single interface to participate in more than one VLAN
- An IEEE 802.1q-tagged frame inserts a 4-byte header into the existing frame
- The VLAN Identifier field is a 12-bit field inside the 802.1q-tagged header; it identifies the VLAN this particular frame belongs to; being 12-bits, the VLAN ID can be any number from 1 to 4,095
- A single interface cannot be an untagged member of more than one VLAN
- A single interface may be a tagged member of many VLANs
- With dual-port mode, a single interface may be an untagged member of one VLAN, and a tagged member of many VLANs
- A Layer 3 VLAN allows for a router interface (or "virtual ethernet" interface) to belong to a VLAN, allowing the switch to route data into and out of the VLAN
- A VLAN Group allows you to configure a VLAN once, but apply that configuration to many VLANs
- All members of a VLAN Group will have identically-defined tagged ports

Chapter Review Questions

1. An IEEE 802.1q-tagged frame adds how much additional data to a standard frame?
 - a. 12 bits
 - b. 4 bytes
 - c. 16 bytes
 - d. It doesn't add any additional data
2. For an interface to be configured in dual-port mode, I must first configure the interface as a _____ member of each VLAN.
 - a. tagged
 - b. untagged
 - c. non
 - d. dual-port
3. What is the largest VLAN ID I can configure?
 - a. 1024
 - b. 65535
 - c. 4095
 - d. 16383
4. An untagged interface may be a member of how many different VLANs?
 - a. 1
 - b. 2
 - c. 8
 - d. 16
5. If I wanted to create VLAN 50 on my switch, which command would I use?
 - a. ip vlan 50
 - b. show vlan 50
 - c. vlan 50
 - d. interface ve 50
6. If I needed to configure hundreds of VLANs on the same switch, and each of the VLANs needed to be configured identically, what could I use?
 - a. VLAN Group
 - b. IEEE 802.1q
 - c. IEEE 802.1p
 - d. Layer 3 VLAN

7. When configuring a Layer 3 VLAN, which command defines the virtual interface for the VLAN?
 - a. `vlan 50`
 - b. `interface ve 50`
 - c. `router-interface ve 50`
 - d. `tagged ve 50`
8. What command would I use to make e 6 an IEEE 802.1q-tagged member of a VLAN?
 - a. `ieee 802.1q interface e 6`
 - b. `q e 6`
 - c. `interface e 6 802-1q`
 - d. `tagged e 6`
9. What command would I use to make the first 16 interfaces of my switch untagged members of a VLAN?
 - a. `ieee 802.1q interface e 1-16`
 - b. `untagged e 1 to 16`
 - c. `untagged e 1-16`
 - d. `interface e 1 to 16 802-1q`
10. What would happen if you connected an untagged interface on one switch to an 802.1q-tagged interface on another switch?
 - a. Communication would fail, the tagged switch would be expecting larger frames
 - b. The tagged switch port would negotiate to dual-port mode
 - c. The tagged switch would superimpose the 802.1q header accordingly
 - d. The untagged switch port would negotiate to dual-port mode

Answers to Review Questions

1. b. 4 bytes. The TPID (which identifies the frame as tagged) takes up 16 bits (2 bytes). The Priority field takes up 3 bits. The CFI takes up one bit. And the VLAN ID field takes up 12 bits. $16 + 3 + 1 + 12 = 32$ bits; $32 \div 8 = 4$ bytes.
2. a. The interface must start as a tagged member of all VLANs. Then, in the Interface config, it will specify the VLAN to which it is to be an untagged member.
3. c. The VLAN ID is a 12-bit field. With 12 bits, the largest number you can represent is 4,095.

4. a. An untagged interface may be a member of one, and only one, VLAN. In dual-port mode, it may be a tagged member of other VLANs, but it may still only be an untagged member of one VLAN.
5. c. This command is issued in the Global config.
6. a. This is the application for the VLAN Group.
7. c. In the VLAN config, you would issue this statement to define the router-interface number. You would use the command in Answer B to configure the interface (with an IP address, etc.).
8. d. This command is issued in the VLAN config.
9. b. This is the proper way to assign VLAN membership of a range of interfaces.
10. a. A tagged interface cannot communicate with an untagged interface (and vice versa). The untagged would be confused by the additional four bytes in the frame, and the tagged would be wondering where the additional four bytes were (so that it could properly direct the frame).

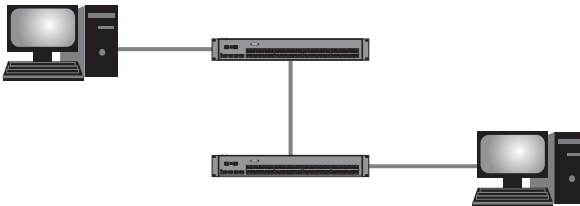
Spanning Tree Protocol

We come to the final chapter of our Layer 2 Section: Spanning Tree Protocol. I want to start this chapter by making it clear that there is so much more to this protocol than we will be covering here. We're going to cover Spanning Tree Protocol at the high level. You should finish this chapter with good understanding of what it is, and what it can do for you.

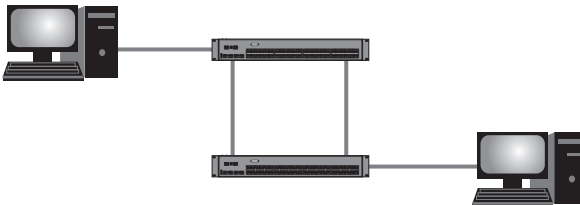
What is Spanning Tree Protocol?

Over a quarter of a century ago, Dr. Radia Perlman, while working at DEC, invented what we now refer to as Spanning Tree Protocol (STP). A lot of people will tell you that the primary purpose of Spanning Tree Protocol (STP) is to prevent loops. Well, they're right, but what does that mean?

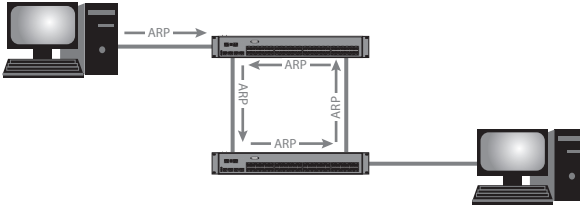
Let's say you had two workstations. Each workstation is connected to a different switch. Those two switches are connected using a single cable. So far, so good, right?



But what if that single cable were to be unplugged or broken? We want to do our best to eliminate single points of failure. We plug another cable in between the two switches; not as a trunk, but as a redundant link.

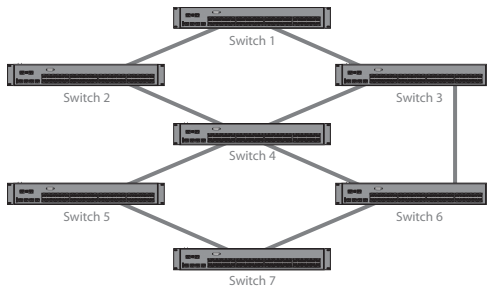


Isn't that great? Now, we're covered if one link goes down, right? Many well-meaning network engineers have started with this scenario only to discover shortly afterwards that their network is down. Why? This scenario creates a Layer 2 loop. If one of the workstations sends a broadcast (say, an ARP request), it would go from the workstation to the switch to the switch, and back and forth between the two switches forever.



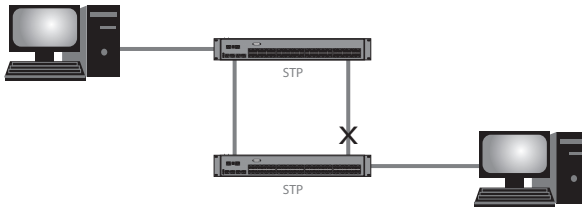
This scenario is referred to as a *broadcast storm*. Because it requires changing the switch's MAC table perpetually (among other functions), both switches' CPU will rise and, likely, peak out at 100% utilization. When any device has used up all of their CPU, unpredictable things happen. In a broadcast storm, traffic for all devices within the broadcast domain (and the switches as a whole) comes to a halt (or a remarkably slow crawl at best).

That's okay. You'll just make sure that you don't have two links going from one switch to another, right? Problem solved. But most of the loops that engineers discover aren't this obvious. Try this one:



Not so easy to see the loop, is it? This scenario, in fact, creates several loops. Tracking down a Layer 2 loop is often one of the hardest tasks any network engineer has to face.

What can be done? Well, thanks to Dr. Perlman, each switch has a mechanism to discover where a loop is and, temporarily, shut it down. This mechanism is called Spanning Tree Protocol or IEEE 802.1d. Let's go back to our two-switch example. What happens when we add STP to both switches?



Notice that one of the links is crossed out. This link is still alive. There's still an established Layer 1 link. The difference is that the switches know not to use that link, unless the other link goes down. This means that we won't have any Layer 2 loops, and we've still got redundant links between the switches. That's a win-win.

IEEE 802.1d (Spanning Tree Protocol)

In review, the Spanning Tree Protocol is designed to prevent Layer 2 loops. It does this by discovering where the loops are, and then making a decision as to which single path will be used (and conversely, which paths will not be used). To discover loops (and to communicate in general with the rest of the switches in the broadcast domain), it uses a special Layer 2 frame.

Bridge Protocol Data Unit (BPDU)

BPDU's are special Layer 2 frames that switches, configured with STP, can use to communicate with other switches in the broadcast domain. There are three types of BPDU's:

- **Configuration BPDU.** These frames are used to elect a root switch, or to receive messages from the root switch to control reconvergence after a link changes
- **Topology Change Notification (TCN) BPDU.** These frames are used by non-root switches to inform the root switch of a link change
- **Topology Change Notification Acknowledgment (TCA) BPDU.** These frames are sent by the root switch to acknowledge receipt of a TCN BPDU from a non-root switch

BPDU's use a multicast (not broadcast) MAC address to send their frames. This is how the communication flows just to switches. When BPDU's are sent, a switch will use the unique MAC address of the port that is sending the message as the source MAC address. It will use a special destination multicast MAC address-01:80:c2:00:00:00-as its destination. BPDU's are sent every two seconds, by default.

The Root Switch

Purists will tell you that the correct term here is *root bridge*. I would like to submit that the term *root switch* be considered. Bridges are very rarely used these days, while switches are extremely popular. You will notice that the term “root bridge” and “root switch” are both found in this book. They are synonymous.

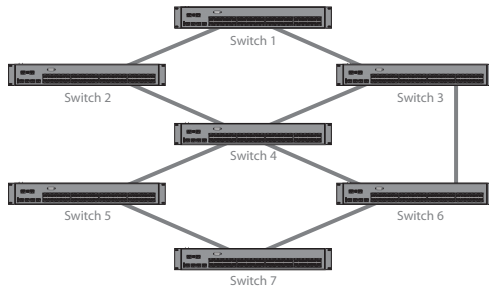
What is a root switch? When the STP process begins, the first item of business is to decide who is right. We have several different paths through the broadcast domain (which, unchecked, would be considered Layer 2 loops). We can temporarily shut down the redundant paths, but how do we decide which paths to keep up and which to shut down? To start, we elect one switch that will always be right. This is the root switch.

The root switch's links will never be shut down. The root switch is king. Whenever a decision is to be made, the root switch's way is the right way.

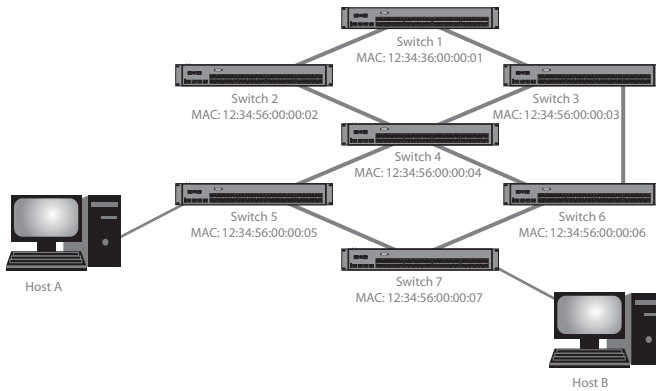
How do we decide who is the root switch? BPDUs are sent out. Contained in the BPDUs is the switch's Bridge ID. The one with the lowest Bridge ID becomes the root switch. The Bridge ID is a combination of the switch's priority, followed by the MAC address of the switch. The priority is a 16-bit number (0 to 65,535). By default, all Brocade switches have a priority of 32,768. If we leave it to the defaults, all of the switches will have the same bridge priority (32,768), so the election will be determined by who has the lowest MAC address.

After the root switch is elected, each switch calculates the shortest path to the root switch. This path is what will remain active. The switch port involved in that path is referred to as the *designated port*. All other paths will be temporarily disabled. On the root switch itself, all ports are listed as *designated*. After all, the root switch is king. It's always right.

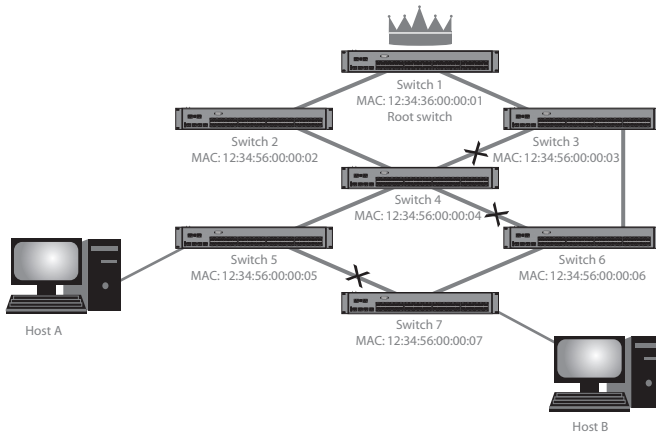
Remember our seven-switch Layer 2 loop example earlier in the chapter? Let's take a look at that again:



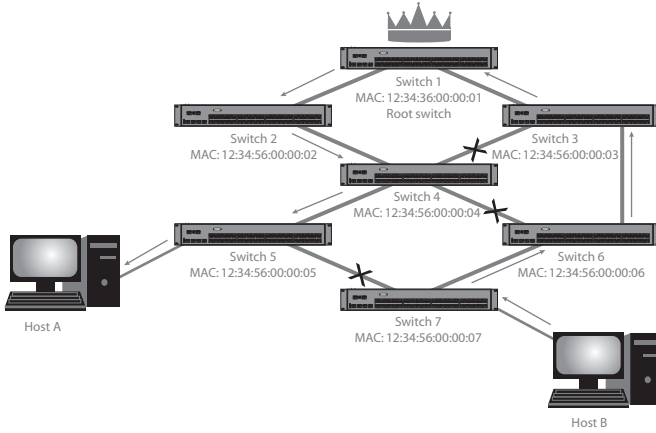
Now, let's look at the MAC addresses for each of these switches:



If we were to enable STP, an election would take place. Who would win? If you guessed Switch 1, give yourself a pat on the back. That is correct. Switch 1 has the lowest MAC address of all seven switches. It becomes the root switch. Now, watch what happens when the paths are decided.



Sometimes, it's not always best to leave the results up to the election, is it? You don't see it? Let's show you the path that Host B will take to talk to Host A:

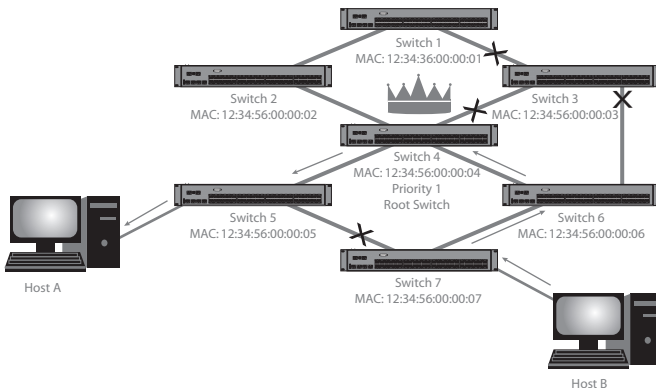


It looks as if Host B was flying from Los Angeles to Las Vegas via the Pacific Ocean, doesn't it? It went through all seven switches! Yet, all it should have needed to do was go from Switch 7 directly to Switch 5, and finally to Host A. How can we fix this?

As we talked about earlier, the bridge priority, by default, is 32,768. This can be changed. Let's say we want Switch 4 to be the root switch. We can make sure Switch 4 will win the election by configuring its priority to a lower number, say, 1.

```
Switch4#conf t
Switch4(config)#spanning-tree priority 1
```

If an election has already taken place, issuing this command will force an immediate re-election. Now, Switch 4 becomes the root switch. What effect will that have on Host A and Host B?



The link between Switch 5 and Switch 7 is still blocked (as it doesn't reflect the shortest path to the root switch for Switch 5 or 7), but the frame takes a much shorter path. It leaves Switch 7, goes to Switch 6, goes to Switch 4 (the root switch), arriving at Switch 5, and finally, to the host itself. Dealing with seven switches, I'm sure you see that we wouldn't be able to maintain the redundancy and provide the shortest possible path for everyone in the broadcast domain. But you need to understand that there are times when leaving the election untampered will yield some undesirable results. You can shape this election in a way that best suits your infrastructure.

Convergence

Convergence is the process any switch interface goes through when participating in the Spanning Tree Protocol. Any interface participating in STP will go through these port states:

1. **Blocking.** Layers 1 and 2 are still active in this state. It's just not passing any normal traffic. The interface in Blocking state can still send and receive BPDUs, but it will neither send nor receive any other traffic. As far as the switch infrastructure is concerned, that port is dead (unusable). An interface starts life in this state as part of a "guilty until proven innocent" method used by STP.
2. **Listening.** This state is also not allowing any traffic other than BPDUs. This state begins by the switch sending out a series of BPDUs and listening to see if they arrive back on another interface. If they do, we have a switching loop, and it will be noted for the next port state. By default, an interface will remain in Listening state for 15 seconds.
3. **Learning.** This state also passes no traffic other than BPDUs. It takes the data we've learned from the Listening state, and it now makes its decision on which ports should be designated and which ports should not. The designated ports are the ports which have the shortest path to the root switch. MAC addresses of the switches in the infrastructure are added to the switch's MAC table. The switch learns how its port fits into the Spanning Tree architecture. By default, an interface will remain in Learning state for 15 seconds, as well.
4. **Forwarding/Blocking.** Some books will tell you that Forwarding is the last state. In reality, it depends on what the results of the two previous states are. If the interface in question has been deemed a designated port (one with the shortest path to the root switch), it will go from Learning to Forwarding. At this point, it will forward and receive all traffic. If the interface in question is deemed not a designated port, it will go into Blocking (forwarding no traffic other than BPDUs). It will remain in blocking until a change in topology happens, and a switch may need to decide again which of its ports are designated and which are not.

With this information in mind, you may have come to the conclusion that it takes a surprisingly long time for a switch port to be added to the infrastructure, using Spanning Tree. You'd be right. It takes a little over 30 seconds for a

switch port to reach Forwarding state (if it ever reaches that). In networking terms, that's quite a long time. As with many things in networking, the trade-off is reliability. If there is a loop to be found, it will be found in those 30 seconds.

There is one more state that you may find a port in: **Disabled**. When a port is in this state, it means that the STP service has been disabled on this port.

STP Commands

There are many commands associated with STP. Here are a few standards that you should know well.

For one, to configure STP globally (all interfaces on the switch):

```
Switch(config)#spanning-tree
```

Simple, isn't it? To turn off STP globally:

```
Switch(config)#no spanning-tree
```

Did you guess that? Good for you! You're paying attention. This may seem almost too simple to go over, but it is important. If you are running switch (Layer 2) code on your switch, STP is *enabled* globally by default. If you are running routing (Layer 3) code, STP is *disabled* globally by default. The default may, or may not, be what you had intended. Remember, *switches enable STP by default, and routers disable STP by default*.

Spanning Tree can also be enabled or disabled at the individual port level. The commands are the same, but note that they take place at the Interface Config level:

```
Switch(config)#interface e 9
Switch(config-if-e1000-9)#spanning-tree
```

Or, to disable spanning tree on an individual port:

```
Switch(config)#interface e 9
Switch(config-if-e1000-9)#no spanning-tree
```

We talked about switch priority already, but here it is again. Notice that this is configured at the Global Config level:

```
Switch(config)#spanning-tree priority 1
```

When a switch is deciding which of its redundant interfaces to set to Forwarding and which to set to Blocking, it uses a similar method to the election of the root switch. All things being equal, it will choose the interface with the lowest MAC address. As we discovered in the root switch election scenario, that may not always be desirable. Not to worry. Individual interfaces have priority settings, too. This is an 8-bit number (0 to 255) and defaults to 128. If you want your switch to favor a particular interface, give it a lower priority number. Peculiar thing about this number. It must be divisible by 16. For example, a port priority of 7 is not acceptable. It must be either 0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224, or 240:

```
Switch(config)#spanning-tree e 9 priority 16
```

Notice that this setting was not entered at the Interface Config level, but the Global config level.

These next three commands configure STP timers. Use these with a great deal of caution. Know what you are doing when you change these settings, especially when interacting with switches from other manufacturers. Know that all of your participants in the infrastructure are going to be expecting STP defaults. Because of this, tampering with the default settings may cause network instability. Use at your own risk.

- **Forward-delay.** This is the time an interface will stay in the Listening and Learning states. By default, this is 15 seconds, as we discussed earlier. The timer for Listening and Learning will always be equal, so the number is only set once. Here's an example of the command you would use to change the forward-delay timer to 10 seconds (speeding up convergence from 30 seconds to 20 seconds):

```
Switch(config)#spanning-tree forward-delay 10
```

- **Hello-time.** The root switch sends BPDUs to all other switches every two seconds, by default. This is the hello-time. Let's say, for some reason, you'd like to change this to five seconds:

```
Switch(config)#spanning-tree hello-time 5
```

- **Max-age.** This is the amount of time a non-root switch will wait before it decides the root switch is dead and call for a re-election. By default, a non-root switch will wait for 20 seconds. This timer is configured on the root switch. In other words, the root switch will dictate how long the other switches should wait before declaring him dead. This command changes the advertised "max-age" to 45 seconds:

```
Switch(config)#spanning-tree max-age 45
```

By far, the handiest of STP commands are those that help you troubleshoot problems; the commands that show you how things actually are. The standard? Take a guess:

```
Switch#show spanning-tree
STP instance owned by VLAN 1
Global STP (IEEE 802.1D) Parameters:
VLAN Root Root Root Prio Max He- Ho- Fwd Last Chg Bridge
ID   ID Cost Port rity Age llo ld  dly Chang cnt  Address
                                Hex  sec sec sec sec sec
1 80000000abc123c00 4 1 8000 20 2 1 15 3499283 0
000abc4321a0
Port STP Parameters:
Port Prio Path State Fwd Design Designated
Designated
Num rity Cost Trans Cost Root Bridge
Hex
1 80 4 FORWARDING 1 0 80000000abc123c00 80000000abc123c00
2 80 4 BLOCKING 0 0 80000000abc123c00 80000000abc123c00
```

This command gives you a lot of good information. Notice that it gives you the Bridge ID of the root switch (8000000abc123c00). Two interfaces are participating in this Spanning Tree (ports 1 and 2). Port 1 is in Forwarding state. Port 2 is in Blocking state. Notice that the timers and the priorities we talked about are defined.

If you add the suffix detail to this command, you get a more thorough report:

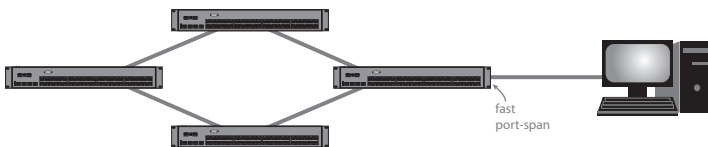
```
Switch#show spanning-tree detail
Bridge identifier      - 0x8000000abc4321a0
Active global timers - None
Port 1 is FORWARDING
  Port - Path cost: 4, Priority: 128, Root: 0x8000000abc123c00
  Designated - Bridge: 0x8000000abc123c00, Interface: 96, Path
cost: 0
  Active Timers - Message age: 2
  BPDUs - Sent: 0, Received: 1750044
Port 2 is BLOCKING
  Port - Path cost: 4, Priority: 128, Root: 0x8000000abc123c00
  Designated - Bridge: 0x8000000abc123c00, Interface: 96, Path
cost: 0
  Active Timers - Message age: 2
  BPDUs - Sent: 0, Received: 1750044
```

Notice that this information is giving much more detail in regards to the individual interfaces. You can see that it gives you much of the same information we had in the **show spanning-tree** command, but it adds some additional data (e.g., BPDUs sent and received).

Speeding Up Convergence

We've gone over the different timers that you can change to speed up convergence. You've also been warned that you should, in practice, not change these settings from their default. You do have other options: *Fast Port Span* and *Fast Uplink Span*.

Fast Port Span is what you would assign to a port that is connected to a workstation or server. The idea is that you're explaining to the switch: "This port's a dead-end; no need to worry about Layer 2 loops here."



Fast Port Span automatically changes your convergence time from 30 seconds (15 seconds in Listening; 15 seconds in Learning) to 4 seconds (2 seconds in Listening; 2 seconds in Learning). The good news is that Fast Port Span is enabled on all of your switch's ports by default.

But the switch is no dummy. It will automatically use standard STP (*not* Fast Port Span) if it detects any of the following disqualifying conditions on a given port:

- The port is an 802.1q tagged member of a VLAN
- The port is a member of a trunk group
- The switch sees more than one MAC address coming from that port; this indicates that there's a hub (or another switch) upstream, and that could mean a possible Layer 2 loop
- The switch sees an STP BPDU coming in on the port; this would indicate that there's another switch upstream, so there would be a potential Layer 2 loop

If, for whatever reason, you'd like to disable this feature, you would do so in the Global config:

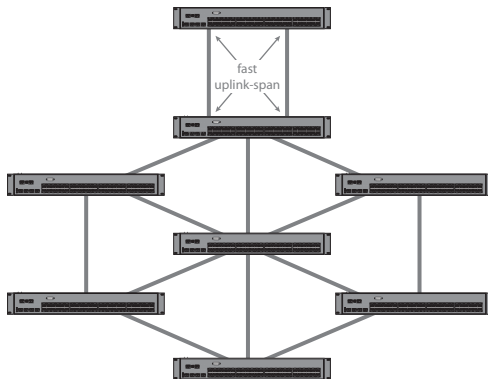
```
Switch(config)#no fast port-span
```

You can re-enable it, as well. Also, if you like, you can enable it switch-wide, but disable it on certain ports:

```
Switch(config)#fast port-span exclude e 48
```

This will keep Fast Port Span enabled on all of your switch ports except e 48. When Fast Port Span is disabled, it will use the standard STP timers.

This is fine if you're dealing with workstations, but what about other switches? Well, let's say you have an edge switch that represents a dead end of the network. You've got two links (not trunked) going from this switch to the core switch.



You know that this switch represents an end to your network. There are no further uplinks to any other switches. Well, then, let's configure the core switch ports to use Fast Uplink Span. Notice that this is done in the Global config:

```
Switch(config)#fast uplink-span e 15 to 16
```

The two links are still part of STP. One will be in Forwarding. One will be in Blocking. The difference is that when the Forwarding link fails, the core switch will converge the second link to Forwarding in four seconds (two seconds for Listening; two seconds for Learning). It works very similar to Fast Port Span.

This tool is to be used cautiously. With Fast Port Span, if you plug another switch or a hub into the port (with Fast Port Span), the switch will detect it, and it will be automatically disabled. With Fast Uplink Span, the switches involved are presuming that you know what you're doing. Make certain you are only using this command for an edge switch. Anything else would cause the same potential for network instability as altering the timers.

IEEE 802.1w: When IEEE 802.1d Just Isn't Fast Enough

IEEE 802.1w is also referred to as Rapid Spanning Tree Protocol (RSTP). It is an enhancement to 802.1d, largely dealing with convergence time. Where 802.1d converges in at least 30 seconds, 802.1w may converge in literally less than half a second.

RSTP vs. STP

RSTP is actually very similar to STP. It is, after all, an enhancement to STP. There is still a root switch. The main thing that makes it different is that it does not depend on timers to establish loops. Each port learns its state through initial handshakes. Little time is wasted when roles need to change.

The port states are a little different in RSTP:

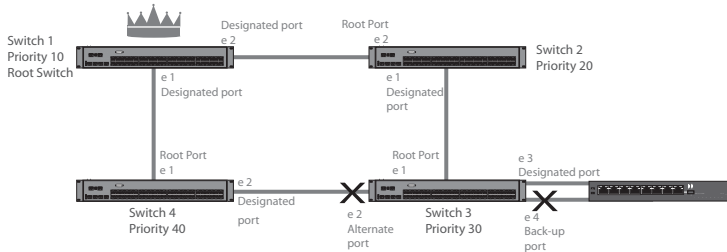
1. **Discarding.** This is state is analogous to STP's Blocking. It forwards no traffic, but it still receives BPDUs.
2. **Learning.** This corresponds to the Learning state in STP. The port is building its MAC table, but not forwarding traffic yet.
3. **Forwarding.** This corresponds to the Forwarding state in STP. The port is active. It is now forwarding traffic.

Also, like STP, a port may be in a **Disabled** state. This would indicate that the port itself is disabled (physically down or administratively disabled), or has spanning-tree disabled.

After initial handshakes, each participating interface will find itself in one of these roles:

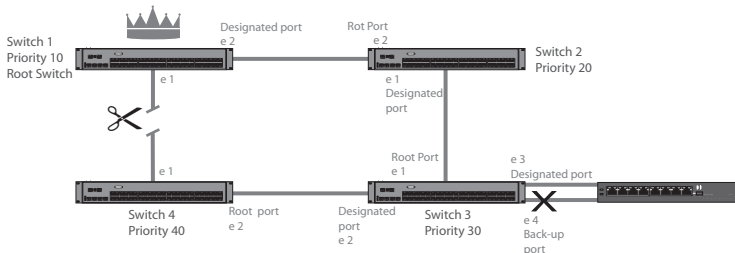
- **Root.** This port has been deemed the best path to the root switch.
- **Designated.** All ports start in this role. On the root switch, all ports stay in this role. It simply means that, of all the BPDUs you've received on this port, yours are the best (highest priority, etc.).
- **Alternative.** This is a non-root port. It will be in Discarding state, unless the switch's root port goes down.

- Backup.** This is similar to Alternative, but it is in Discarding state because it received a superior BPDUs from itself (its own switch). This is certainly an unusual circumstance. Try to think of it as a hub with two links connecting to your switch. The switch will send BPDUs out both ports. The superior BPDUs will be in Forwarding state. The other will be in Backup state. The thing that separates Backup from Alternative is that it has been placed in Discarding because it received a superior BPDUs from its own switch. Alternative is placed in Discarding because it received a superior BPDUs from a different switch.



You can see here that the switch with the lowest priority (Switch 1) has become the root switch. Switch 2 and Switch 4 have direct connections to the root switch, so they become root ports. On Switch 3, e 1 becomes the root port because Switch 2 has the lower priority (as opposed to Switch 4). On e 2 of Switch 3, Switch 4 has the shorter path to root, so its e 2 is Designated, while Switch 3's e 2 is Alternative. Also on Switch 3, e 3 is Designated and e 4 is in Backup. This is regarding traffic to the attached hub.

Now let's see what happens when a failure occurs:



Very little needed to change here. Switch 1's e 1 port went down. It's still the root switch. Switch 4 has to make a change, as it has lost its root port. Its Designated port (e 2) is instantly upgraded to Root port. Because of that, Switch 3's port is immediately upgraded to Designated port. These changes happen sub-second, and are almost undetectable by the end users.

Configuring RSTP

How do we configure it? It is not enabled by default (most likely to maintain compatibility with other manufacturers' switches that may not be so lucky). To enable RSTP globally, go to the Global config:

```
Switch(config)#spanning-tree 802-1w
```

Notice that you use a "-" instead of a "." to separate the "2" from the "1." This can get confusing, so you'll see a lot of more experienced engineers simply using the short-cut:

```
Switch(config)#span 8
```

This command accomplishes the exact same thing. You can also define the switch's priority:

```
Switch(config)#spanning-tree 802-1w priority 10
```

When dealing with RSTP, it's wise to define the individual interfaces in one of two types. The premise of these types is very similar to STP's Fast Port Span and Fast Uplink Span:

- **Admin-edge-port.** This is a similar concept to Fast Port Span in STP. There's no timer change (as RSTP doesn't use timers in the same way), but you've defined this port as a dead end (e.g., a workstation, a server, etc.).
- **Admin-pt2pt-mac.** This is a similar concept to Fast Uplink Span in STP. You are defining this port as a "point-to-point" connection. It is a direct link between two switches.

You would configure this at the Interface config:

```
Switch(config-if-e1000-9)#spanning-tree 802-1w admin-pt2pt-mac
Switch(config-if-e1000-9)#int e 15
Switch(config-if-e1000-15)#spanning-tree 802-1w admin-edge-port
```

802.1w show Commands

The two main show commands you'll want to use are very similar to the two we introduced in STP. The first is:

```
Switch#show 802-1w
--- VLAN 10 [ STP Instance owned by VLAN 10 ] -----
-----
Bridge IEEE 802.1W Parameters:
Bridge          Bridge Bridge Bridge Force      tx
Identifier      MaxAge Hello   FwdDly Version Hold
hex             sec      sec      sec          cnt
8000000abc123a01 20      2        15          Default 3
RootBridge      RootPath DesignatedBri-   Root  Max Fwd Hel
Identifier      Cost      dge Identifier   Port  Age Dly lo
hex             hex              hex          sec sec sec
8000000abc123a01 0          8000000abc123a01 Root 20 15 2
```

Port IEEE 802.1w Parameters:

```
<--- Config Params -->|<----- Current state ----->
Port  Pri PortPath  P2P Edge Role State Designa-  Designated
Num Cost          Mac Port          ted cost  bridge
7   128 20000    F   F DESIGNATED FORWARDING 0 80000000abc123a01
8   128 20000    F   F DESIGNATED FORWARDING 0 80000000abc123a01
9   128 20000    F   F DISABLED   DISABLED   0 80000000abc123a01
```

Like **show spanning-tree**, this command shows you the root switch (bridge) ID, timers, and the roles and states of the participating interfaces. Similar again to STP, there is also a more detailed report:

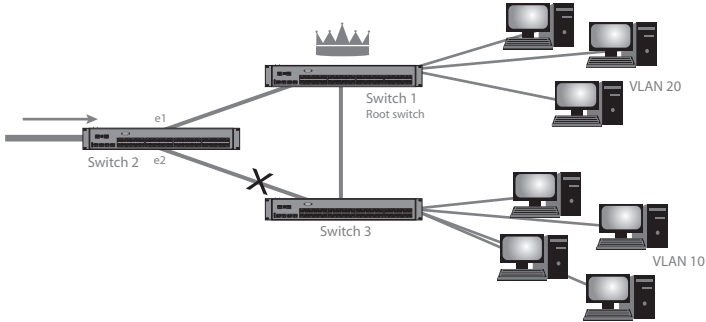
```
Switch#show 802-1w detail
=====
VLAN 10 - MULTIPLE SPANNING TREE (MSTP - IEEE 802.1w) ACTIVE
=====
BridgeId 80000000abc123a01, forceVersion 2, txHoldCount 3
Port 7 - Role: DESIGNATED - State: FORWARDING
  PathCost 20000, Priority 128, AdminOperEdge F, AdminPt2PtMac
F
  DesignatedPriority - Root: 0x80000000abc123a01, Bridge:
0x80000000abc123a01
  ActiveTimers - helloWhen 3
  MachineStates - PIM: CURRENT, PRT: DESIGNATED_PORT, PST:
FORWARDING
                        TCM: ACTIVE, PPM: SENDING_RSTP, PTX:
TRANSMIT_IDLE
  Received - RST BPDUs 37, Config BPDUs 0, TCN BPDUs 0
Port 8 - Role: DESIGNATED - State: FORWARDING
  PathCost 20000, Priority 128, AdminOperEdge F, AdminPt2PtMac
F
  DesignatedPriority - Root: 0x80000000abc123a01 , Bridge:
0x80000000abc123a01
  ActiveTimers - helloWhen 3
  MachineStates - PIM: CURRENT, PRT: DESIGNATED_PORT, PST:
FORWARDING
                        TCM: ACTIVE, PPM: SENDING_RSTP, PTX:
TRANSMIT_IDLE
  Received - RST BPDUs 37, Config BPDUs 0, TCN BPDUs 0
Port 9 - DISABLED
```

Notice that this output defines whether or not a port has been defined as “admin-pt2pt-mac” or “admin-edge-port.” Also, Rapid Spanning Tree (RST) BPDUs are distinguished from regular STP BPDUs. Notice that these are logged on the individual interfaces.

IEEE 802.1s (MST)

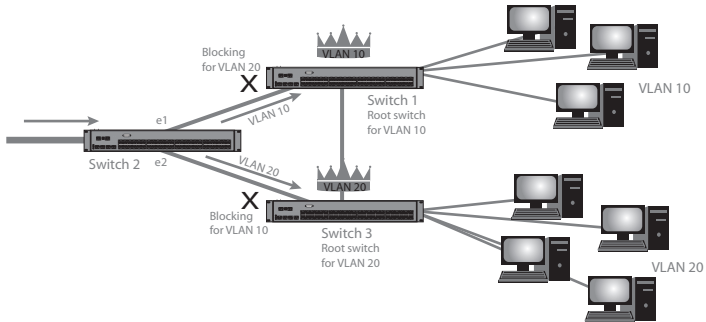
If you've been paying close attention to some of the previous show command outputs, you may have noticed mention of MST. It stands for Multiple Spanning Tree, and it's an open standard defined in IEEE 802.1s. It is enabled by default on all Brocade switches running switch code.

What is it? Well, let's say you are using the STP you know and love from the beginning of this chapter. You've got three switches and two VLANs configured across all the switches:



Notice that the VLAN 20 traffic is not really traveling optimally. It's going from Switch 2 all the way through Switch 1 to Switch 3 (where the hosts are). But STP is providing an optimal path for VLAN 10. Wouldn't it be great if we could provide an optimal path for VLAN 10 *and* an optimal path for VLAN 20? "Now you're dreaming," I hear you moan. Am I?

When we've been configuring the switch for STP (or even RSTP), we've been doing so at the Global level. All we're doing here is making it possible for that same type of configuration, but done at the VLAN config level. Let's take a look at the diagram after we've taken advantage of MST:



There are "two" different Spanning Trees defined here. For VLAN 10, Switch 1 becomes the root switch. For VLAN 20, Switch 3 becomes the root switch. On Switch 2, e 2 is in Blocking state for VLAN 10 (as e 1 is the Designated port), but e 2 is the Designated port for VLAN 20 (and in Blocking for VLAN 10).

The advantage with MST is that each VLAN can have its own Spanning Tree. Each VLAN can have its own root switch. Interfaces that may be in Blocking state in one instance of Spanning Tree may still be in Forwarding state by

another instance of Spanning Tree (in a different VLAN). In fact, you can mix and match further. For example, one VLAN's Spanning Tree may be using 802.1d. Another VLAN's Spanning Tree may be using 802.1w.

You will hear MST sometimes referred to as Per VLAN Spanning Tree (PVST). PVST is a predecessor to what has evolved into IEEE 802.1s.

There is nothing to consciously configure to “enable” 802.1s. It's there for you to use. All you need to do is define it at the VLAN config level:

```
Switch(config)#vlan 10
Switch(config-vlan-10)#spanning-tree
```

You could also configure 802.1w:

```
Switch(config-vlan-10)#spanning-tree 802-1w
```

You can configure your switch's priority, as it pertains to your VLAN's Spanning Tree:

```
Switch(config-vlan-10)#spanning-tree priority 10
```

You can adjust timers (again, unique to your VLAN's Spanning Tree):

```
Switch(config-vlan-10)#spanning-tree max-age 15
```

If you're using RSTP, you can define ports as “admin-pt2pt-mac” or “admin-edge-port”:

```
Switch(config-vlan-10)#spanning-tree 802-1w e 9 admin-pt2pt-mac
Switch(config-vlan-10)#spanning-tree 802-1w e 15 admin-edge-port
```

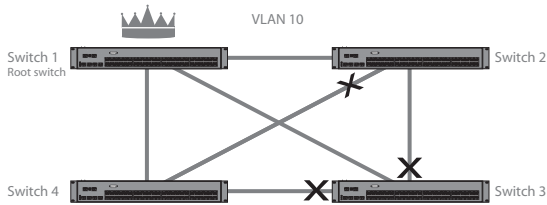
Or, if you prefer, you can disable Spanning Tree all together, for your particular VLAN:

```
Switch(config-vlan-10)#no spanning-tree
```

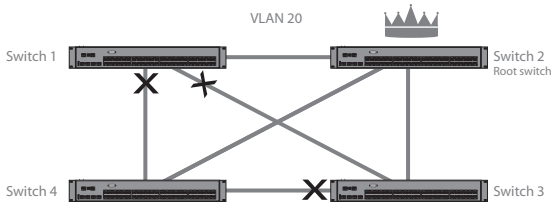
The important thing to remember is that all of these commands are being issued within the VLAN Config mode. It's your own global Spanning Tree encapsulated in your VLAN. You can do anything with it that you would normally have done on the global level.

STP Load Sharing

Let's say you have four switches fully meshed. Let's say you are running at least two VLANs on all four of these switches. If we left the Spanning Tree elections to their natural state, both VLANs would most likely choose the same links to be in Forwarding and Blocking.



Using MST, you can configure Spanning Tree such that the Blocking links for one VLAN are the Forwarding links for the other (and vice versa). This way, all the available links are being actively used. This provides greater bandwidth for both VLANs.



All we had to do is change a priority. This technique is referred to as STP Load Sharing. STP, by nature, is active/standby. When two or more paths are presented, one is chosen, and the others are disabled. With STP Load Sharing, you are taking advantage of STP's active/standby nature by making links active for some, and “standby” for others. It's a great way to get more out of your infrastructure.

Topology Groups

MST is not necessarily the end-all solution for every infrastructure. For one, MST for every defined VLAN doesn't scale well. Running over 100 Spanning Tree instances is very CPU and memory intensive. Plus, you could run out of Spanning Tree instances before you run out of VLANs. Most chassis will allow up to 4,000 VLANs, but only 128 STP instances.

One way to solve this problem is with Topology Groups. Topology Groups take a group of VLANs and assign one instance of STP to it. Now, all of the VLANs in the Topology Group have to share the same instance of STP. A Topology Group can contain as few or as many VLANs as you'd like.

Each Topology Group has a Master VLAN. This VLAN is where your STP settings are defined. Say, for example, you had three VLANs: 10, 30, and 50. You want these three VLANs to share the same STP instance, and you want to configure that instance in VLAN 30.

First, you'd have to configure the VLANs:

```
Switch(config)#vlan 10
Switch(config-vlan-10)#tagged e 1 to 4
Switch(config-vlan-10)#vlan 30
Switch(config-vlan-30)#tagged e 1 to 4
Switch(config-vlan-30)#vlan 50
Switch(config-vlan-50)#tagged e 1 to 4
```

Then, you'd want to add any specific STP settings you wanted to VLAN 30. Let's say, you wanted to set the STP priority to 10:

```
Switch(config)#vlan 30
Switch(config-vlan-30)#spanning-tree priority 10
```

Now, you need to define the Topology Group:

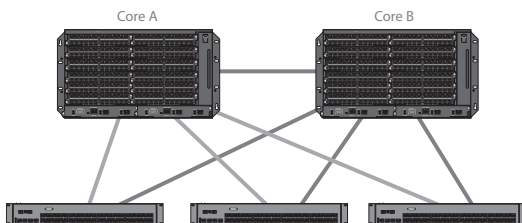
```
Switch(config)#topology-group 1
Switch(config-topo-group-1)#master-vlan 30
Switch(config-topo-group-1)#member-vlan 10
Switch(config-topo-group-1)#member-vlan 50
```

Notice that we've defined VLAN 10 and VLAN 50 as *member* VLANs. That is to say, these VLANs are members of the Topology Group, but not the Master VLAN. As defined above, the Master VLAN is 30. Any STP configurations made under the VLAN 30 configuration will be used for VLANs 10 and 50 and any other members of the Topology Group. You can configure up to 256 Topology Groups on a single switch. A VLAN may be a member of only one Topology Group.

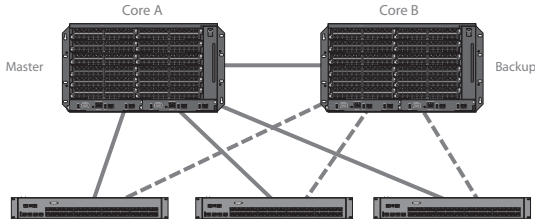
Virtual Switch Redundancy Protocol (VSRP)

There is an alternative to Spanning Tree Protocol (and its variants). It is Virtual Switch Redundancy Protocol (VSRP). This protocol is less about redundant links, and more about redundant switches. It is based on VRRP-e, a protocol that we'll talk more about in Chapter 13.

Consider this arrangement:

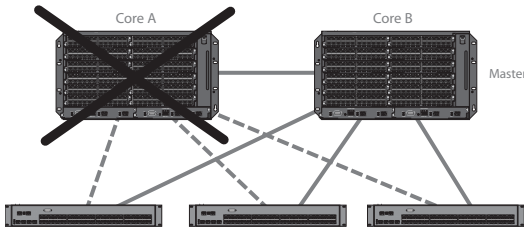


We would configure Core Switch A and Core Switch B with VSRP. They would hold an election to see which of them should be considered Master. Let's say that the election results in Core Switch A becoming the Master. Core Switch B would now be the Backup. Traffic will not be forwarded on Core Switch B's links. Core Switch B would only listen for Hello packets from Core Switch A (the Master).



The three lower switches, in this scenario, do not have to be VSRP-aware. In fact, they don't even have to be Brocade switches. VSRP-aware switches will failover faster. This is something to consider when designing an infrastructure like this.

The Master switch (Core Switch A) is now forwarding traffic and is also sending Hello packets to the Backup (Core Switch B). Hello packets are sent out all of the VSRP ports on the Master switch (it's a good idea to have more than one). When the Backup stops receiving Hello packets after a defined period of time, it presumes that the Master is dead, and it assumes the role of Master. Core Switch B's links are now active, and Core Switch A's links are not.



The failover time for VSRP is similar to RSTP. It's sub-second (for VSRP-aware switches). In this example, we've used one Master and one Backup. Depending on configuration, an election can result in one Master and up to four Backup switches.

To configure VSRP, let's presume that all the links in our diagram are in the same VLAN (VLAN 10). Within VLAN 10's config, we need to define a Virtual Router ID (VRID).

We need to define this on both switches. For Core Switch A:

```
CoreSwitchA(config)#vlan 10
CoreSwitchA(config-vlan-10)#vsrp vrid 1
CoreSwitchA(config-vlan-10-vrid-1)#backup
CoreSwitchA(config-vlan-10-vrid-1)#enable
```

We arbitrarily chose a VRID of 1. Notice that we defined Core Switch A as “backup.” All switches participating in the same VRID must be defined as Backups. The election will promote the one switch to Master. Finally, we enabled VSRP. Now, for this to function, we must configure Core Switch B as well:

```
CoreSwitchB(config)#vlan 10
CoreSwitchB(config-vlan-10)#vsrp vrid 1
CoreSwitchB(config-vlan-10-vrid-1)#backup
CoreSwitchB(config-vlan-10-vrid-1)#enable
```

Now an election will be held, Core Switch A will be declared Master (let's say), and the design will continue as we've illustrated earlier in the Chapter. Like STP, this is an active/standby configuration. One is active. The others are inactive. VSRP is simply another method to provide redundancy to your network infrastructure.

Summary

- Spanning Tree Protocol prevents Layer 2 loops
- A Layer 2 loop may cause a broadcast storm, which will cause the switches' CPUs to be overwhelmed
- Spanning Tree Protocol is defined in IEEE 802.1d
- Spanning Tree Protocol uses BPDU frames to discover loops and exchange information
- An election decides which switch is the root switch
- The root switch interfaces are never in “Blocking” state
- Configuring a switch's priority allows the engineer to manipulate the election
- IEEE 802.1d Spanning Tree goes through four states before the interface may pass traffic:
 - Blocking - preventing all traffic, aside from BPDUs
 - Listening - listening for BPDUs returning (indicating a loop); this takes 15 seconds
 - Learning - learning the Layer 2 infrastructure including MAC addresses and the root switch; this takes 15 seconds
 - Forwarding/Blocking - allowing (or preventing) traffic based on the result of the first three states

- Fast Port Span and Fast Uplink Span help speed up the interface convergence process
- Rapid Spanning Tree Protocol (RSTP), defined by IEEE 802.1w, speeds up interface convergence to the sub-second level
- Multiple Spanning Tree (MST), defined by IEEE 802.1s, provides a Spanning Tree instance tied to a VLAN; this means that a trunk port between switches could be forwarding for one VLAN, but blocking for a different VLAN
- Topology Groups provide one Spanning Tree instance to a group of VLANs
- Virtual Switch Redundancy Protocol (VSRP) provides active/standby redundancy for the Layer 2 infrastructure

Chapter Review Questions

1. On a Brocade switch that is globally running STP, you plug in your laptop. What state is this switch port in?
 - a. Blocking
 - b. Listening
 - c. Learning
 - d. Forwarding
2. On a Brocade switch that is globally running STP, you plug in your laptop. How long will it take before your laptop is able to start passing traffic?
 - a. 0 seconds (instantly)
 - b. 10 seconds
 - c. 30 seconds
 - d. 4 seconds
3. On a Brocade switch that is globally running STP, you plug in another switch. How long will it take before the switches start passing LAN traffic to each other?
 - a. 0 seconds (instantly)
 - b. 10 seconds
 - c. 30 seconds
 - d. 4 seconds

4. BPDUs:
 - a. Are used to share routing tables between switches
 - b. Are multicast frames sent between switches participating in STP (or any of its variants)
 - c. Are broadcast frames sent between switches participating in STP (or any of its variants)
 - d. Use UDP to send MAC table information between switches
5. The root switch is elected as the switch with:
 - a. The lowest Bridge ID
 - b. The lowest MAC address
 - c. The lowest Bridge priority
 - d. The lowest IP address
6. On a non-root switch, RSTP will label the port with the shortest path to the root switch as:
 - a. Designated Port
 - b. Root Port
 - c. Backup Port
 - d. Alternate Port
7. Which STP parameter would you adjust if you wanted the root switch to send BPDUs less frequently?
 - a. max-age
 - b. forward-delay
 - c. hello-time
 - d. priority
8. On a Brocade switch that is globally running STP, an interface is actively passing LAN traffic. What state is that interface in?
 - a. Blocking
 - b. Listening
 - c. Learning
 - d. Forwarding

9. Which STP variant allows for a single STP instance per VLAN?
 - a. IEEE 802.1d
 - b. IEEE 802.1s
 - c. IEEE 802.1q
 - d. IEEE 802.1w
10. An interface is actively discovering where the Layer 2 loops are (if any). What state is this interface in?
 - a. Blocking
 - b. Listening
 - c. Learning
 - d. Forwarding

Answers to Review Questions

1. a. All interfaces start in Blocking state. “Guilty until proven innocent.”
2. d. This is a tricky one. Remember that “fast port-span” is enabled by default. The port will not fit the qualifications of an uplink to another switch, so the Listening and Learning states will each only last two seconds.
3. c. This time, the switch will manifest itself as an uplink (by sending BPDUs, at the very least). The “fast uplink-span” option is not enabled by default, so the port will go through the standard STP convergence cycle. It will spend 15 seconds in Listening, and 15 seconds in Learning.
4. b. BPDUs are multicast, not broadcast.
5. a. This is a tricky one. Yes, the bridge priority and MAC address both play a role in the election, but not distinctly. Both numbers are combined into one number: the Bridge ID. That is the number that is used for the election. The lowest Bridge ID wins.
6. b. Remember that we’re using RSTP, not STP. If it were STP, it would be distinguished as a “designated port.” RSTP uses “root port.”
7. c. This would be the hello-time.
8. d. The Forwarding state is the only state that actively passes LAN traffic. All of the other states only pass BPDUs.
9. b. IEEE 802.1s, also known as MST.
10. b. Listening is the state that sends BPDUs and listens for them to return on unexpected ports (indicating a loop).

Part Three: Layer 3 Switching (Routing)

The following chapters are included in Part Three:

- “Chapter 10: Routing Basics” starting on page 219
- “Chapter 11: Open Shortest Path First (OSPF)” starting on page 239
- “Chapter 12: Multi Protocol Label Switching (MPLS)” starting on page 261
- “Chapter 13: Border Gateway Protocol (BGP)” starting on page 271
- “Chapter 14: Security, Redundancy and More” starting on page 289



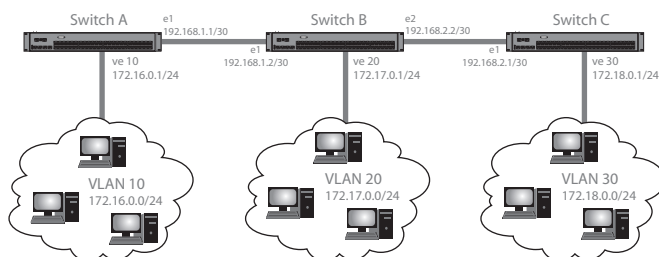
Routing Basics

Welcome to the Layer 3 portion of the book. For the next five chapters, we're going to the Network Layer of Brocade's technologies. We will explore the flexibility of the Layer 3 switch.

To do so, we need to make sure that your switch is running Base Layer 3 or Full Routing code. Few, if any, of the commands listed in the next few chapters will have any application if your switch is not running routing code. Review Chapter 5 if you need to.

Disabling Layer 2

If we're going to use our Brocade switch as a router (or at least part of it), we may find it wise to make that clear. Let's look at an example:



Each switch is housing its own subnet. Each switch is housing its own broadcast domain. In this example, each switch is playing the part of a switch to its locally connected hosts. It's also playing the part of router to connect to the other switches. Switch A is connecting to Switch B. Each switch has one of two addresses in the 192.168.1.0/30 subnet. There are no other hosts in that subnet (nor can there be). The only reason that subnet exists is to route traffic between Switch A and Switch B. Likewise, the connection between Switch B and Switch C is set up the same way, but with the 192.168.2.0/30 subnet.

The point is that, for the 192.168.1.0/30 and 192.168.2.0/30 subnets, we are not at all interested in playing the role of a switch (Layer 2). This function can be turned off, for just such an instance. This way, no Layer 2 information is passed. It is as if the involved interfaces are actually routers.

Let's start with Switch A:

```
BR-SwitchA#conf t
BR-SwitchA(config)#int e 1
BR-SwitchA(config-if-e1000-1)#ip address 192.168.1.1/30
BR-SwitchA(config-if-e1000-1)#route-only
```

Pay close attention to that last command. The **route-only** command tells the switch that you do not want interface e 1 to function as a Layer 2 switch port. Similar configurations are needed for Switch B:

```
BR-SwitchB#conf t
BR-SwitchB(config)#int e 1
BR-SwitchB(config-if-e1000-1)#ip address 192.168.1.2/30
BR-SwitchB(config-if-e1000-1)#route-only
BR-SwitchB(config)#int e 2
BR-SwitchB(config-if-e1000-2)#ip address 192.168.2.2/30
BR-SwitchB(config-if-e1000-2)#route-only
```

And for Switch C:

```
BR-SwitchC(config)#int e 1
BR-SwitchC(config-if-e1000-1)#ip address 192.168.2.1/30
BR-SwitchC(config-if-e1000-1)#route-only
```

The **route-only** command can also be used in the Global config. Use caution. Make sure that this is what you want to do before you do it. VLANs, STP, everything we just spent the last several chapters discussing will be unavailable. Don't get too nervous. If you change your mind, you can always:

```
BR-Switch(config)#no route-only
```

The Routing Table

In Chapter 2, we introduced you to the routing table. This is a very simple table that describes a network (and its accompanying mask), and the address of the gateway that will get traffic bound for that network to its destination. On a Brocade switch, the routing table will also tell you the cost of the route (more on this later), and the *type* of the route (e.g., how it learned about this route).

Let's take a look at Switch A's routing table from the previous example. Note the command that we use:

```
BR-SwitchA#show ip route

Total number of IP routes: 2
Start index: 1  B:BGP D:Connected R:RIP S:Static O:OSPF
*:Candidate default
      Destination NetMask          Gateway      Port Cost   Type
1 192.168.1.0 255.255.255.252 0.0.0.0      1      1      D
2 172.16.0.0 255.255.0.0      0.0.0.0     v10      1      D
```

Let's take a moment to go over what we're looking at.

- **Start Index.** Each routing table entry is numbered. The “start index” tells us what number it will start with. In this case, its 1. There are two routes in this table. Each route is labeled with a number: “1” and “2.”
- **Destination.** This is the network that an incoming packet is heading for. When a router receives the incoming packet, it will check the destination address in the IP header. Then, it will check its routing table to see if it knows how to get to where the packet wants to go. If it can, it forwards the packet on its way. If it can't, the packet is dropped, and the sender receives an ICMP message.
- **NetMask.** This is the subnet mask of the destination network. It's important when you look at the routing table to remember to pair the Destination with the NetMask. There will be times when it appears that there are multiple identical entries in the routing table, but closer inspection will show that there are actually two entries with the same Destination, but different NetMasks.
- **Gateway.** This is the IP address of the router that the incoming packet should be forwarded to. If the router is directly connected to the Destination subnet, the gateway is listed as 0.0.0.0.
- **Port.** This is the interface that is either connected to the Destination subnet, or is connected to the gateway of the Destination subnet.
- **Cost.** This is the Administrative Distance of the route. We'll talk more about this one later.
- **Type.** This explains how the route got in the routing table in the first place. Some types you may see will include:
 - B - the route was learned through BGP
 - D - the route is directly connected to this switch
 - R - the route was learned through RIP
 - S - the route is statically configured on the switch
 - O - the route was learned through OSPF
 - * - the route is a candidate default route

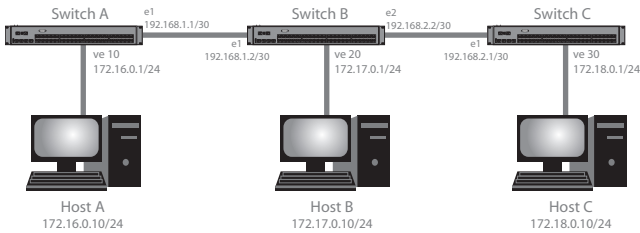
Connected Routes

How did these routes get into Switch A's routing table? Remember Chapter 2? We configured a static IP address on two of the interfaces. On any Layer 3-capable device (including workstations), when you configure one of your interfaces with a static IP address, an entry is automatically made in the device's routing table. When you configure the IP address, you are saying that this device is a participating member of this particular subnet.

In the previous example, we configured interface e 1 on Switch A with an IP address of 192.168.1.1/30. This told the switch that it was a participating member of the 192.168.1.0/30 subnet. Because of that, it automatically added an entry into the routing table detailing that it knows how to get to 192.168.1.0/30. It has an interface that is participating in that subnet. The second interface (ve 10) was configured previously to be 172.16.0.1/16. Its connected route is added to the table as well.

Connected routes constitute the fundamental concepts of a gateway. “You don’t know how to get there, but I’m connected to there. You can get to there through me.” But what happens when your gateway isn’t directly connected to your destination?

Let’s look at our example with hosts involved:



If Host A were to use Switch A as its default gateway, and Host C were to use Switch C as its default gateway, could Host A talk to Host C? Study it carefully.

No, it can't. Switch A doesn't know how to get to 172.18.0.0/16. Likewise, Switch C doesn't know how to get to 172.16.0.0/16. When we issued the **show ip route** command, we noticed that Switch A only has two routes: 192.168.1.0/30 and 172.16.0.0/16. These are the only routes it knows about.

Similarly, if we were to issue a **show ip route** on Switch C, we would see connected routes for 192.168.2.0/30 and 172.18.0.0/16. These are the only routes that Switch C knows about. How can we get Host A and Host C to talk to each other?

Static Routes

One way to do it is with static routes. These are route statements that are manually configured on the switch. You are straight-forwardly explaining to the switch how to get to a certain destination. Static routes are configured at the Global config level. Let's add a static route so Switch A knows how to get to Host C's network (172.18.0.0/16):

```

BR-SwitchA#conf t
BR-SwitchA(config)#ip route 172.18.0.0 255.255.0.0
192.168.1.2
  
```

How was that again? I typed **ip route** followed by the destination network and subnet mask (172.18.0.0 255.255.0.0), and lastly, I added the gateway address, or the router that knows how to get to that network. In this case, 192.168.1.2 is the address of Switch B (the link that directly connects to Switch A). Why did I use Switch B? Why not Switch C? Switch C is the one connected to 172.18.0.0/16. Shouldn't I have used 192.168.2.1 as my gateway instead? Ask yourself this, does Switch A know how to get to 192.168.2.0/30? Go ahead. Look back at the **show ip route** statement a few pages back. It's not in there, is it? Switch A doesn't know how to get to 192.168.2.0/30, so if we used 192.168.2.1 as our gateway address, the route still wouldn't work.

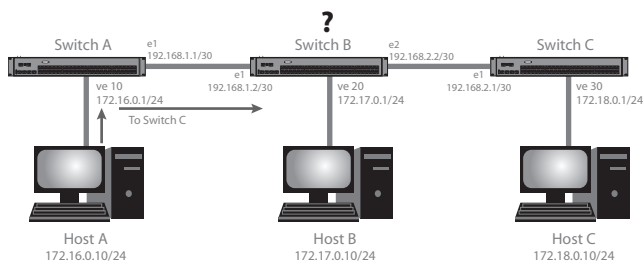
Let's see Switch A's new routing table:

```
BR-SwitchA#show ip route
Total number of IP routes: 3
Start index: 1  B:BGP D:Connected R:RIP S:Static O:OSPF
*:Candidate default
```

	Destination	NetMask	Gateway	Port	Cost	Type
1	192.168.1.0	255.255.255.252	0.0.0.0	1	1	D
2	172.16.0.0	255.255.0.0	0.0.0.0	v10	1	D
3	172.18.0.0	255.255.0.0	192.168.1.2	1	1	S

Look at that! We have a new route. It's the route we added for the 172.18.0.0/16 network. And notice that it has a gateway address (192.168.1.2), instead of being 0.0.0.0. Also notice its type. It's "S" for "Static." It's a static route.

We used Switch B. We told Switch A, "Hey, if anyone wants to get to the 172.18.0.0/16 network, send it to my good friend Switch B at 192.168.1.2." Does Switch A know how to get to 192.168.1.0/30? Sure. It's a directly connected route. Switch A has the address 192.168.1.1. Are we finished? Can Host A talk to Host C yet?



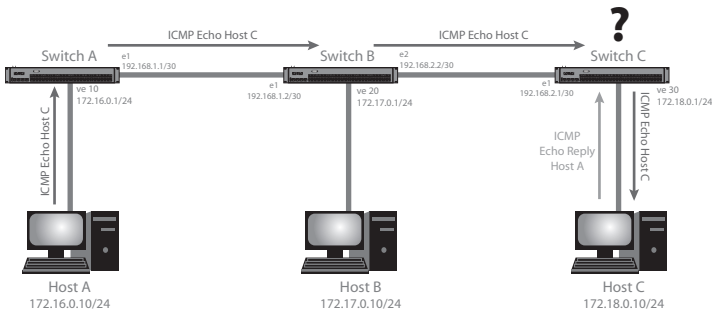
It stopped at Switch B?!? Hmm.... Maybe we should take a look at Switch B's routing table:

```
BR-SwitchB#show ip route
Total number of IP routes: 3
Start index: 1 B:BGP D:Connected R:RIP S:Static O:OSPF
*:Candidate default
      Destination    NetMask          Gateway    Port    Cost    Type
1  192.168.1.0      255.255.255.252  0.0.0.0    1        1        D
2  192.168.2.0      255.255.255.252  0.0.0.0    2        1        D
3  172.17.0.0       255.255.0.0      0.0.0.0    v20       1        D
```

Wow, Switch B has three routes! Did you see that? They're all connected routes. Do you see a route for the 172.18.0.0/16 network? You don't? Hmmm... Are you sure? Rats, you're right. Switch A knows to send 172.18.0.0/16 traffic to Switch B, but Switch B doesn't know how to get to 172.18.0.0/16. Let's fix that.

```
BR-SwitchB#conf t
BR-SwitchB(config)#ip route 172.18.0.0 255.255.0.0
192.168.2.1
```

We used Switch C's address of 192.168.2.1. Does Switch B know how to get to 192.168.2.0/30? Sure. It's a connected route. And we know Switch C knows how to get to 172.18.0.0/16, because that's a connected route. Why don't we have Host A send a ping to Host C?



Hurray! We made it all the way to Host C! What's that? Well, that's true. Host A didn't get a reply to its ping. Looking at the diagram, the reply seems to have gotten stuck at Switch C. Let's take a look at Switch C's routing table:

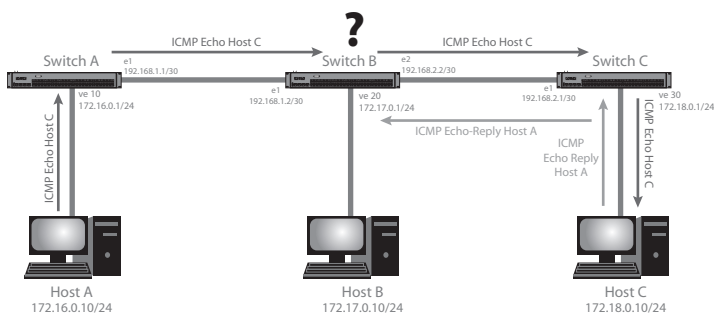
```
BR-SwitchC#show ip route
Total number of IP routes: 2
Start index: 1 B:BGP D:Connected R:RIP S:Static O:OSPF
*:Candidate default
      Destination    NetMask          Gateway    Port    Cost    Type
1  192.168.2.0      255.255.255.252  0.0.0.0    1        1        D
2  172.18.0.0       255.255.0.0      0.0.0.0    v30       1        D
```

Hmm...Well, Switch C definitely knows how to get to 172.18.0.0/16, but it doesn't know how to get back to 172.16.0.0/16. I think we're going to have to tell it.

```
BR-SwitchC#conf t
BR-SwitchC(config)#ip route 172.16.0.0 255.255.0.0
192.168.2.2
```

Notice here again that we used the IP address of Switch B. Switch C doesn't know how to get to 192.168.1.0/30. We chose a gateway we knew Switch C could reach.

That's fixed it, right? Let's send that ping again!



We still didn't get a reply? Nuts! Let's look at the diagram. Hmm... It seems to have stopped at Switch B this time. What was Switch B's routing table again?

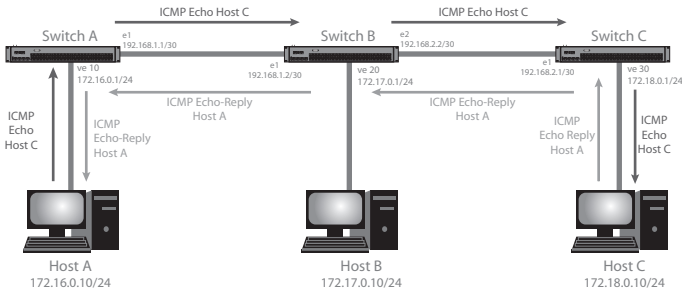
```
BR-SwitchB#show ip route
Total number of IP routes: 4
Start index: 1 B:BGP D:Connected R:RIP S:Static O:OSPF
*:Candidate default
```

	Destination	NetMask	Gateway	Port	Cost	Type
1	192.168.1.0	255.255.255.252	0.0.0.0	1	1	D
2	192.168.2.0	255.255.255.252	0.0.0.0	2	1	D
3	172.17.0.0	255.255.0.0	0.0.0.0	v20	1	D
4	172.18.0.0	255.255.0.0	192.168.2.1	2	1	S

Well, well. It seems Switch B doesn't know how to get to 172.16.0.0/16 either! Whew! This is getting hard. Okay, well, let's tell Switch B how to get there.

```
BR-SwitchB#conf t
BR-SwitchB(config)#ip route 172.16.0.0 255.255.0.0
192.168.1.1
```

Now, will it work? Shall I try another ping from Host A to Host C? I'm afraid to look!



What? It worked?! Woo hoo! Ahem... I mean, of course, it did. Why did I take you through this long painful exercise? Two reasons, really. One, I wanted you to become familiar with how we configure static routes. And two (and perhaps more important), I wanted to show you the fundamental rule for troubleshooting a routing problem: Always remember that a *packet must know how to reach its destination and how to get back*.

Too often, network engineers will focus too much energy on trying to troubleshoot a packet getting to its destination when they may not realize that it is getting to its destination. It just can't get back. If you're dealing with a rare protocol that only works in one-way communication, this concern doesn't really apply. However, 90-95% of all protocols you will troubleshoot require a reply. You've got to go through the path both ways.

Now we're not tied down to connected routes! We can configure static routes to get anywhere we want to go. In fact, we haven't talked about a special static route yet: the default gateway. The *default gateway* (or "gateway of last resort") is a way of telling your router, "Hey, if you need to forward a packet to a network that's not in my routing table, forward it to this router."

The default gateway works the same way as it does on your workstation. Host A, for example, could function just fine in talking to his neighbors in the 172.16.0.0/16 network without a default gateway. The only time Host A would need a default gateway is to talk to a host *outside* of its subnet. Now, could we add static routes to the workstation's routing table (remember that all Layer-3 capable devices have routing tables)? Sure, but oh, the pain! Wouldn't it just be simpler to say, "Hey workstation, if you need to go outside your subnet, hand your packets to this guy." That's the default gateway.

That's great for a workstation, but would a router really need a default gateway? They're actually very common. In fact, if you have broadband Internet access at home, odds are that your broadband router has a default gateway on it. Internet access is one of the more common uses of default gateways these days. In this case, the default gateway would be the IP address of the Internet Service Provider's router. You're trusting your ISP's router to know how to forward your packet to anywhere on the Internet you wish to go.

How do you configure a default gateway on a router?

```
BR-Switch#conf t
BR-Switch(config)#ip route 0.0.0.0 0.0.0.0 192.168.0.1
```

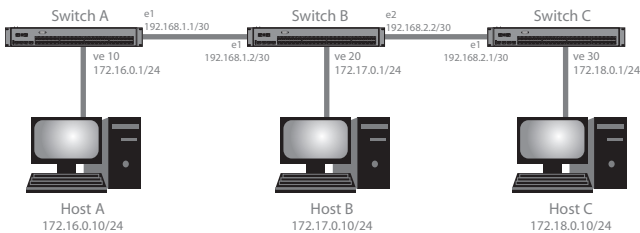
Hey, that looks like a static route! It is. Except the destination network and subnet mask are all zeros. In the router world, that means a default gateway. Does a router have to have a default gateway? No. Switch A, B, and C didn't need one. Not all routers will need a default gateway. It all depends on your infrastructure.

Well, that was a lot of work, but I'm sure glad that's over. Can you imagine configuring static routes like that on...10 routers? How about 50? How about several hundred thousand? Good thing the Internet doesn't use static routes! We'll get into that later. What's the alternative?

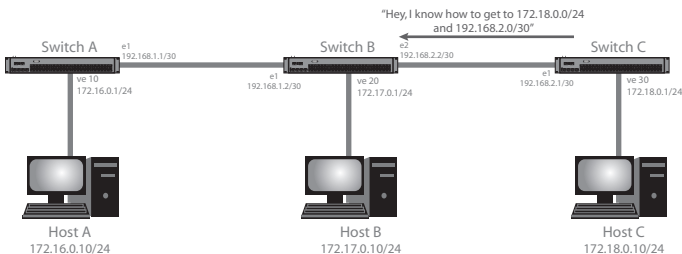
Dynamic Routing Protocols

The whole point of a dynamic routing protocol is to provide a way for routers to share what they know. With static routes, we had to manually configure each of the routes we wanted in each router's routing table. With dynamic routing protocols, we let the routers do the talking.

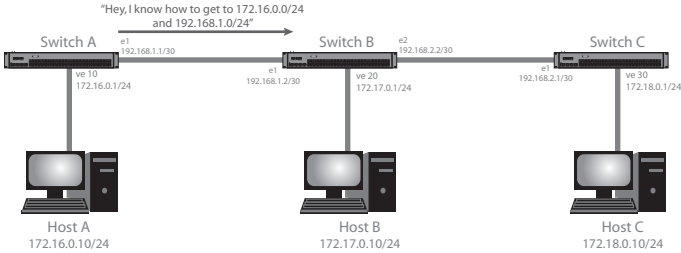
Let's go back to our earlier example. Let's say we hadn't configured the static routes yet.



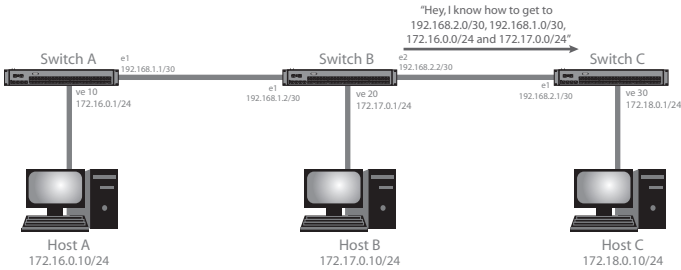
What if, instead of configuring static routes, we simply configured a dynamic routing protocol on each of the switches. What would happen? Well, to start, we might see this:



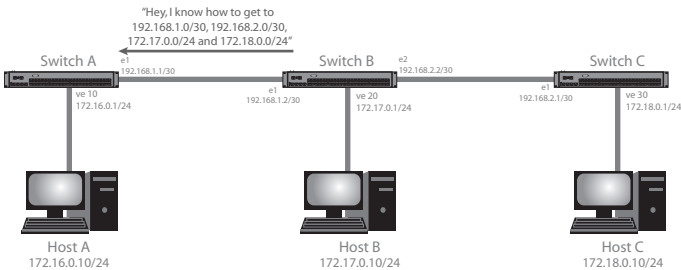
Through the dynamic routing protocol, Switch C tells Switch B that he knows how to get to 172.18.0.0/16 and 192.168.2.0/30. Now Switch B already knows how to get to 192.168.2.0/30 (it's a connected route), but it didn't know how to get to 172.18.0.0/16. The chain continues.



Switch A also talks to Switch B. Switch A says that it knows how to get to 172.16.0.0/16 and 192.168.1.0/30. Again, Switch B already knows how to get to 192.168.1.0/30 (it's a connected route), but it didn't know how to get to 172.16.0.0/16. Switch A and C have been talking, but now it's Switch B's turn.



Switch B tells Switch C that it knows how to get to 192.168.2.0/30 (which Switch C already knew; connected route), 192.168.1.0/30, 172.17.0.0/16, and 172.16.0.0/16 (it's sharing what it learned from Switch A). And the conversations continue.

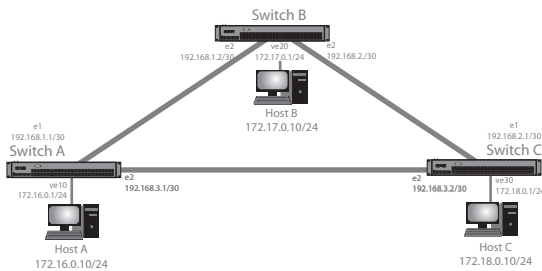


Switch B tells Switch A that it knows how to get to 192.168.1.0/30 (which Switch A knew already), 192.168.2.0/30, 172.17.0.0/16, and 172.18.0.0/16 (sharing what it learned from Switch C).

Now, in a real dynamic routing protocol, the conversations would not normally be this slow or this serial (one right after the other). I broke it up for you to show you how the conversations affect the routers in the environment. In real life, these conversations would be happening almost simultaneously, depending on the dynamic routing protocol you've chosen.

So now, Switch A knows all of Switch B's and Switch C's routes. Switch B knows all of Switch A's and Switch C's routes. And, of course, Switch C knows all of Switch A's and Switch B's routes. If I tried to ping Host C from Host A, would it work? Sure! All of the routers know how to get my packet there and back again. And I didn't have to configure one static route! All I had to do was enable a dynamic routing protocol, and let the routers do the talking.

But it gets better. What happens when there's a change in the network? Let's throw in this little curve ball.



We've added a new subnet (192.168.3.0/30 between Switch A and Switch C), and we've added another link. What happens to our switches' routing table? Not to worry, friends. Did we not say that it was a dynamic routing protocol? Switch A and Switch C detect the change, and inform Switch B (as well as each other) of the new route.

But wait a minute! Don't we have a loop now? For example, if Switch A wants to get to Switch C, it's got two choices. It could go directly to Switch C via the 192.168.3.0/30 network, or it could get there the way it used to get there (through Switch B). Which will it choose?

That's actually a deep question. It actually depends on the dynamic routing protocol you've chosen. We'll go more into that later on. But the important thing for you to know now is that the router will make a decision, and, most often, will stick to that decision unless there's a change in the network.

"So, you're saying that if one of those links goes down, my routers will detect the failure and route traffic through another link automatically?!" Yep, that's exactly what I'm saying. It's a beautiful thing, isn't it? The routers know their options and they will adjust their routing tables to accommodate the change. Some examples of dynamic routing protocols that we'll be discussing include RIP, OSPF, and BGP. Love the dynamic routing protocol!

Administrative Distance

Now, here's a question. You can run several different dynamic routing protocols on the same router and you still have your connected and static routes. When there are duplicate routes for the same network (say, one that's static and one that's learned through a dynamic routing protocol), how does the router choose which one is better?

This problem was tackled a while ago. The answer is that each type of route in the routing table has an administrative distance. The administrative distance is just a number. It's an eight-bit number, so it can be any number from 0 to 255. Think of the administrative distance number as a cost to use the route. As in money, the higher the cost, the lower the desirability. The lower the cost, the better.

What determines the cost, the administrative distance? Refer to this table for details:

Route Type	Administrative Distance
Directly Connected	0
Static Route	1
External BGP	20
OSPF	110
RIP	120
Internal BGP (iBGP)	200

You would do well to memorize that table. It never hurts to know them off the top of your head, especially if you're mixing dynamic routing protocols in your infrastructure. There are more routing protocols than what are listed in the table, but these are the protocols we'll be talking about. Notice that the directly connected routes have an administrative distance of 0. This means that no matter what static routes you may have configured, or what routes your router may have learned from a dynamic routing protocol, the router will *always* choose the connected route (if it has one).

What if your router has a route that it learned from RIP, but you've configured a static route for the same network? The router will choose the static route. It has a lower administrative distance.

What if you router has a route that it learned through RIP, but it also learned the same route through OSPF? The router will choose the OSPF route (110 is a smaller number than 120).

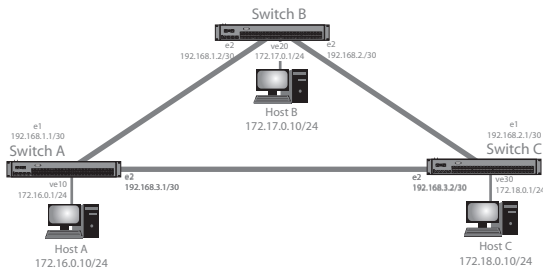
Now, the router won't forget all these duplicate routes. The router will make a decision as to which route it will use, but it will still remember that it has other options. Why? Failures happen. Often. If the decided route were to fail for

whatever reason, the router now has an alternative. Duplicates in your routing table are actually a good thing. They provide redundancy, and redundancy means more up time, and less complaints from your users.

Routing Information Protocol (RIP)

Let's talk about a specific dynamic routing protocol: Routing Information Protocol (RIP). RIP is the fundamental routing protocol. It is one of the oldest dynamic routing protocols. It is very simple in its design. When a router is configured with RIP, it will send its entire routing table out all of its participating interfaces, by default, every 30 seconds. The routing table is contained in packets, but encapsulated using UDP port 520 for delivery. All other routers that are configured with RIP and are within this router's broadcast domain will receive the broadcast, and add the newly discovered routing information into their own table. They, in turn, will broadcast their entire routing table to anyone who will listen.

RIP is referred to as a *distance vector* routing protocol. This term describes how the routing protocol makes its decisions. In other words, what happens when a router learns two (or more) different routes for the same network? How does it choose? In RIP, this depends on *distance*. It measures distance by *hops*. A hop count is how many routers the packet would have to pass through to get to its destination. Let's look at our example again.



If Switch A wanted to talk to Switch C, it's got two choices. One, it could speak to Switch C directly on its 192.168.3.0/30 network link. Or two, it could hand off to Switch B, using its 192.168.1.0/30 link, and let Switch B forward it to Switch C using its 192.168.2.0/30 link. If RIP were employed, which would it choose? The path going from Switch A to Switch C is one hop. The path going from Switch A to Switch B and then to Switch C is two hops. RIP would choose the direct path from Switch A to Switch C. It has the shorter hop count. This is the only characteristic RIP uses to make its routing decision. It is a distance vector protocol.

When RIP was developed, the concept of having an infrastructure so big that your packet would have to travel through more than 15 routers was laughable. We know this today to be quite short-sighted. But this characteristic still exists with RIP today. It can only comprehend routes whose hop counts are from 1 to

15. A hop count of 16 is interpreted as being impossible to reach. Typically, if you find yourself designing an infrastructure containing more than 16 routers, you're not going to want to use RIP anyway. We'll go more into that later.

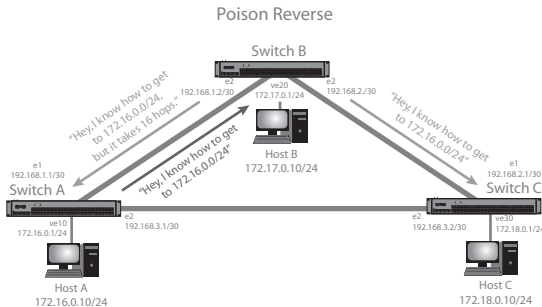
RIP actually utilizes this number to its advantage. For one, if a route goes down, there's no way, in RIP, to tell your neighboring routers to remove a particular route. Even if you don't readvertise it, they'll still have the one you originally advertised in their routing tables. RIP will advertise the route, but will give it a hop count of 16. The routers will update their tables and consider the route unreachable.

Loop Avoidance

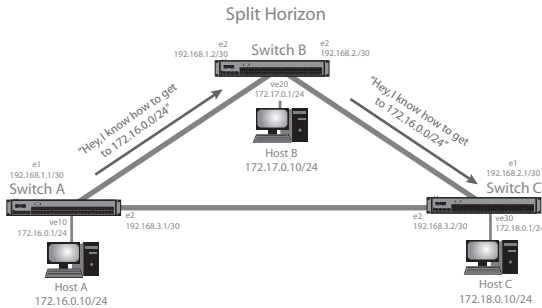
In our example, you can actually see a physical loop. This means that it's theoretically possible for the routers to play an infinite game of “hot potato” with your packets. In other words, Host A could send a packet to Switch A (destined for Host C). Switch A could say, “I know how to get to Host C- through Switch B” and hand it off to Switch B. Switch B says, “I know how to get to Host C- through Switch A” and hand it back to Switch A. Switch A will not remember that it originated the packet. It will simply see a packet coming in destined for Host C, and it will say, “I know how to get to Host C- through Switch B.” And the game will continue forever (or until the upper-layer protocols decide it has taken too long to get there).

This is called a *routing loop*, and it's a bad thing. It doesn't usually abuse your switches like a Layer-2 loop does, but it is still a bugger to troubleshoot. The main problem is that traffic does not get to its destination. It just bounces between two or more routers, like a trapped pinball.

RIP uses two methods to resolve this problem. The first (and default) is called *poison reverse*. Because Switch B will broadcast its entire routing table out all of its routing ports, Switch A might get back some routes that Switch B learned from Switch A. What poison reverse does is keep track of which port a route was learned from. When it advertises its routing table out that port, it changes the routes that it learned from that port to have a hop count of 16 (unreachable).



On Brocade switches, you have the option to turn off poison reverse and use a different method: *split horizon*. Split horizon is very similar. The difference is that instead of advertising the routes as unreachable, it just doesn't advertise them at all. If Switch B were to have learned a route for 172.16.0.0/16 from Switch A, Switch B will not advertise a route for 172.16.0.0/16 back to Switch A. It will to Switch C, but not to A.



RIP version 1 vs. RIP version 2

In 1994, RIP version 2 was introduced, and it is typically the preferred version of RIP to run. There are two main improvements on the original RIP:

- RIPv2 is *classless*. RIPv1 is *classful*.
- RIPv2 provides authentication between routers using a shared password hashed by MD5

Discussion of the authentication goes beyond the scope of this book, but let's take some time to discuss the classless versus classful. This is a big deal, and the principle reason why RIPv2 is preferred.

Flip back to Chapter 2. Don't worry, I'll wait. Classful means that you don't pay attention to the subnet mask. You simply rely on the Class system to determine your mask. For example, if the address's first octet is, say, 10 (the full address being, say, 10.1.1.3), a classful protocol will assume a Class A mask (255.0.0.0). It doesn't understand the concept of subdividing beyond the Class.

Classless means we've got full rein to subdivide however we see fit. The idea is that RIPv1 is Classful. It doesn't send subnet mask information with the routes. It just sends the network address and the receiving router assumes a default subnet mask. RIPv2 includes the network address and the subnet mask information. It is Classless. It is not bound to use only the default subnet masks. It can use whatever the engineer configured.

In our examples with Switch A, B, and C, this isn't too dangerous a circumstance. You should know, that the 192.168.x.x routes will be advertised as full Class Cs instead of /30s. This means that they'll be advertised as 192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24. This is okay for now, but if we want to use, say, some subnets like 10.1.1.0/24, 10.2.2.0/24, and

10.3.3.0/24. All three of these subnets will be advertised as the same subnet under RIPv1 (10.0.0.0/8). For this to function the way we'd want it to, we would need to make sure we're using RIPv2.

With all of this hype about RIPv2, is there really any reason to use RIPv1? Only one, really: legacy systems. In other words, let's say you're maintaining a network infrastructure that contains routers incapable of using RIPv2. You have no choice but to use RIPv1 (and plan your networks carefully). This is becoming less and less of an issue, as RIPv2 has been out for over a decade. Still, you may run into this circumstance. Your other option is, of course, to replace the legacy router with something more modern that can run RIPv2.

Configuring RIP

Configuring RIP is actually very, very easy. There are really only two steps:

1. Enable RIP globally on the router
2. Define each interface you want participating in RIP (and specify which version they will use)

To enable RIP globally, you go to the Global config:

```
BR-Switch#conf t
BR-Switch(config)#router rip
```

There are additional options that you can define in the RIP subconfig, but all you need to do is start with this command. Next, you specify an interface that has an IP address. This can be a real interface or a ve.

```
BR-Switch(config)#int e 3
BR-Switch(config-if-e1000-3)#ip rip v2-only
```

Other options include "ip rip v1-only" and "ip rip v1-compatible-v2." In our command above, we've defined this interface to use RIPv2. It is possible to have one interface using RIPv2 and another interface using RIPv1 on the same router.

For loop avoidance, remember that poison reverse is enabled by default. If you wish to use split horizon instead, you simply disable poison reverse:

```
BR-Switch(config)#int e 3
BR-Switch(config-if-e1000-3)#no ip rip poison-reverse
```

You do not have the option to disable loop avoidance entirely. If you disable poison reverse, split horizon takes over automatically.

Something that is often forgotten when dealing with dynamic routing protocols is that the switch now keeps two routing tables. One is its actual routing table (seen by using **show ip route**). Once you've enabled RIP, the router will actually keep a RIP routing table. This table will show all of the routes learned from RIP, and it will even show duplicate route entries. When RIP has made the decision as to which of the duplicate routes is best, it will feed that route to the actual routing table.

To see RIP's routing table and any additional global configuration settings:

```
BR-Switch#show ip rip
```

Finally, with that in mind, remember this as well. RIP will only advertise directly connected routes and routes from the RIP routing table. By default, RIP will not advertise static routes, or routes learned from any other routing protocol. If you want RIP to advertise routes that it has learned from other protocols (including static routes), you need to set up route redistribution. This is very easy to set up.

```
BR-Switch#conf t
BR-Switch(config)#router rip
BR-Switch(config-rip-router)#redistribution
```

RIP will now advertise static routes and routes that your router has learned from other protocols.

The “Null” Route

There's a funny little trick that's included in Brocade switches. You have the ability to forward packets to, well, nowhere. Why in the world would you want to do this? Well, let's say, you're administering a Corporate LAN. You've discovered that quite a few of the company's employees have been wasting their time going to a particular web site. The boss wants you to put a stop to this.

Now, you could use a firewall, but some firewalls only protect traffic coming in from the outside. In this case, you want to stop traffic from the inside going out. Implementing a proxy server is also a popular solution, but you have neither the time nor the budget. Instead, you can add a static route to your router. You tell it to forward the network range for that web site to “null0.” Null0 is a special interface that tells the router to simply drop the packet. It is configured like this:

```
BR-Switch#conf t
BR-Switch(config)#ip route 169.254.213.0/24 null0
```

Notice that the only gateway I specified is “null0.” Now, imagine if you were using route redistribution, and you wanted to quickly make sure that no one on any of your routers could go to this network. One static route to “null0” on one of your routers is all it takes. Thanks to dynamic routing protocols and route redistribution, in a short time, this null route will be in all of your routers' routing tables.

Summary

- You can disable Layer 2 functions on a switch that is running router code by using the **route-only** command
- **route-only** can be used on individual interfaces or globally
- A router stores all of the routes it knows about in a routing table
- You can use the **show ip route** command to see the contents of the routing table

- For routed communication to be successful, routers along the path must know how to get to the destination and how to get back
- Connected routes are added to the routing table whenever an interface is configured with an IP address
- Static routes manually tell the router how to get to certain networks it may not know about
- Default gateway is a route that can be used when a router doesn't know how to get to a particular destination
- Dynamic routing protocols allow routers to share their routing tables automatically
- Administrative distance is used to decide which type of route (directly connected, static, RIP, OSPF, etc.) is preferred
- RIP is a distance vector routing protocol
- RIP broadcasts its entire routing table out all of its participating interfaces every 30 seconds, by default
- RIP uses “hop count” (the number of routers away) to decide which route is better
- RIP can only count up to 15 hops; 16 hops is considered unreachable
- RIP avoids routing loops using poison reverse (by default) or split horizon
- RIPv1 is Classful
- RIPv2 is Classless
- RIP keeps a separate routing table that contains only routes learned through RIP
- Use route redistribution to have RIP advertise static routes or routes learned by other routing protocols
- To configure RIP, you need only enable it globally, and specify each interface that will participate
- A “null” static route can be used to prevent users from reaching certain networks

Chapter Review Questions

1. What command is used to disable Layer 2 functions?
 - a. ip route
 - b. route-only
 - c. ip route-only
 - d. router rip

2. What is the administrative distance for RIP?
 - a. 100
 - b. 1
 - c. 200
 - d. 120
3. How would you configure your switch's default gateway to be 10.1.1.1?
 - a. `ip route 0.0.0.0 0.0.0.0 10.1.1.1`
 - b. `ip route default 10.1.1.1`
 - c. `route 0.0.0.0 0.0.0.0 10.1.1.1`
 - d. `ip default gateway 10.1.1.1`
4. By default, what method does RIP use to avoid routing loops?
 - a. Split horizon
 - b. "Null" routes
 - c. Redistribution
 - d. Poison reverse
5. What command would display a switch's routing table?
 - a. `show routing-table`
 - b. `show ip route`
 - c. `show ip rip`
 - d. `show arp`
6. What command is used to enable RIP globally on the switch?
 - a. `rip`
 - b. `ip rip`
 - c. `router rip`
 - d. `rip router`
7. What command(s) will configure a connected route?
 - a. `router rip`
 - b. `int e 8`
`ip address 192.168.1.0/24`
 - c. `ip route connected 192.168.1.0/24 e 8`
 - d. `ip route 192.168.1.0/24 connected e 8`

8. What is the administrative distance for a static route?
 - a. 0
 - b. 110
 - c. 1
 - d. 120
9. What will RIPv1 advertise 172.16.3.0/24 as?
 - a. 172.16.0.0
 - b. 172.16.3.0/24
 - c. 172.0.0.0
 - d. 128.0.0.0
10. In RIP, how many hop counts are considered unreachable?
 - a. 15
 - b. 16
 - c. 256
 - d. 0

Answers to Review Questions

1. b. The **route-only** command is used at either the global or the interface level to disable Layer 2 functions.
2. d. The default administrative distance for RIP is 120.
3. a. We would have also accepted **ip route 0.0.0.0/0 10.1.1.1**
4. d. Poison reverse is enabled by default. If it is disabled, split horizon will be enabled.
5. b. **show ip route** is correct.
6. c. **router rip** in the Global config will enable RIP. You then need to specify interfaces that will participate.
7. b. Nasty question. You don't configure a connected route, per se. It is created when you assign an IP address to an interface.
8. c. The default administrative distance for a static route is 1.
9. a. RIPv1 is Classful. The first octet of the address is 172. This falls in the Class B realm. Class B's default subnet mask is /16, or 255.255.0.0. This includes the first two octets for the network address, but assumes the last two octets are for hosts.
10. b. RIP can only count up to 15 hops. A route that is advertised as 16 hops is considered unreachable.

Open Shortest Path First (OSPF)

We talked about RIP's short-comings in the previous chapter. The routing community needed something more robust. In October of 1989, RFC 1131 was released and a new routing protocol was born. It was called Open Shortest Path First (OSPF), and it used *Dijkstra's algorithm* to calculate the shortest path between points. Modern implementations (including Brocade's) are based on OSPFv2, as defined in RFC 2328.

Before we go on, I'd like to make one point clear. OSPF is a very robust and complex routing protocol. Entire books have been written on OSPF alone. We will only be scratching the surface, giving you a sound foundational understanding of the protocol, and how it can serve you.

With that, let's compare OSPF to the methods and protocols we've discussed in the previous chapter.

Link State vs. Distance Vector (OSPF vs. RIP)

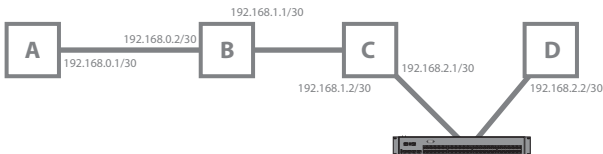
Let's dive right into the differences:

- **Metric.** OSPF is a *link state* protocol. This means that each router participating in the protocol is responsible for reporting the state of each of its links. OSPF uses a link's cost to make its routing decisions. The cost is determined by dividing the speed of the link (in Mbps) into 100, by default (this method can be customized; more on this later). Just as in consumer pricing, the lower the cost, the more desirable the route. *Distance vector*, as you may recall from Chapter 9, means that the protocol bases its routing decisions on distance. If you have two or more routes to the same destination, you choose the one with the shortest distance. RIP is the most common example of this type of protocol. For its distance, it uses *hop count*, or the number of routers a packet must traverse to get to its destination.
- **Route Advertisements.** RIP advertises its entire routing table. OSPF sends only updates.

- Route Advertisement Recipients.** RIP broadcasts its entire routing table to any address within its configured broadcast domains. It doesn't matter if you're a router or a workstation. You will receive a copy of the router's routing table. In OSPF, it sends updates *multicast*. A multicast packet is a packet that originates from a single host, but is delivered to multiple destinations. It might help to think of broadcast versus multicast in the same way you understand an Ethernet hub versus a switch. An IP broadcast, in some ways, is like a hub. With a hub, if a frame comes in one port, it gets duplicated and sent out every other port. There's no intelligence to it. It just gets duplicated. An IP multicast is more like a switch with VLANs configured. An incoming frame will be duplicated, but only sent out the port of the frame's destination, or, if that is unknown, the ports that are participating in the VLAN. In this case, OSPF sends its updates to all OSPF routers. Two special multicast addresses to remember are:
 - 224.0.0.2 - all routers
 - 224.0.0.5 - all OSPF routers
- Updates.** RIP broadcasts its entire routing table every 30 seconds. OSPF sends updates only when changes occur.
- Authentication.** MD5 hash authentication is available in RIP, but only in version 2. OSPF also provides MD5 hash authentication.
- Variable Length Subnet Masks (VLSM).** RIP version 2 supports this. OSPF supports it, too.

Getting to Know Your Neighbors

No, I haven't misplaced a chapter from *Social Graces for Nerds*. OSPF defines a *neighbor* as a router that has an interface with an IP address in the same broadcast domain. Sounds simple, right? No? Well, maybe a picture would help.



Router A has one neighbor: Router B. Router B has two neighbors: Router A and Router C. Router C has two neighbors: Router B and Router D. Router D has one neighbor: Router C. The important thing to note here is not only where there are neighbors, but where there are not neighbors. Router A and Router C are not neighbors. They do not share an interface in the same broadcast domain. They can reach each other, but only through Router B. Likewise, Router B and Router D are not neighbors (much less Router A and Router D).

How do we get to know our neighbors? Well, similar to real life, we start by saying, "Hello."

Hello Protocol

When a router (and an interface) is configured for OSPF, the router will start to send Hello packets out its OSPF-configured interfaces. The packets are not a broadcast, but a multicast. Specifically, the Hello packets are sent to 224.0.0.5. This is a special multicast address registered specifically for OSPF. If a message is sent to this address, all reachable OSPF routers will receive it.

OSPF sends Hello packets for a couple of reasons. For one, that's an easy way to discover who your neighbors are. When your router sends a Hello packet, your neighbors are now aware of you. When the neighbors send their Hello packets, you are now aware of them.

The Hello packet is also used to ensure two-way communication between you and your neighbor. It is also used as a keepalive, so that the router is aware of when his neighbor may have become unavailable. It is also what is used to elect a Designated Router.

Designated Router (DR)

"A what?" you ask. When configuring OSPF using a broadcast multi-access medium (such as Ethernet), OSPF will designate two special routers: a Designated Router (DR) and a Backup Designated Router (BDR). In OSPF, the participating routers need to synchronize their routing tables, right? Well, they could synchronize to every router they know about, and likewise, every other router could synchronize to every router they know about. This might work okay if you've only got two or three routers, but what if you have a few hundred? That's a lot of traffic!

OSPF has, instead, instigated a Designated Router. This is a router that will receive all the updates on a given segment. The Backup Designated Router is the second-in-command. This router receives the updates, too, but doesn't send them back out. When a router needs to update other routers of a change, it will send the update to the Designated Router and the Backup Designated Router. The Designated Router will then send the update to all of the other routers on its segment.

When the Designated Router becomes unavailable, the Backup Designated Router instantly becomes the Designated Router, and an election is held to see who becomes the next Backup Designated Router.

This way, each router need not be aware of all of the OSPF routers involved. It needs only to communicate its updates (and receive updates) from a single source. This minimizes traffic, and allows the OSPF design to expand nicely, as needed.

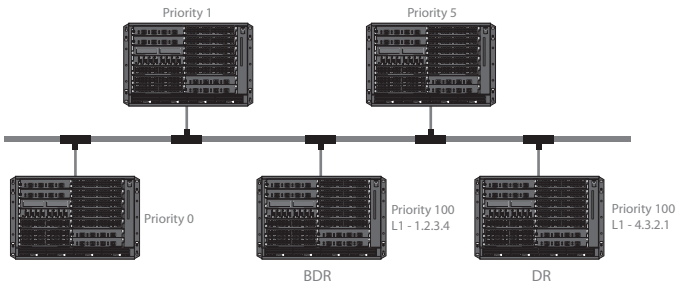
The Designated Router Election

Now that we know what a Designated Router is, how is it elected? When Hello packets are sent, they contain information about the router who sent it. Specifically, what is being watched for is a router's priority.

The priority is an eight-bit number (0-255) that can be configured on each router. By default, this value is 1. If you configure the value to 0, that tells the router (and its neighbors) that it will never take part in DR/BDR elections. The main reason to do this is if you have a router whose resources (CPU, RAM, etc.) are limited. Being a DR or a BDR takes a fair amount of processing and RAM. If you've got a very old router, you may want to make sure it never takes on that role. To do so, you would configure its priority to 0.

The router with the highest priority wins the election, and becomes the Designated Router. The router with the second highest priority becomes BDR.

What if no priorities are configured? All the routers will have the same priority (default value is 1). The decision will then be made by looking at the router's Router ID. The Router ID is the IP address of the lowest-numbered loopback interface. If there are no loopback interfaces configured on the router, the Router ID is the IP address of the lowest-numbered interface. When you have two or more routers with the same highest priority, the decision is made by which of those has the highest Router ID. The second-highest Router ID would become BDR.

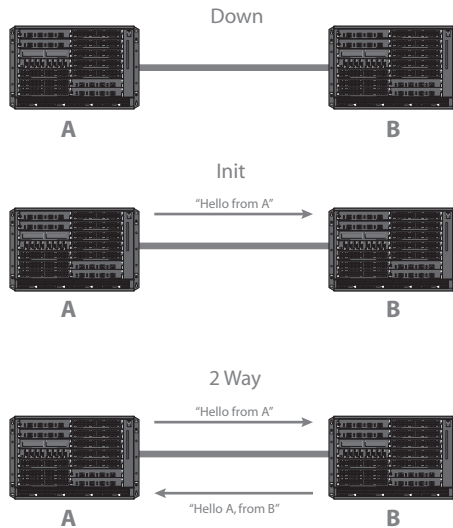


Neighbor States

The neighbors on a given router will go through six states on their way to fully becoming synchronized with its routing table. The first three states are referred to as the *neighboring process*. These are the states neighbors go through in order to establish themselves as neighbors.

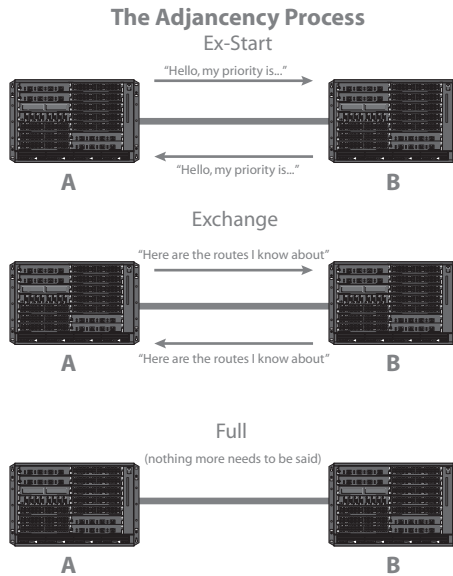
- **Down.** The router has not sent a Hello packet, nor has it received one
- **Init.** A Hello packet has been sent to a neighbor, but no Hello packet has been received from that neighbor
- **2 Way.** The router has sent a Hello packet to a neighbor, and has received a Hello packet back; the reply will contain your router's Router ID inside; now, your router knows that the neighbor knows your router as well

The Neighboring Process



The next three states are referred to as the *adjacency process*. To establish adjacency means that your OSPF database is synchronized with your neighbor (there's currently no further information to share).

- **Ex-Start.** This is short for “Exchange Start”; this is where the DR/BDR election takes place; routers are sending Hello packets to determine who will become the DR and BDR
- **Exchange.** Now the neighbors are finally talking; in this state, your router is sharing everything it knows, and the neighbor router is sharing everything it knows
- **Full.** We're adjacent! The neighbors have no more information to share; they've shared it all



Areas

The routers in OSPF are divided into groups called areas. This is a similar concept to the broadcast domain in Layer 2. To subdivide routers into areas reduces traffic, but it also reduces CPU load for the routers. In OSPF, there is always at least one area.

Like broadcast domains, areas are defined by the engineer. If you're only dealing with four or five routers, you may find that it may not make any sense to subdivide them further, and simply make them all a member of one area. On the other hand, if you're dealing with 20 routers, you may find a benefit to breaking them up a bit.

Areas can be organized any way you see fit. Some engineers have organized routers by city (e.g., routers in Chicago in one area, routers in New York in another area, etc.). Some engineers have organized routers on different floors of the same building to be in different areas. It all depends on your need.

When you create an area, you assign it a number. This number is 32 bits, so it can be anything from 0 to 4,294,967,295. Few, if any, organizations need to define 4 billion areas, but this gives you a great deal of freedom in your numbering system. Given the size, it is not uncommon to give an area number that looks like an IP address (e.g., area 1.2.3.4). An IP address is a 32-bit number as well. This can be confusing, so for this chapter, we'll stick to decimal numbers. Rest assured that many an IT department has established their OSPF area as Area 51 (Don't get the reference? Pat yourself on the back. You're not as far gone as you thought you were).

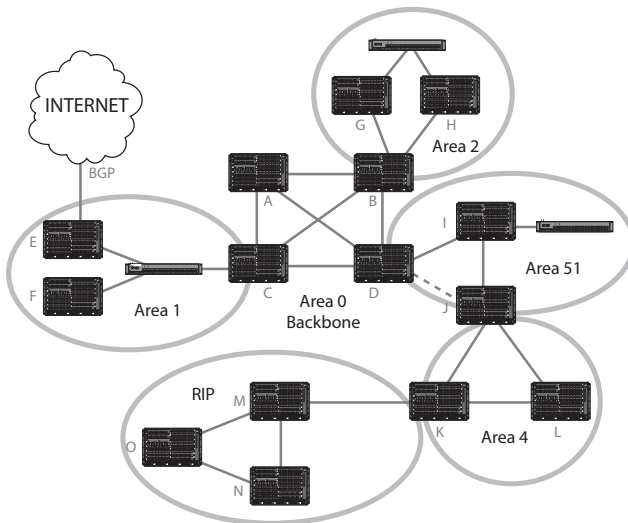
Area 0 - The Backbone

As I mentioned earlier, there must be at least one area, and that area is Area 0. Every OSPF implementation must have an Area 0. This is the backbone. All inter-area communication travels through the backbone. Every area must be connected to Area 0.

It is not uncommon for an OSPF configuration to only have Area 0. This certainly would be suitable for smaller networks. For larger networks (or for smaller networks that need room to grow), Area 0 should be central, redundant, and high-bandwidth. After all, all of your other areas are going to communicate to each other through it.

Area Border Router

This is a router that has one interface in one OSPF area, and another interface in a different OSPF area. A single router can be a member of several different areas (one area per interface). An ABR may be the DR or BDR for its area, but it certainly doesn't have to be. The ABR's job is simply to share its area's routes with other areas that it's connected to. An ABR will typically always have one interface in Area 0 (there's one exception that we'll talk about later).



Routers A, B, C, and D all have interfaces in Area 0. Routers B, C, and D have interfaces in other areas. They are Area Border Routers. Router A is what we would call an Internal Router (IR). It's participating only in its own area. Some might still refer to Router A as a Backbone Router, as it is in Area 0. Router J is also an ABR, but we'll talk more about that a little later. That one's a special case.

Autonomous Systems (AS)

In OSPF, an Autonomous System (AS) is a group of routers that operate under the same routing protocol. An implementation of OSPF, whether it has one area or a thousand areas, constitutes one Autonomous System. A group of routers that are using RIP constitute a different Autonomous System. A group of routers using BGP constitute yet another Autonomous System. You get the idea. The routing protocol being used defines the Autonomous System.

Why is this important? Well, there are not too many network infrastructures that run one, and only one, routing protocol. Especially in larger businesses, it is very common to come across connected systems that use OSPF, BGP, RIP, and others. OSPF designates these as individual Autonomous Systems. The foreign Autonomous Systems (e.g., BGP, RIP, etc.) can share their routes with OSPF, and OSPF can share its routes with the foreign Autonomous Systems. But to do so, we need to define the Autonomous Systems boundary.

Autonomous System Boundary Router (ASBR)

An Autonomous System Boundary Router (ASBR) is a router that participates in more than one routing protocol. This is a router that may have one interface participating in OSPF, and another interface participating in RIP, BGP, or some other protocol. Look back at our area example. Router K (in Area 4) is an ASBR. It has one interface in Area 4 participating in OSPF, and it has another interface participating in RIP. Router E in Area 1 is also an ASBR. It has one interface participating in OSPF Area 1, and another interface participating in BGP to the Internet.

Like an ABR, the ASBR need not be a DR or a BDR. Its role is defined simply because it is participating in one or more routing protocols in addition to OSPF. Because of this, it has a responsibility of sharing, or not sharing, routes between the routing protocols. This is also referred to as route redistribution.

Getting RIP and OSPF To Talk To Each Other

Just because you see a route in your switch's routing table does not mean that your routing protocol is going to advertise it. OSPF will advertise connected routes (of interfaces that are participating in OSPF) and routes that it learns from other neighbors, but that's it. It will not advertise routes learned from RIP, BGP, etc., nor will it advertise static routes. It can, but you have to tell it explicitly to do so.

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#redistribution rip
```

This command will take all of the routes that the router learned from RIP, and include them in the advertisements to OSPF routers.

This still doesn't account for RIP learning routes from OSPF. This is done with the following command:

```
BR-Switch#conf t
BR-Switch(config)#router rip
BR-Switch(config-rip-router)#redistribution
```

RIP isn't specific. This command essentially says, "Hey, collect all of the routes learned by any other routing protocol, and add them to my list to advertise to my RIP buddies."

What else? What if you want to include other connected routes of interfaces that are not participating in OSPF? You will need this command:

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#redistribution connected
```

And what about static routes? Let's say you want to include those as well? You should be able to guess by now, but in case you can't:

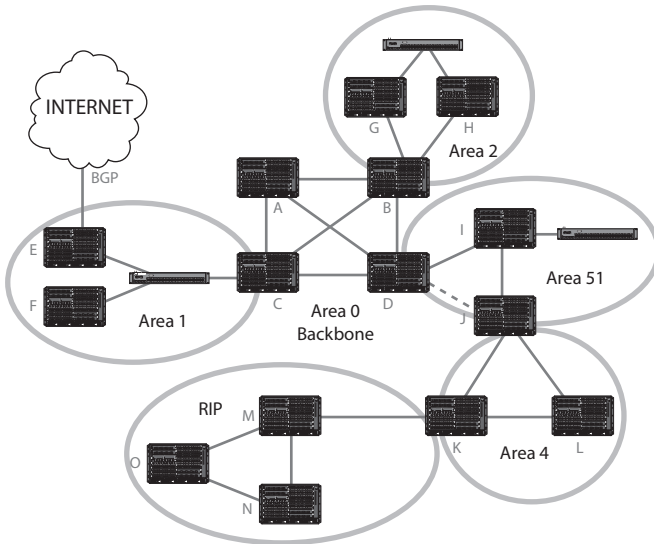
```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#redistribution static
```

An ASBR may also summarize the routes that it knows about to make it easier on the OSPF routers its advertising to. After all, if you don't need to clutter their route tables with the details, why would you? Let's say you had an ASBR that knew routes for 10.1.0.0/18, 10.1.64.0/18, 10.1.128.0/18, and 10.1.192.0/18. Instead of advertising all four of these routes, it could summarize them into one (after all, the OSPF routers can only get there through the ASBR):

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#summary-address 10.1.0.0/16
```

Special Areas

Now that you know what areas and autonomous systems are, we need to talk about a few specially defined areas. Let's look at our area example again:



Stub Area

A stub area is an area that doesn't get fed routes from outside the autonomous system. In our example, Area 2 is a Stub Area. It's not that Area 2 doesn't want to use the Internet. It's not that Area 2 doesn't want to know how to get to the RIP domain. It doesn't need to know the details. It just needs to a default or summarized route to send it where it needs to go. In the example of the Internet, Area 0 merely needs to advertise a default gateway to Area 2. It's got to travel through Area 0 anyway.

Routes coming from outside the autonomous system are branded as such. If there's an area that really only has one way into the infrastructure (e.g., Area 2 through Router B), it really doesn't need to know anything else other than to send the traffic to Area 0 (and let Area 0 handle it). This is a Stub Area.

Not So Stubby Area (NSSA)

Yes, that's really the name of this next area. This is an area that can receive routes from outside the autonomous system, and advertise them to Area 0, but it can't receive other routes from outside the autonomous system from Area 0. Look back at our image. Look at Area 4. This is an example of an NSSA. It has Router K, which is an ASBR, because it has one interface participating in RIP. This area can receive the routes from RIP, and it can pass these routes back to Area 0. It will not accept Internet routes from Area 1. It doesn't need them.

A NSSA will typically have an ASBR in its area (like our example). The idea is that it probably won't be the only ASBR in the autonomous system (again, like our example). This is a way that you can say, "I'll accept routes from ASBRs that are in my area, but not routes from any ASBRs that are not in my area."

Totally Stubby Area

The names keep getting sillier, don't they? We don't have an example for this one in our image. A Totally Stubby Area is an area that won't accept routes from another autonomous system (like a stub area), but it also won't even accept routes from other areas. You would typically only configure a Totally Stubby Area if there was only one link connecting this area to the backbone, and it was a router that was not very powerful. In this situation, the only route that would be advertised would be a default gateway to Area 0.

This is similar to a Stub Area, but let me rehash the difference again. A Stub Area will accept routes from within the autonomous system (e.g., routes to other areas advertised from Area 0). A Stub Area will not accept routes from outside the autonomous system. A Totally Stubby Area will not accept routes from within the autonomous system or without the autonomous system. Nothing other than a simple default gateway to its ABR in Area 0 is accepted.

Link State Advertisement (LSA)

A Link State Advertisement (LSA) is a packet that is sent between OSPF routers. They come in six types (that we'll talk about; there are actually 11, currently). LSAs are the packets that contain routing information that is passed from router to router. Different types travel to and from different routers.

- **Type 1 (Router LSA).** These are sent by every router participating in OSPF; they list all of the router's links (and, thus, the connected routes for all interfaces participating in OSPF); these are only sent to routers within their respective area
- **Type 2 (Network LSA).** These are sent by the DR only; these packets list all of the routers on the same broadcast multi-access segment; if you think of the DR as the king, this advertisement is listing his loyal subjects
- **Type 3 (Summary LSA).** These are generated by ABRs only; they summarize all of the routes within their respective areas, and advertise them to Area 0; Area 0, in turn, advertises these summarized routes to other areas; Totally Stubby Areas would receive only a default gateway
- **Type 4 (ASBR-Summary LSA).** These are generated by ASBRs only; for those areas that need nothing more than a summarized gateway to routes outside the autonomous system, a Type 4 LSA is sent; a Stub Area or NSSA would receive these; a Totally Stubby Area would not
- **Type 5 (External LSA).** These are the non-summarized routes generated by ASBRs; these are advertised to Area 0, and Area 0 will, in turn, advertise them to the rest of the areas (except for Stub, NSSA, and Totally Stubby areas)

- **Type 7 (NSSA).** These are sent by an ASBR in an NSSA to the ABR within their respective area; the ABR will then translate the Type 7 to a Type 5, and advertise it to Area 0 (which will propagate it from there); because any other area would see this as a Type 5, Stub Areas, Totally Stubby Areas, and even other NSSAs would not see these LSAs

Configuring OSPF

For as complicated as the OSPF protocol is, configuring it is surprisingly easy. The first thing you have to do is enable it globally:

```
BR-Switch#conf t
BR-Switch(config)#router ospf
```

Next, you have to specify the areas that the router will be participating in. You can list as many areas as the router will be participating in, but for this example, I'll just put it in Area 0:

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#area 0
```

Finally, you must configure the router interface with the OSPF area that its participating in. Say, for example, that interface e 5 had an IP address configured on it, and you want it participating in Area 0:

```
BR-Switch#conf t
BR-Switch(config)#int e 5
BR-Switch(config-if-e1000-5)#ip ospf area 0
```

As soon as that last command is issued, OSPF is fully enabled, and Hello packets will be sent out interface "e 5" to start the neighboring process. You could also define the router's priority. This is done at the interface config:

```
BR-Switch#conf t
BR-Switch(config)#int e 5
BR-Switch(config-if-e1000-5)#ip ospf priority 100
```

What if you want an interface to be part of OSPF, but you don't want it to send or receive routing updates? This is what is called a passive interface. It is configured like this:

```
BR-Switch#conf t
BR-Switch(config)#int e 5
BR-Switch(config-if-e1000-5)#ip ospf passive
```

Loopback Interfaces

It is often very wise to configure a loopback interface on the router that is participating in OSPF. Why? Well, the loopback address doesn't belong to a particular interface. If you have a router participating in OSPF with several physical links, the router will have to be identified by the IP address of one of those links (the Router ID). What if that link goes down? You have to start the neighboring process all over again using a new Router ID.

If you use a loopback address, the Router ID doesn't change. It's also certainly not contingent on whether one of the physical interfaces is up or down. The loopback is the IP address of the device, not an individual interface.

Common practice would be to choose a unique /30 network (some even use a /32 network), assign an IP address to a loopback interface, and assign that interface to be a member of a particular area.

```
BR-Switch#conf t
BR-Switch(config)#int lo 1
BR-Switch(config-if-lbif-1)#ip address 192.168.100.1/30
BR-Switch(config-if-lbif-1)#ip ospf area 0
```

Now a single interface can only be a member of one OSPF area. What do you do if you're configuring an ABR? Simple, create another loopback interface. Most switches will allow up to four.

Cost

Let's talk a little bit more about cost. For any given link, OSPF advertises the cost of the link. The cost is 100 divided by the bandwidth of the link (in Mbps). A 10 Mbps link would have a cost of 10 ($100 \div 10 = 10$). A DS3 (45 Mbps) line would have a cost of 3 ($100 \div 45 = 2.222$; this will be "rounded" up to 3). A 100 Mbps link would have a cost of 1 ($100 \div 100 = 1$). As you can see, with this system, a higher capacity link has a lower cost, and therefore a higher desirability.

But what about a gigabit link? It would also have a cost of 1 ($100 \div 1,000 = 0.1$; this will be rounded up to 1). Now we have a problem. If our network is all Ethernet, all of the links (100 Mb or more) will have the same cost! We would want our gigabit and 10-gigabit links to be more highly favored over our 100 Mb links. There are two ways to take care of this.

One way is to manually configure the cost. This is done at the interface level. Instead of letting OSPF calculate the cost, assign one yourself:

```
BR-Switch#conf t
BR-Switch(config)#int e 5
BR-Switch(config-if-e1000-5)#ip ospf cost 20
```

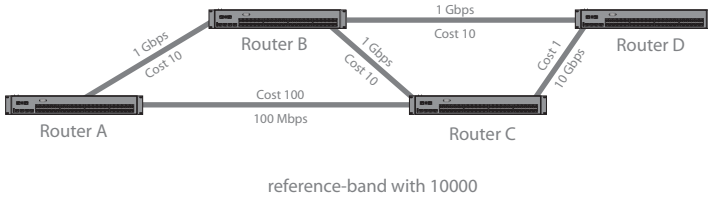
We chose a cost of 20 here, but you can choose any 24-bit number (1-16,777,215). The lower the cost, the more desirable the link. Engineers will often assign a manual cost to make a link more (or less) desirable, even if there's a higher bandwidth alternative. It just depends on your infrastructure.

That may be okay if you've just got a couple of links to distinguish, but what if you have a lot? Wouldn't it be easier to just tell OSPF to use a larger number than 100 to calculate cost?

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#auto-cost reference-bandwidth
10000
```

Notice that this was configured in the global OSPF config. Now, it will divide the bandwidth in Mbps into 10,000. This means that a 10 Mbps link will now cost 1,000; a 100 Mbps link will cost 100; a gigabit link will cost 10; and a 10-Gigabit link will cost 1. The number that you use for your reference bandwidth (in this example 10,000) can be any number between 1 and 4,294,967.

Now that we know how to adjust the costs, how do the costs play into our routing decisions? Consider the following image.



Router A has two paths to get to Router C. The most obvious path (visually) would be the directly connected path (the 100 Mbps line). This has a cost of 100. However, if Router A were to go through to Router B (cost of 10) and from B to Router C (cost of 10 again), it would be a total cost of 20. This cost would be considerably less than 100 (the cost of its direct connection to C). It would choose the path through Router B.

Now, what about from Router A to Router D? There are more paths it could take. Router A to B (cost of 10) to D (cost of 10) would be a total cost of 20. Router A to B (cost of 10) to C (cost of 10) to D (cost of 1) would be a total cost of 21. Router A to C (cost of 100) to D (cost of 1) would be a total cost of 101. Which route will Router A prefer? The path from A to B to D (total cost of 20). That's the cheapest route.

Adjusting OSPF Timers

By default, all OSPF interfaces will send a Hello packet to their neighbors every 10 seconds. This is defined as a 16-bit number, so any number of seconds from 1-65,535 may be used. I don't recommend you delay the Hello interval too long, as this is a means for the router to be aware that his neighbor may be down. This setting is customized to the individual interface. Here's an example in which we change this interface's Hello interval to be 20 seconds, instead of the default 10:

```

BR-Switch#conf t
BR-Switch(config)#int e 5
BR-Switch(config-if-e1000-5)#ip ospf hello-interval 20

```

The Router Dead Interval is the amount of time a router will wait before declaring a neighbor dead. By default, if an interface does not receive a Hello packet from its neighbor in 40 seconds, it declares that neighbor down. This is configured on the individual interface. This is also a 16-bit (0-65,535) number (in seconds).

Here's an example to change the interval to 60 seconds:

```
BR-Switch#conf t
BR-Switch(config)#int e 5
BR-Switch(config-if-e1000-5)#ip ospf dead-interval 60
```

The Retransmit Interval defines how long a router will wait to retransmit an LSA to a neighbor. By default, this is five seconds, and can be a number from 0 to 3,600 (an hour). Again, this is configured on the interface:

```
BR-Switch#conf t
BR-Switch(config)#int e 5
BR-Switch(config-if-e1000-5)#ip ospf retransmit-interval 10
```

The Transit Delay defines the amount of time it takes to transmit a Link State Update on a particular link. By default, this is one second. It can be a number from 0 to 3,600 (an hour). This is also configured on the interface:

```
BR-Switch#conf t
BR-Switch(config)#int e 5
BR-Switch(config-if-e1000-5)#ip ospf transit-delay 5
```

And finally, there are two SPF timers that are configured on the same line. The first is the SPF Delay timer. When the router receives a routing update, it will wait for five seconds (by default) to start its SPF calculation. If this timer is set to 0, the router will immediately start its SPF calculation on receiving a change. The second timer is the SPF Hold Time. This is the amount of time the router will wait after calculating a new SPF before trying to calculate another SPF. By default, this is 10 seconds. Both of these timers may be configured lower for a quicker response from the router (when a change occurs), but this could also cause a heavier processing load. Your mileage may vary. These timers are configured in the OSPF config mode:

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#timers spf 10 20
```

Advertising a Default Route

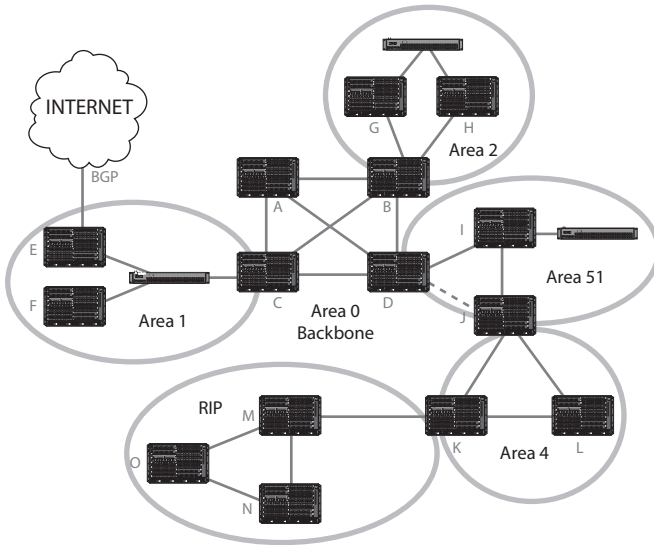
We know that we can redistribute static routes into OSPF with the **redistribute static** command. Unfortunately, the default gateway is treated specially. It's entered just like any other static route, but it is not included in this command. This is the command you need:

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#default-information-originate
```

Now, if you have a default gateway configured on your router, it will be advertised to its OSPF neighbors.

Configuring Stubby Areas

We talked about three kinds of Stubby Areas: Stub Area, Not So Stubby Area (NSSA), and Totally Stubby Area. Let's look at our image again:



Configuring Stub Areas

Area 2 is an example of Stub Area. To configure it as a Stub Area, all three of the routers in this area (Router B, Router G, and Router H) will need to define their area as Stub. This is done in the global OSPF configuration of each router in the area:

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#area 2 stub 10
```

This defines Area 2 as Stub. What's the 10? When you define an area as Stub, OSPF no longer shares all of the cost information for the myriad of different routes outside of the Area. Remember, a Stub area is intended to simply use a default gateway to get outside of the Area. The 10 is an assigned cost for entering or leaving the Area. I chose 10, but I could have easily chosen 5 or 1 or 65,000, but when you define an area as Stub, you must define the cost for leaving or entering the area.

Configuring Not So Stubby Areas (NSSA)

Area 4 is an example of a Not So Stubby Area. We have Router K as an ASBR, and we need to feed the routes it receives to Area 0, but we don't want any other routers from outside the AS. This needs to be defined on every router in the area (Routers J, K, and L).

Again, this is defined in the global OSPF config:

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#area 4 nssa 5
```

The number at the end (“5”) defines the cost for entering or leaving the area. My choice of 5 was completely arbitrary.

Configuring Totally Stubby Areas

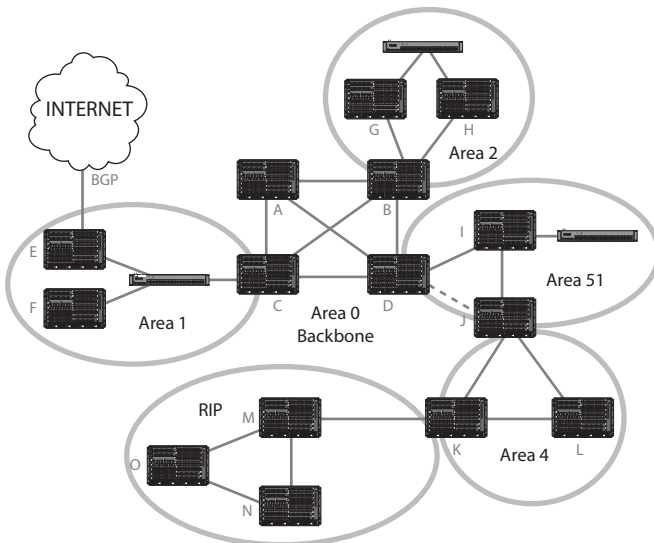
Defining a Totally Stubby Area is very similar to defining a Stub Area. We just need to add one important keyword at the end:

```
BR-Switch#conf t
BR-Switch(config)#router ospf
BR-Switch(config-ospf-router)#area 6 stub 100 no-summary
```

When you define a Stub Area, you know that no Type 5 LSAs (External) will be advertised to the area. When you define a Totally Stubby Area, no Type 5, Type 4, or Type 3 LSAs (with the exception of a default gateway) will be advertised to the area. This is what that “no-summary” keyword will do.

Configuring a Virtual Link

Remember how we said that every area must be connected to Area 0? What if that's physically impossible? Let's look at our areas again:



Area 4 does not physically connect to Area 0. It can't. Router J only has one link, and that's to Router I. Router I is not the ABR for Area 0. Router D is. All is not lost. There is a reason that dashed line is there. To conform to the rule, we establish a Virtual Link.

A Virtual Link acts as a tunnel to pass information from one ABR to another ABR that is connected to Area 0. The non-backbone area acts as a *transit area* to get to Area 0. There are two rules for configuring Virtual Links:

1. Virtual Links are only configured on ABRs; they must be configured on both of the ABRs involved
2. The transit area (the area that the Virtual Link is created in) must have full routing information (e.g., it must not be a Stub, Totally Stubby, or NSSA)

To configure our Virtual Link, we need to go to the global OSPF config:

```
BR-RouterJ#conf t
BR-RouterJ(config)#router ospf
BR-RouterJ(config-ospf-router)#area 4 virtual-link 1.2.3.4
```

In this example, Router J is configuring the Virtual Link and using the Router ID address of Router D. We also need to configure Router D:

```
BR-RouterD#conf t
BR-RouterD(config)#router ospf
BR-RouterD(config-ospf-router)#area 4 virtual-link 2.3.4.5
```

In this case, Router D is using the Router ID of Router J. And now we have our Virtual Link.

I have one final thought on Virtual Links. They are meant to be a solution when your network is physically restricted. In other words, if you have no possibility of directly connecting an area to your backbone, a Virtual Link is your last resort. It's good to know that it's there. Use it when you need to, but don't plan to use it. It can make troubleshooting very difficult, and it adds a layer of complexity to your infrastructure. Again, use it if you need it, but don't if you don't.

OSPF show Commands

Here are some commands that will prove very helpful in making sure your OSPF environment functions smoothly:

- `show ip route` — This isn't an OSPF exclusive command per se, but it does show you your routing table; remember that this will contain routes learned from all sources (connected, static, RIP, etc.)
- `show ip ospf route` — Displays only routes learned through OSPF
- `show ip ospf database` — Displays the OSPF link state database
- `show ip ospf area` — Displays information on all of the defined areas
- `show ip ospf neighbor` — Displays information on all of your OSPF neighbors; this one is helpful for checking the state of your neighbors; remember, FULL is good
- `show ip ospf interface` — Displays area ID and adjacency information for each interface; it gives similar, but much more in-depth information, to `show ip ospf neighbor`

- `show ip ospf virtual-link` — Displays information about configured Virtual Links; make sure that you see “ptr 2 ptr”; if you don't, your Virtual Link is not active
- `show ip ospf trap` — Displays the state of all OSPF-related SNMP traps defined
- `show ip ospf border` — Displays information about all ABRs and ASBRs
- `show ip ospf config` — This command is often overlooked, but it's awfully handy if you've got a switch with a very large config; this will show you just the section of the config pertaining to OSPF

Summary

- OSPF is a link state protocol
 - It uses a link's state and cost to make its routing decisions
 - It uses Dijkstra's algorithm to calculate the Shortest Path
 - After adjacency is reached, routers only send updates as they occur
 - It uses multicast to send its messages
- Neighbors are routers that have interfaces that share the same broadcast domain
- OSPF routers use Hello packets to negotiate a conversation and to verify health
- The Designated Router (DR) and Backup Designated Router (BDR) receive all updates for a given broadcast multi-access segment
- The Designated Router (DR) forwards updates to the rest of the routers on its segment
- The Designated Router and Backup Designated Router are chosen through an election
 - The router with the highest priority becomes Designated Router
 - The router with the second highest priority becomes Backup Designated Router
 - Should the Designated Router fail, the Backup Designated Router will immediately become the Designated Router, and an election will be held to see who becomes the Backup Designated Router
- Areas subdivide and group different OSPF routers
- All OSPF configurations must have at least one area: Area 0 - the Backbone
- All areas must physically connect to Area 0
- All inter-area traffic must travel through Area 0

- A router who has interfaces in two or more areas is an Area Border Router (ABR)
- An Autonomous System encompasses all routers participating in a single routing protocol
- A router who is participating in more than one routing protocol is an Autonomous System Boundary Router (ASBR)
- A Stub Area is an area that receives no routes from outside the Autonomous System
- A Not So Stubby Area may receive routes outside of the Autonomous System from an ASBR in its own area, but it will not receive external routes from Area 0
- A Totally Stubby Area receives no routes from outside the Autonomous System, nor does it receive routes from outside its own area (other than a default gateway)
- Link State Advertisements are used to send information about the state of each of its links, as well as routing information
- A cost is assigned to each link in the OSPF path; the total cost of the path is assessed; the lower the cost the more desirable the route
- When a physical connection to Area 0 is impossible, a Virtual Link may be established

Chapter Review Questions

1. In a Designated Router election, what happens in the event of a tie (two routers with the same priority)?
 - a. One router will be chosen round-robin; the other becomes BDR
 - b. The router with the highest Router ID becomes DR; the other becomes BDR
 - c. The router with the lowest loopback address becomes DR; the other becomes BDR
 - d. The router with the lowest cost becomes DR; the other becomes BDR
2. A Totally Stubby Area will not receive:
 - a. Type 3 (Summary) LSAs (except a default gateway)
 - b. Type 4 (ASBR-Summary) LSAs
 - c. Type 5 (External) LSAs
 - d. All of the above

3. By default, a link's cost is calculated by:
 - a. 100 divided by the maximum bandwidth of the link in Mbps
 - b. 10,000 divided by the maximum bandwidth of the link in Mbps
 - c. 100 divided by the maximum bandwidth of the link in bits per second
 - d. 10,000 divided by the maximum bandwidth of the link in bits per second
4. Which area is referred to as the Backbone?
 - a. Area 1
 - b. Area 0
 - c. Area 10
 - d. Area 51
5. By default, how often will an OSPF router send Hello packets to its neighbor?
 - a. 2 seconds
 - b. 4 seconds
 - c. 10 seconds
 - d. 30 seconds
6. Which router states describe the Adjacency Process?
 - a. Ex-Start, Exchange, Full
 - b. 2 Way, Ex-Start, Exchange
 - c. Down, Init, 2 Way
 - d. Init, Exchange, Full
7. Which describes a Not So Stubby Area (NSSA)?
 - a. This area will not receive routes outside of the Autonomous System
 - b. This area will not receive routes outside of its own area
 - c. This area will not receive routes outside of the Autonomous System from Area 0
 - d. A and B
8. An ASBR is:
 - a. A router that participates in more than one routing protocol
 - b. A router with interfaces participating in more than one area
 - c. A router with the highest priority
 - d. A router with the highest Router ID

9. Which OSPF show command will tell you the states of your neighboring routers?
 - a. show ip ospf database
 - b. show ip ospf route
 - c. show ip ospf border
 - d. show ip ospf neighbor
10. An ABR is:
 - a. A router that participates in more than one routing protocol
 - b. A router with interfaces participating in more than one area
 - c. A router with the highest priority
 - d. A router with the highest Router ID

Answers to Review Questions

1. b. The router with the highest Router ID becomes DR; the second highest Router ID becomes BDR. The Router ID is either the IP address of the lowest loopback interface, or the IP address of the lowest physical interface (if no loopback is configured).
2. d. It will not receive any of these (other than a default gateway).
3. a. You can change this, but by default, it is 100 divided by the maximum bandwidth in Mbps.
4. b. Area 0 is the Backbone. Area 51 is for those wishing to amuse themselves.
5. c. By default, Hello packets are sent every 10 seconds.
6. a. The last three neighbor states constitute the Adjacency Process. Answer C describes the Neighboring Process.
7. c. An NSSA may receive external routes from an ASBR in its own area, but it will not receive external routes advertised from the backbone (thus having originated from an ASBR other than the one in its own area).
8. a. An Autonomous System Boundary Router is a router participating in more than one Autonomous System (AS or routing protocol).
9. d. **show ip ospf neighbor** or **show ip ospf interface** will give you this information.
10. b. An Area Border Router is a router that participates in more than one OSPF area.

Multi Protocol Label Switching (MPLS)

Now we are ready to learn about Multi Protocol Label Switching (MPLS) and how it can be used in your network.

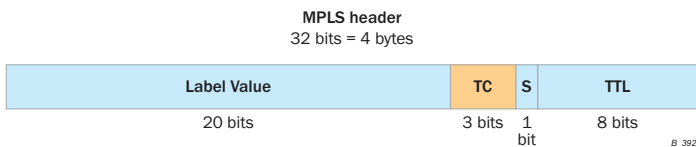
What is MPLS?

Let's take a look at MPLS. MPLS is a highly scalable, protocol agnostic, data-carrying mechanism. In an MPLS network, data packets are assigned labels, and the packet-forwarding decisions are based solely on the contents of this label, without the need to examine the packet itself. This allows users to create end-to-end circuits across any type of transport medium, using any protocol. The MPLS label lookup and label switching is faster than the Routing Information Base as it is performed in the switch fabric and not the CPU.

MPLS Concepts

How does it work? MPLS works by prefixing packets with an MPLS header, containing one or more "labels." This is called a *label stack* and each label stack entry contains the following four fields:

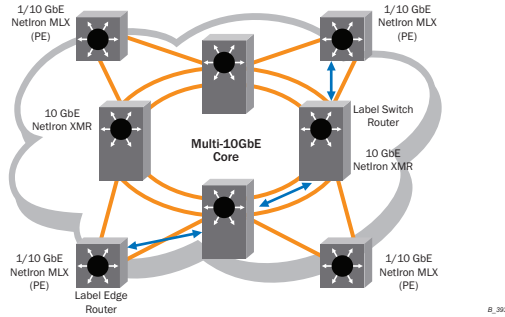
- A 20-bit label value
- A 3-bit Class of Service and Explicit Congestion Notification (ECN)
- A 1-bit end of stack
- A 8-bit time to live field



At the edge of the MPLS network, packets are labeled when entering the network and labels are removed when exiting the network. The labeling of the packet as it enters the MPLS network is called a *push* and the removal of the label as it exists the network is called a *pop*. This is performed by the Label

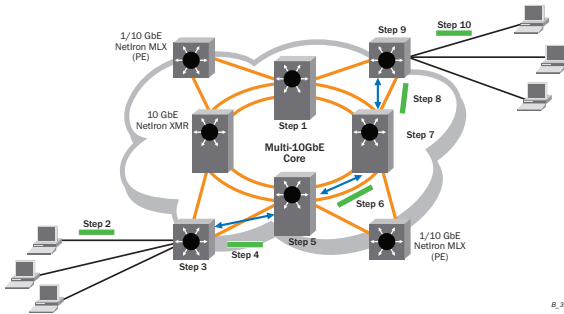
Edge Router (LER). Routers that perform the routing based only on the label are called Label Switch Routers (LSR). The labels are distributed between LERs and the LSRs using the Label Distribution Protocol (LDP). The Label Switch Routers in MPLS networks will regularly exchange label and reachability information with each other to obtain a *virtual map* of the MPLS network. This virtual map is what the LSRs will use to forward their packets appropriately.

Let's take a look at how it works:



Label Switch Paths (LSPs) are established by network operators. LSPs are traffic engineered circuits/paths created to route traffic along a specific path through the network.

MPLS Operations



Here's how MPLS carries traffic:

Step 1. A routing protocol (OSPF) must be established in the Layer 3 topology. The LDP establishes label values for each device according to the routing topology, to pre-configure maps to the destination points creating LSP

Step 2. A packet from the customer's network is sent the ingress of the LER

Step 3. The ingress of the LER receives the packet, performs Layer 3 "value added" services (e.g., QoS, Bandwidth management, etc.) and labels the packet based on the information in the forwarding tables (*push operation*)

Step 4. The labeled packet is forwarded to the LSR via the LSP

Step 5. The LSR reads the label of the packet on the ingress interface, and based on the label, sends the packet out the appropriate egress interface with a new label (*swap operation*)

Step 6. The labeled packet is forwarded to the next LSR via the LSP

Step 7. The LSR reads the label of the packet on the ingress interface, and based on the label, sends the packet out the appropriate egress interface with a new label (*swap operation*)

Step 8. The labeled packet is now forwarded to the LER

Step 9. The egress LER strips (*pop operation*) the label and sends the packet to its destination

Step 10. The packet is received on the customer's network

Now that you understand how MPLS carries traffic. Here are some additional, helpful tips:

- In a swap operation, the label is swapped with a new label and the packet is forwarded along the path associated with the new label
- In a push operation, the label is pushed on top of the packet or existing label
- In a pop operation, the label is removed from the packet

Configuring MPLS Basics on Brocade

How do you configure MPLS? Well, let's take a look at some basic configuration examples. MPLS requires a Layer 3 topology to operate. Brocade utilizes OSPF to provide routing in the MPLS cloud.

On the LER (Provider Edge Router) and LSR (Provider Core Router) enter the following to enable OSPF:

```
BR-Switch(config)# router ospf
  BR-Switch(config)# area (area number)
BR-Switch(config)# route-only
BR-Switch(config)# interface (intf)
  BR-Switch(config)# ip address (address/mask)
  BR-Switch(config)# ip ospf area (area number)
  Note that the "area" value should be the same as the "area"
  value in router OSPF.
  BR-Switch(config)# enable
BR-Switch(config)# write memory
```

To validate the configuration for OSPF, enter the following:

```
BR-Switch# show IP OSPF neighbor
```

All OSPF neighbor relationships should be fully adjacent.

Now, to add MPLS to the network, enter the following:

```
BR-Switch(config)# router MPLS
BR-Switch(config)# mpls-interface (intf number)
BR-Switch(config-interface #/#)# ldp-enable
BR-Switch(config)# write memory
```

Only the interfaces for the Provider and Provider Edge are given the **ldp-enable** command. *Do not* enable MPLS or LDP on any Provider Edge to Customer Edge links.

To validate the configuration for MPLS, enter the following:

```
BR-Switch# show mpls ldp neighbor
```

All MPLS neighbor relationships should be shown and connected.

```
BR-Switch# show mpls ldp session detail
```

This command should provide complete details on the MPLS LDP connections, indicating if the path is established with its corresponding IP address.

```
BR-Switch# show mpls ldp path
```

This command should provide details on the LDP paths created in the MPLS label forwarding table.

If the steps are completed correctly, basic MPLS is up and running.

MPLS Virtual Leased Lines (VLL)

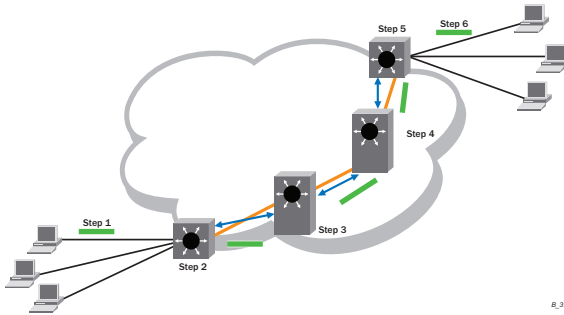
What is VLL? VLL is also known as Pseudo Wire Emulation (PWE). MPLS VLL is a method of providing point-to-point Ethernet/VLAN connectivity over an MPLS domain. The VLL is considered to be a Layer 2 VPN circuit across a single Layer 3 backbone.

How Virtual Leased Lines work

Packets from the Customer edge device A are routed to the Provider Edge LER. The LER assigns the packet to a RSVP-signaled LSP whose endpoint is the destination LER that is connected to the customer edge device B. The destination LER is known as the VLL Peer for the LER connected to Customer device A.

The RSVP-signaled LSP used to reach the VLL-peer is known as the tunnel LSP. For the packet to arrive at the other LER, two labels are used, the *inner* VC label and the *outer* tunnel label. The inner VC label is used for determining what happens to the packet once it reaches its VLL peer. The outer label is used for forwarding the packet through the MPLS domain. Note that the LDP tunnel is unidirectional and will require VLL peers to be built in both directions.

If Class of Service is set for the VLL, the Brocade device selects a tunnel LSP that also has this CoS value. If no tunnel LSP with this CoS value is available then the Brocade device selects a tunnel LSP with the highest CoS value.



B_395

Here's how it works:

Step 1. Packet flows from Customer edge to Provider Edge

Step 2. Packet is received by LER and is assigned to a RSVP-signaled LSP that transverse to the destination LER (VLL peer). Also the inner VC label and the outer tunnel label are added.

Step 3. The MPLS packet is received by the LSR where the outer tunnel label is removed and replaced with a new outer label prior to forwarding to next LSR (consider a *Penultimate hop*)

Step 4. The Penultimate Hop LSR receives the MPLS packet where the outer tunnel label is removed and forwards the packet to LER

Step 5. The LER (VLL Peer) receives the packet and inspects the VC label. On the VLL Peer, the VC label is mapped to an endpoint for the VLL. The endpoint of the VLL specifies what happens to the packet when it exits the VLL. Note that if the port is tagged, the VLAN ID is attached to the packet.

Step 6. Inner VC label removed and sent to the customer edge.

Note that two VLL peers advertise VC labels to each other using the LDP, alternatively static local and remote VC labels can be configured on both VLL Peers.

Configuring VLL on Brocade

Now you'll learn how to configure VLL. To configure VLL, enter the following:

```
BR-Switch(config)# IP router-id (IP address)
Note this IP address is usually the loopback
BR-Switch(config)# router mpls
BR-Switch(config)# lsp (tunnel name) to (destination IP
address)
Note this IP address is the loopback of the destination LER.
BR-Switch(config)# enable
BR-Switch(config)# vll (vll name) (vll id)
BR-Switch(config)# vll-peer (IP address)
Note the IP address is the loopback of the destination
LER.
BR-Switch(config)# untagged (intf number)
Note that both sides must be configured.
```

To validate the configuration for VLL, enter the following:

```
BR-Switch# show mpls vll
```

All VLL tunnels should be listed and their corresponding condition (up, down, etc.).

```
BR-Switch# show mpls VLL detail
```

This command should provide complete details on the VLL tunnel, including the *local* and *remote* peers.

MPLS Virtual Private LAN Segment (VPLS)

Curious about VPLS? VPLS enhances the point-to-point (VLL) connectivity by specifying a method for Virtual Circuits (VC) to provide point-to-multipoint connectivity in the MPLS domain. VPLS traffic flows between remote sites similar to connectivity on a Layer 2 switch. VPLS works similar to VLL using MAC forwarding (Layer 2). The following steps indicate how forwarding flows:

1. The Customer Edge router forwards the packet to the Provider Edge
2. The Provider Edge finds the VC LSP associated with the MAC and pushes an inner VC label and outer tunnel label into the packet
3. The packet is label-switched along the tunnel based on the outer label, until it reaches the LER provider Edge peer
4. The remote Provider Edge looks up the MAC for the VPLS and switches it to the Customer edge
5. If the Provider Edge device does not find the MAC in the VPLS MAC Database, it floods the packet to all remote Provider Edges and the remote Provider Edges flood to all their configured Customer Edge routers

Provider Edge devices perform local and remote VLAN tag translation.

Configuring VPLS on Brocade

To configure VPLS on Brocade, enter the following:

```
BR-Switch(config)# IP router-id (IP address)
```

Note this IP address is usually the loopback.

```
BR-Switch(config)# router mpls
```

```
BR-Switch(config)# lsp (tunnel name) to (destination IP address)
```

Note this IP address is the loopback of the destination LER.

```
BR-Switch(config)# enable
```

```
BR-Switch(config)# vpls (vpls name) (vpls id)
```

```
BR-Switch(config)# vpls peer (vpls peer) (vpls peer) (vpls peer)
```

Note the vpls peer(s) are the destination Peer(s) loopback IP addresses.

```
BR-Switch(config)# vlan (VLAN Number)
```

```
BR-Switch(config)# untagged (intf number)
```

Note all Peers must be configured.

To validate the configuration for VPLS, enter the following:

```
BR-Switch# show mpls VPLS detail
```

This command should provide complete details on the VPLS Peers and their condition (up, down, etc.).

MPLS Layer 3 VPNs

Let's wrap up MPLS with Layer 3 VPNs. With Layer 3 VPNs, the service provider participates in the customer's Layer 3 routing. The customer's CE router at each of their sites speaks a routing protocol, such as BGP or OSPF, to the provider's PE router, and the IP prefixes advertised at each customer site are carried across the provider's MPLS network.

In the MPLS VPN backbone, the label stack is used to forward VPN packets. As the packet leaves the Customer Edge router, it is forwarded to the Provider Edge Router (LER). The LER pushes two tags onto the packet. The inner label used for the VPN, informs the Peer LER to send the packet to an outgoing interface or a Virtual Routing and Forwarding Table (VRF). The outer label is the LDP label that guarantees that the packet will reach the peer LER. If the second label points to a VRF, the LER will perform a label lookup to find the target VRF and then perform an IP Lookup within the VRF. The packet is then routed to the customer edge router. Note that penultimate hop popping can be performed in frame based MPLS. Penultimate hop popping removes the outer label at the last LSR prior to the LER. This allows the LER to perform a faster and simpler lookup.

As stated above, the LER provides the inner and outer label, but for the inner label to provide details of the VPN, it must know every VPN attached to the peer LER. This is accomplished by the peer LER providing the inner label to the LER that contains a summarized VPN route table to every attached customer edge router. The mechanism used for the peer LER to provide this inner label to LER is Multi Protocol - Border Gateway Protocol (MP-BGP). Every MP-BGP update carries a label assigned by the peer LER together with the 96-bit VPNv4 prefix. MPLS VPN packet forwarding requires that the LER specified as the BGP next hop in the incoming BGP update is the same LER as the one that assigned the second label to the label stack.

The customer edge router connects to the LER via a routing protocol (RIPv2, OSPF, External BGP and static routes). Note that the LER must provide connectivity for multiple VPN customers. In order to support multiple VPN customers, the LER uses VRFs. A VRF is the routing and forwarding instance for a set of sites with identical connectivity requirements. The following is associated with VRFs:

- IP Routing Tables
- Routing Protocol Contexts
- Interfaces that use the VRF
- Route Distinguisher
- Route Targets

Since a LER can only support one routing process, routing contexts must be used. The routing contexts create *virtual routing instances* (IP routing tables), allowing for multiple routing instances. Each VRF (note that a VRF is not a VPN, only the routing table) is associated with a routing context.

Now that we have the routing tables separated for each of the customers on the LER, we still have the problem of possible overlapping of MPLS VPN address. To overcome this problem, Route Distinguisher is used. Route Distinguisher (RD) converts the non-unique IP address into a unique VPN-IPv4 addresses. To determine which VPN the customer is associated and provide this information to other Peer LERs, a *route target* is used.

A route-target extended community, or route target, is a type of BGP extended community that you use to define VPN membership. The route target appears in a field in the update messages associated with VPN-IPv4. Route-target *import* lists and route-target *export* lists are created for each VRF. The route targets that are placed in a route target export list are attached to every route advertised to other LERs. When a LER receives a route from another LER, it compares the route targets attached to each route against the route-target import list defined for each of its VRFs. If any route target attached to a route matches the import list for a VRF, then the route is imported to that VRF. If no route target matches the import list, then the route is rejected for that VRF.

Configuring MPLS Layer 3 VPNs

Here's an example for configuring MPLS Layer 3 VPNs. This configuration example makes the assumption that it is CE to PE is OSPF and that the MPLS core (LSR) are configured.

On the LER (Provider Edge Router), enter the following to enable OSPF:

```
BR-Switch(config)#ip vrf (vrf name)
BR-Switch(config)# rd (asn:nn)
BR-Switch(config)# route-target both (asn:nn)
BR-Switch(config)# router ospf
BR-Switch(config)# area (area number)
BR-Switch(config)#route-only
BR-Switch(config)# interface (intf)
BR-Switch(config)# ip vrf forwarding (vrf name)
BR-Switch(config)# ip address (address/mask)
BR-Switch(config)# ip ospf area (area number)
Note the AREA value should be the same as the AREA value in
router OSPF.
BR-Switch(config)# enable
BR-Switch(config)# router ospf vrf (vrf name)
BR-Switch(config)# area (area number)
BR-Switch(config)# area (area number) sham-link (source)
(destination) (cost)
BR-Switch(config)# redistribute bgp
BR-Switch(config)# router bgp
BR-Switch(config)# local-as (as number)
BR-Switch(config)# neighbor (IP address) remote-as (as
number)
BR-Switch(config)# neighbor (IP address) update-source
loopback 1
BR-Switch(config)# address-family vpnv4 unicast
BR-Switch(config)# neighbor (IP address) activate
BR-Switch(config)# address-family ipv4 unicast vrf (vrf
ame)
BR-Switch(config)# redistribute ospf
BR-Switch(config)# router mpls
BR-Switch(config)# mpls-interface (intf)
BR-Switch(config)# ldp-enable
BR-Switch(config)# write memory
```

To validate the configuration for the VRF, enter the following:

```
BR-Switch# show ip route vrf (vrf name)
```

The response should show the Customer Edge router loopback.

```
BR-Switch# show ip bgp neighbor
```

The connection state should established if configured correctly.

Summary

- In an MPLS network, data packets are assigned labels, and the packet-forwarding decisions are based solely on the contents of this label.
- MPLS works by prefixing packets with an MPLS header, containing one or more “labels.” Each label stack entry contains the following four fields:
 - A 20-bit label value
 - A 3-bit Class of Service and Explicit Congestion Notification (ECN)
 - A 1-bit end of stack
 - A 8-bit time to live field
- MPLS requires a Layer 3 topology to operate. Brocade utilizes OSPF to provide routing in the MPLS cloud.
- MPLS VLL is a method of providing point-to-point Ethernet /VLAN connectivity over an MPLS domain.
- VPLS enhances the point-to-point (VLL) connectivity by specifying a method for Virtual Circuits (VC) to provide point-to-multipoint connectivity in the MPLS domain.
- In the MPLS VPN backbone, the label stack is used to forward VPN packets.
- A route-target extended community, or route target, is a type of BGP extended community that you use to define VPN membership.

There are no review questions and answers for this chapter.

Border Gateway Protocol (BGP)

For many today, it's hard to imagine life without the Internet. While it took a couple of decades to become as popular as it is today, it's actually older than most realize. On October 29, 1969, a network connection was established between systems at UCLA and systems at the Stanford Research Institute. Within six weeks, two more nodes were added to the network: one at the University of California, Santa Barbara; and the other at the University of Utah in Salt Lake City. This fledgling network was referred to as the Advanced Research Projects Agency Network (ARPANET). This network was the predecessor to the modern global network we know today as the Internet.

Before we go too far, I'd like to make a similar disclaimer to the one I made at the beginning of Chapter 10. BGP is a very complex protocol, and many books have been written on BGP alone. We will be presenting a high-level overview of the protocol here. We've got just enough room in this book to whet your appetite.

A Routing Protocol for the Internet

Back at the end of 1969, routing protocols were not really that much of a concern. Heck, there were only four nodes to deal with. During the 1970s, when the Internet grew, we had more nodes to deal with. There needed to be a way to manage the routes of so many nodes.

On January 1, 1983, the National Science Foundation (NSF) constructed a TCP/IP-based wide area network. This was designed to be a large backbone network interconnecting universities throughout the United States. There are some that consider this to be the official birth of the Internet. During the 1980s and early 1990s, the Internet was dependent upon this backbone (referred to as the *NSFNet*). Traffic was routed through this backbone (and only this backbone).

To help manage the many routes that evolved for the NSFNet, a protocol called Exterior Gateway Protocol (EGP) was created. This protocol carried the Internet through the 1980s, and into the 1990s. What were EGP's shortcomings? Well, one was that it periodically sent its entire routing table to all of its peers

(sound like RIP? It should). But the big push to replace EGP had to do with the NSFNet. As I mentioned, the NSFNet acted as a Core through which all traffic was routed. This generated heavy dependency on the infrastructure, and engineers quickly realized that the Internet would function far better if it was independent of a single entity. The Internet was also evolving into a much more international network. The dependency on one network, any network, would not suffice.

In the late 1980s, a new routing protocol called Border Gateway Protocol (BGP) was written. By 1994, BGP version 4 was the standard routing protocol for the Internet. The protocol allowed the Internet to become an independent entity. It supplanted EGP and eliminated the dependency on the NSFNet. The Internet had left the nest, and was flying on its own now.

What is BGP?

BGP is a robust routing protocol that is capable of preventing routing loops for extraordinarily large networks. This is what makes it so ideal for use with the Internet (the largest network). As of 2007, the Internet routing table housed over 200,000 routes.

BGP is described as a *path vector* protocol. Remember our discussion on RIP in Chapter 9? RIP is a distance vector protocol. It makes its routing decisions based on the “distance” from the source to the destination. With RIP, we know that “distance” refers to the number of “hops” (or routers) a packet will traverse to get to its destination. BGP makes its routing decisions based on the “path” from the source to the destination. The “path” describes the packet's journey through multiple *autonomous systems* (sound familiar? It should), not through individual routers (hops).

Another unique feature is that BGP uses TCP (port 179) to communicate to its peer routers. An initial session is established to a peer router and the entire routing table is exchanged. After the initial exchange, only updates are sent.

In some ways, BGP is kind of weird hybrid of RIP and OSPF. Its routing decisions are still based on distance (though at a “path” level), like RIP, but it also only shares updates after an initial exchange (like OSPF).

IGP vs. EGP

So many acronyms! What next?! Routing protocols are grouped into two categories: Interior Gateway Protocols (IGPs) and Exterior Gateway Protocols (EGPs). Now, I should make it clear that from now on, when I'm referring to “EGP,” I'm referring to a collection of Exterior Gateway Protocols, not to be confused with the routing protocol from the 1980s actually named EGP (e.g., EGPv3). IGP and EGP refer to collections of routing protocols.

An IGP is used within a network or series of networks that are under one entity's control. RIP and OSPF are both examples of IGPs. If I'm designing a network for my business, I could use static routes, RIP, or OSPF (or a mixture of the three) to allow my routers to communicate to each other throughout my

company's network. I'm not routing outside the company's network. Or at least, I'm not controlling the routing outside my company's network. I'm controlling it on the inside. This is an Interior Gateway Protocol (IGP).

But what if I need to control routing as traffic traverses outside of my entity (through untold numbers of entities)? This is where an EGP comes into play. BGP version 4 is probably the most well-known EGP example, but there are others. Intermediate System to Intermediate System (IS-IS) is one that can still be found in use. The EGP's job is to guide traffic as it travels outside of your control to its destination. It doesn't necessarily have to be the Internet that it traverses, but that's clearly the most common example.

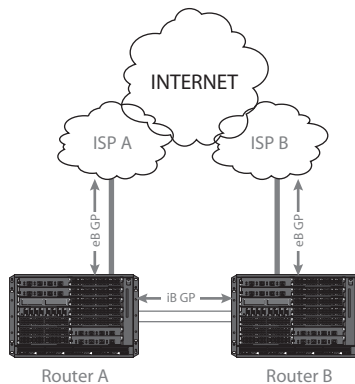
eBGP

Remember when I said BGP was the most well-known example of an EGP? I was right, but specifically, I should have said eBGP. BGP can be used as both an IGP and an EGP. The EGP version is known as *exterior* BGP (eBGP). eBGP is what you would use to communicate to your Internet Service Providers (ISPs) for communication through the Internet. It is the protocol that guides your traffic as it leaves your control. You don't control the Internet, but you can use eBGP to send and receive traffic through the Internet.

iBGP

If eBGP is the exterior version of BGP, then iBGP would be the *interior* version. This is the IGP version of BGP. Very often, this is used in *multihoming*. Multihoming is a fancy-sounding way of saying that you're using two or more Internet Service Providers actively. This allows traffic inbound from the Internet (and outbound to the Internet) to take more than one route. Multihoming provides redundancy and a certain level of link load balancing.

As multihoming is popularly used for redundancy, usually you've designed two or more routers to house the ISP connections. Here's an example of a multihomed Internet connection.



In the image, eBGP is used from each router to communicate to their respective ISP. Between your routers, iBGP is used to share the information obtained from each ISP.

OSPF, RIP or any other IGP could be used to share the routes between Router A and Router B, but there's a cost. Let's say you wanted to use OSPF. OSPF is a link state protocol. It uses cost to determine the best route. BGP (whether interior or exterior) uses path to determine the best route. Any time you convert from one protocol to another, you may not get the most accurate portrayal of which route is the best. Just as in translating languages from one to another, sometimes the meaning gets "lost in the translation." With iBGP, on the other hand, you're using an IGP, but an IGP that uses the same metrics (path vector), so nothing is lost in translation between eBGP and iBGP. Nothing needs to be translated.

iBGP may also be used without necessarily tying in with eBGP. It is an IGP. It could be used in place of RIP, OSPF, or any other IGP. Commonly, iBGP will be deployed in either a large scale environment (often in connecting two companies who may have merged) or, as we showed earlier, in multihoming.

The Autonomous System (AS)

With BGP, the Internet is broken up into smaller subgroups called Autonomous Systems (AS). Autonomous systems, when pertaining to BGP, refer to a collection of routers that are all under the same administrative entity. It's easy to get these confused with the autonomous systems we described in OSPF. They're similar, but not always the same. In OSPF, the autonomous system was a collection of routers running the same routing protocol. In BGP, the autonomous system is still a collection of routers, but they could be running multiple IGPs amongst themselves (and still be considered a single autonomous system). The key with BGP is that the collection of routers is administrated (or maintained) by a single entity.

For example, consider a business with offices in Chicago, Salt Lake City, and Los Angeles. Should this business interface with the Internet, commonly it would do so as a single AS. It has offices that are geographically diverse, but all of the routers in the networks are maintained by the same entity.

In BGP, an autonomous system is designated by a number: the AS number. This number is 16 bits, so it can be any number from 1 to 65,535. However, just like IP addresses that are used on the Internet, AS numbers must be publicly registered. The good news is, the place to publicly register IP addresses and the place to publicly register AS numbers are the same.

As of 2010, there are five organizations that publicly register IP addresses and AS numbers. Each one covers a geographic region.

Registry	Geographic Coverage
AfriNIC	Africa
APNIC	Australia and most of Asia
ARIN	North America
LACNIC	Central and South America
RIPE	Europe (including Russian Federation) and the Middle East

For more specific geographic information, consult any of the five registry websites. Each website is “www.” followed by its name, followed by “.net” (e.g., “www.arin.net,” “www.apnic.net,” “www.lacnic.net,” etc.).

It is not essential for you to register a public range of IP addresses or a public AS number to communicate with the Internet. Very often, ISPs will share a portion of their IP addresses. If you do have a publicly registered range of IP addresses, you may still advertise them to the Internet under the umbrella of your ISP's publicly registered AS number (with your ISP's approval, of course). The larger your administrative entity is, however, you will find that you may want (or even need) to publicly register a range of IP addresses and a public AS number.

Just like IP addresses, the registries have set aside a special range of AS numbers that are considered *private* AS numbers, and are not routed throughout the Internet. The private range is 64,512-65,535. Now, where the heck did they come up with 64,512?! When you're dealing with anything computer or networking related, and you wonder why a certain, seemingly arbitrary number is chosen, convert the number to binary or hexadecimal. You'll find that it makes much more sense. That rule applies here. In hexadecimal, 64,512 is FFC0. So, from FFC0 to FFFF is private.

Why would you use a private AS number? You can use them within your own entity, certainly (especially if you are using iBGP). More commonly, they are used between an entity and an ISP. Let's say you have a publicly registered range of IP addresses. You would like to advertise those IP addresses through your ISP to the Internet, but you don't have a publicly registered AS number.

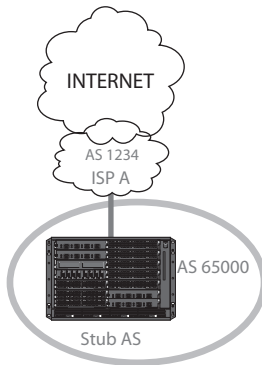
Here, you would have to coordinate with your ISP. Your ISP would set up a private AS, so that you may BGP peer with it. It will receive your routing advertisements, and pass them along as part of its own publicly registered AS number. You would still be using eBGP to talk to your ISP, but your AS number (the private AS number) would not be advertised to the Internet. Instead the IP address range would show as being part of your ISP's AS.

BGP counts the number of ASs a packet would have to travel through to get to its destination. In an over-simplifying way, the shorter the AS path (the number of ASs to get to the destination), the more desirable route. This is how path vector and distance vector are related. There's no consideration for a particular link's capacity or congestion. The foundational element BGP uses to make its decision is the length of the AS path (or the number of ASs a packet will travel through to get to its destination).

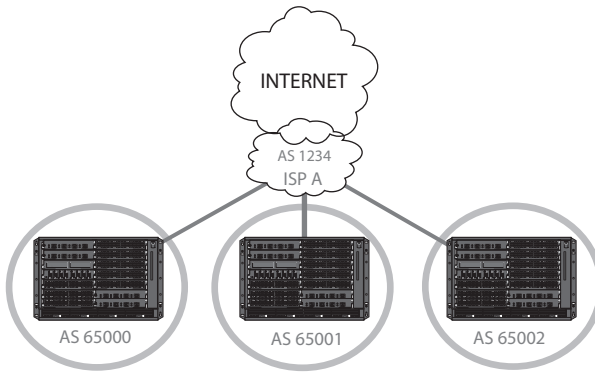
The AS path is also how BGP prevents routing loops. If a router sees its own AS within the AS path, it will drop the route. BGP depends on the IGP within the AS to prevent routing loops within the AS. Without the AS, BGP depends on the AS path.

Stub AS

Where have we seen the word “stub” before? Oh yeah! Last chapter! There, we were talking about a Stub Area. A Stub AS is a similar concept. Essentially, it's an AS that has only one link to the external infrastructure (for example, the Internet). Here, you don't necessarily need to know your ISP's full Internet routing table. You know what your next hop is going to be. If you want to get to the Internet, you've got only one way to go.



In this example, the Stub AS (AS 65000; a private AS) is peering with ISP A. ISP A is advertising AS 65000's IP addresses through its own AS (AS 1234). Now, imagine this ISP doing the same thing for multiple customers.



All three of these customers are peering with ISP A using a different private AS number. Each of these three ASs are Stub ASs. They only have one link to the infrastructure outside of their own AS (in this case, the Internet). But here's a unique thing. Each of the private ASs are depending on the same public AS. In essence, their kind of a subdivided AS to the bigger AS (AS 1234). When multiple Stub ASs are under the control of a single, larger AS, that's called a *confederation*.

Now, you may be looking at the Stub AS, and thinking, "Why are they even using BGP? Couldn't you just use a statically-configured default gateway, and have ISP A advertise your IP addresses for you?" The answer is yes, and very often, a default gateway is all that's done. Probably the number one reason for creating a BGP session for a Stub AS is for planning ahead. Sure, you're only using one link and one ISP now, but who knows what the future will bring? If you use a static default gateway, and you want to start multihoming, you'll have to make an intrusive change (e.g., you'll experience a temporary loss in service) to implement it. But if you're already using BGP, adding another peer is often not intrusive at all. Another reason may have to do with having control over how your IP addresses are presented to the Internet, but that goes beyond the scope of this book.

Neighbor Peering

With OSPF, the protocol discovered your neighbors for you (remember the neighboring process?). BGP does not using a self-discovery mechanism. You have to specifically define your neighbors (or, as they are sometimes called, *peers*). The neighbor definition must be configured on both sides (e.g., both routers), or the session will not succeed.

Once the neighbors are defined on both routers, a TCP session (using TCP port 179) will be created between the two routers. They will exchange their routes (that they have been configured to exchange; not necessarily their entire routing table). Now, they will keep the same TCP session open, and use it to exchange updates as they happen. This initial TCP session is maintained for a

long time. It is not uncommon for Internet BGP routers to have sessions that have not been reset for many months (even years). Every bit of communication that needs to be done between the two neighbors is done using that existing session.

Peering States

When neighbors peer, they may go through six states. These often happen so quickly that most are not seen, but if you're troubleshooting a problem, you may recognize some of these states:

- **Idle.** Nothing's happened yet; the neighbor has been defined, but the peering session hasn't started; you may see this state if a peering session has been hard reset, or if a connection has been failing, and it is about to try again
- **Connect.** The router has sent a TCP SYN on TCP port 179 to the neighbor; if it succeeds and the three-way handshake completes, we move to the OpenSent state; if it fails, the router will continue to listen on TCP 179 (for the neighbor to initiate a session), and keep trying; if you see a neighbor in Connect state, it is most likely a problem with Layer 3; try pinging the neighbor; if it can't reach the neighbor via Layer 3, it certainly isn't going to reach it via Layer 4 (TCP)
- **Active.** You'd think this is a favorable state to see; it isn't; Active means that the router has attempted to establish a TCP session with the neighbor and it has failed for some reason; it also means that it is "actively" listening on TCP 179 for the neighbor to make a connection; if you see a neighbor hung in Active, you've either got a Layer 4 problem (e.g., a firewall between the router and the neighbor is not permitting TCP 179 traffic) or perhaps the neighbors are not defined at both ends; you will also see this if you are using an MD5 key, and the key is not the same on both routers
- **OpenSent.** The router is waiting for an OPEN message from the neighbor it just established a TCP session with; it's the neighbor's way of saying, "I'm ready"; upon receiving the OPEN message, the router will send back a KEEPALIVE message to the neighbor as its way of saying, "Got it"; if you see a router hung in this state, there is probably something wrong with the neighboring router (perhaps under heavy load?); another possibility is a Layer 7 firewall that is, for some reason, allowing TCP 179, but not allowing BGP OPEN messages
- **OpenConfirm.** The router has sent its own OPEN message to the neighbor and is waiting for the KEEPALIVE message to come back; if a router is hung in this state, you've either got something wrong with your neighbor's router, or a bizarre Layer 7 firewall configuration in between the two peers
- **Established.** Once the KEEPALIVE has been heard from the neighbor, we switch to Established state, and the initial exchange of routes begins; the peer will remain in this state, unless the session is somehow interrupted (e.g., link goes down, router is rebooted, etc.)

To summarize, the peers go through this checklist:

- TCP three-way handshake (Connect state) — *check*
- You told me you were ready (OPEN), and I acknowledged (KEEPALIVE) (OpenSent state) — *check*
- I told you I was ready (OPEN), and you acknowledged (KEEPALIVE) (OpenConfirm state) — *check*
- We're Established! Let's start talking!

Configuring BGP

As complicated as the BGP protocol is, configuring a basic BGP setup is very straightforward. We start by telling the switch's Global config that we want to use BGP:

```
BR-Switch#conf t
BR-Switch(config)#router bgp
```

Now, one of the very next things we'll want to do is define which AS the switch is in. Where OSPF may have interfaces in many different areas, one switch may only be a member of one AS. It may peer with other ASs (and often does), but it may not be a member of more than one AS.

```
BR-Switch(config)#router bgp
BR-Switch(config-bgp-router)#local-as 12345
```

Here, we've defined our AS as AS 12345. This is a public AS number, so we would want to make sure we have it registered with the appropriate Internet Registry. We could certainly have used a private number as well (which would require no registration).

Configuring Neighbors (Peers)

Neighbors are defined in the BGP config. You define the IP address of the neighbor, and the AS that the neighbor belongs to.

```
BR-Switch(config)#router bgp
BR-Switch(config-bgp-router)#neighbor 1.2.3.4 remote-as 731
```

We've defined our neighbor's address as 1.2.3.4, and we've specified the neighbor's AS as AS 731. This is important, because this defines whether you are using iBGP or eBGP. The neighbor we've configured above is eBGP. Why? Because the AS number of the neighbor is different from the local AS we defined earlier (AS 12345). The following is an example of defining an iBGP neighbor:

```
BR-Switch(config)#router bgp
BR-Switch(config-bgp-router)#neighbor 4.3.2.1 remote-as 12345
```

Even though the command says **remote-as**, the AS defined is the same AS as the local AS we defined earlier. This is an iBGP peer.

You can define as many neighbors as your switch's resources will allow. Some can be iBGP. Some can be eBGP.

Also, like OSPF, you can define an MD5 key for authentication. This key must be defined the same on both sides, otherwise, the peering will never establish.

```
BR-Switch(config)#router bgp
BR-Switch(config-bgp-router)#neighbor 1.2.3.4 remote-as 731
BR-Switch(config-bgp-router)#neighbor 1.2.3.4 password
Something
```

Later, if you show the running config, “Something” will be replaced by an encrypted hash.

Advertising Routes

I implied this earlier, but to share routes via BGP, you must define which routes you want to share. You do so using this command:

```
BR-Switch(config)#router bgp
BR-Switch(config-bgp-router)#network 1.2.3.0/24
```

Notice that you have defined only one network: 1.2.3.0/24. This will be the only route passed to your eBGP peer. Also, if you are peering with an Internet Service Provider, you will find that you will need to tell them which routes you are advertising. They've learned long ago that there are people who will try to advertise network ranges which they do not own, or they may make a mistake in the configuration. Coordinating with the ISP provides an extra checkpoint.

When configuring an iBGP peer, you will share all of the eBGP routes learned to your iBGP neighbors by default. No extra configuration is needed.

When routes are sent, they include the network address, the AS number (that the network belongs to), the next-hop (the IP address of the router you would send to; usually the IP address of your router, the router that advertised the route), and the route origin (which would be the AS path up to this point; if your router is originating the route, it would include only your AS number). For example, we are advertising network 1.2.3.0/24. When this route is passed to our eBGP peer, the neighbor will see:

- Network address — 1.2.3.0/24
- AS number — 12345
- Next-hop — <IP address of our router>
- Route Origin — 12345

Loopback Interfaces (iBGP)

Just as in OSPF, it's very wise to use a loopback interface for the neighbor address. Here again, if you have multiple ways that your switch may be reached, you don't want a session to die purely because only one of your interfaces went down. The loopback interface is always up, as long as the switch is up.

But how do we use our loopbacks? You need to configure an additional command when you configure your iBGP neighbor:

```
BR-Switch(config)#router bgp
BR-Switch(config-bgp-router)#neighbor 4.3.2.1 remote-as 12345
BR-Switch(config-bgp-router)#neighbor 4.3.2.1 update-source
loopback 2
```

The **update-source** command tells BGP what address your router should be communicating from when you're speaking to this neighbor. Next, you specify the interface with the IP address that you want to use as your "from" address. In this case, we chose "loopback 2." It could just as easily have been "loopback 1" or "loopback 4" or whatever. Now, your iBGP neighbor will see the TCP session as the address of your loopback interface.

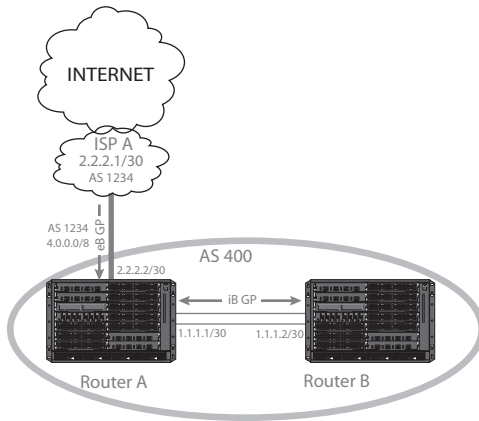
But I keep specifying iBGP, don't I? Well, it's certainly possible to use a loopback interface with eBGP, but it is not very common. Largely, this is due to the expense of the links. If you are connecting to an Internet Service Provider, each link costs money. More often than not, only one link will be purchased. For redundancy, many engineers purchase additional single-links from different Internet Service Providers. The point is, if you're dealing with one link, you only have one path to your switch. The loopback is used because you may have multiple paths to your switch.

Also, let's say you only have one link from your switch to your ISP. When that link goes down, you want your session to register as down. That way, BGP will start sending and receiving over other links. In summary, loopbacks in iBGP are a great idea; loopbacks in eBGP is not usually practical.

iBGP - next-hop-self

Using iBGP with eBGP is great. You've got no metric conversions. You're sharing your routes inside your own AS. But let's remember one thing. When a route is advertised through BGP, it is advertised as a path into a new AS. It will give the AS number and the next hop IP address of a router that will get you there.

Consider this illustration:



ISP A has advertised a route for 4.0.0.0/8 which is in its own AS (1234). ISP A tells Router A that the next hop for this network is 2.2.2.1. Router A has an iBGP peer with Router B. Router A sends the route to Router B: 4.0.0.0/8 in AS 1234 has a next-hop of 2.2.2.1. Great, right?

Router B has a problem. He doesn't know how to get to 2.2.2.1! BGP makes no assumptions. Just because he received the route from Router A doesn't mean that Router A knows how to get there either. And it doesn't matter. Router B looks in its routing table and finds no entry for a 2.0.0.0/8 (or any sub addresses) anywhere! This becomes a dead route.

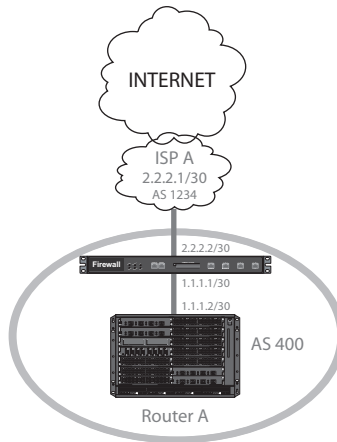
Somehow, we need Router A to relabel the route to show that it is Router A that is the next hop. In Router A's BGP config, we need the following:

```
BR-RouterA(config)#router bgp
BR-RouterA(config-bgp-router)#neighbor 1.1.1.2 remote-as 400
BR-RouterA(config-bgp-router)#neighbor 1.1.1.2 next-hop-self
```

The "next-hop-self" tells Router A that when it advertises its eBGP routes to Router B (neighbor 1.1.1.2), it should show the next hop router as itself (Router A). With this addition, Router B will see the route for 4.0.0.0/8 in AS 1234, and it will have a next-hop of 1.1.1.1 (Router A). Next-hop-self is almost always a wise move when you're using iBGP.

eBGP - multihop

What happens when you have an Internet router that is not powerful enough to run BGP? Or maybe you've got a nice BigIron to run your eBGP, but you want to put a firewall in front of it? While some firewalls can run BGP, generally speaking, BGP is not a firewall's area of expertise. It's designed for packet analysis, not routing.



In this illustration, we want Router A to handle the eBGP, not the firewall. Here's what we need to do:

For ISP A:

- We need a static route for 1.1.1.0/30 to go to 2.2.2.2
 - `ip route 1.1.1.0/30 2.2.2.2`
- We need to add one special command to ISP A's BGP config:
 - `local-as 1234`
 - `neighbor 1.1.1.2 remote-as 400`
 - `neighbor 1.1.1.2 ebgp-multihop 2`

For Router A, we need a nearly identical configuration:

- We need a static route for 2.2.2.0/30 to go to 1.1.1.1
 - `ip route 2.2.2.0/30 1.1.1.1`
- We need to add one special command to Router A's BGP config:
 - `local-as 400`
 - `neighbor 2.2.2.1 remote-as 1234`
 - `neighbor 2.2.2.1 ebgp-multihop 2`

Notice that we had to first provide a static route for ISP A to reach Router A (and vice versa). We used static routes, but you can use other IGP. The idea is that the two distant-separated routers have to be able to talk to one another (e.g., they have to know the route there and the route back).

Once they can talk to each other, they simply configure an “ebgp-multihop.” This tells both routers that they are not directly connected to each other. They must pass through other routers (hops) to communicate. In this case, we’ve

specified that each router is “2” hops away from each other (the firewall is one hop; the destination is the second hop). Notice that this must be configured on both ends. Also, notice that this function is only available with eBGP. There is not a function for this in iBGP.

BGP show Commands

When configuring a routing protocol, it always helps to be able to see how the protocol is doing (and also to make sure you configured it correctly). Here are some handy show commands that will help you do just that:

- **show ip route** — You know by now that this command is not unique to one specific routing protocol, but it's awfully handy (and often forgotten) when troubleshooting routing protocols, including BGP; as a reminder, this command displays the contents of your router's routing table; this shows all routes learned, regardless of the protocol; warning: If you are receiving full Internet routes from an ISP via BGP, you may want to specify a specific network or IP address when using this command (e.g., **show ip route 10.0.0.0**); we all have better things to do than page through over 200,000 routes
- **show ip bgp route** — Displays only routes learned through BGP; this command includes the AS Path, next hop, and other details of each route; this command will also display all routes learned; for example, if you were multihoming, you would likely receive at least two different routes (one from each provider) for any given network; the **show ip route** command will only show you the route that has been chosen as “best”; the **show ip bgp route** command will show you all the routes it has learned for each network, and will show you which route it has chosen as “best”; warning: the same rule applies with **show ip bgp route**, although here, it's worse; you'll have over 200,000 routes per ISP; once again, use **show ip bgp route <network>**
- **show ip bgp neighbor** — Displays information on all of your BGP neighbors (peers); this will tell you the neighbor's state; whether its eBGP or iBGP, BGP messages sent, all kinds of good information
- **show ip bgp summary** — This one gets used all the time; it's a quick and dirty display of all of your neighbors (peers), what state they're in, how many routes they've sent and received, how long the session's been up, and more; all in a nifty little table
- **show ip bgp config** — Just like the OSPF command, this is one of the handiest ever, especially if you've got a switch with a very large config; this will show you just the section of the config pertaining to BGP

Summary

- BGP is the standard routing protocol for the Internet
- BGP is a path vector routing protocol
- BGP is both an IGP (iBGP) and an EGP (eBGP)
- An autonomous system (AS) is a collection of routers that are all administered by the same entity
- Multihoming is providing more than one entrance and exit link for your AS (e.g., peering with more than one Internet Service Provider)
- A Stub AS is an AS that only has one entrance and exit link
- A confederation is a collection of Stub ASs
- In BGP, a neighbor (peer) must be manually defined
- A neighbor goes through several states:
 - Idle - no packets have been sent or received yet
 - Connect - a TCP session has been attempted
 - Active - the TCP session failed, but the router is actively listening
 - OpenSent - TCP succeeded, and the router is waiting for an OPEN message
 - OpenConfirm - router sends an OPEN message, and waits for KEEPALIVE
 - Established - routers exchange routing information
- Networks must be defined; by default, BGP shares nothing
- Neighbors in the same AS are automatically iBGP
- Neighbors in different ASs are automatically eBGP
- Use loopback interfaces with iBGP, but be sure to use “update-source”
- When using iBGP, remember to use “next-hop-self” to make sure those within your AS know how to get out
- There's no metric conversion when using iBGP and eBGP together
- When your BGP router can't be directly connected to your eBGP peer router, use “ebgp-multihop”

Chapter Review Questions

1. On what metric does BGP make its routing decisions?
 - a. next hop
 - b. link cost
 - c. hop count
 - d. AS path
2. Two neighbors in the same AS are using what type of BGP?
 - a. iBGP
 - b. eBGP
 - c. BGP
 - d. EGP
3. What type of routing protocol is BGP?
 - a. distance vector
 - b. link state
 - c. path vector
 - d. distance state
4. In iBGP, what command can I use to make sure my iBGP peers know the way out of the AS?
 - a. ebgp-multihop
 - b. next-hop-self
 - c. update-source loopback 1
 - d. password
5. What state would a peer be in if it tried to establish a TCP session, but the session failed?
 - a. Idle
 - b. Connect
 - c. Active
 - d. OpenSent
6. What is a collection of Stub ASs called?
 - a. Confederation
 - b. Autonomous System (AS)
 - c. Peer
 - d. Stubbies

7. What describes each AS a packet must travel through to get to its destination?
 - a. BGP routing table
 - b. Peer
 - c. AS path
 - d. MED
8. What is the private AS range?
 - a. 1,024 - 65,535
 - b. 1,023 - 65,535
 - c. 32,768 - 65,535
 - d. 64,512 - 65,535
9. I've placed a firewall between my BGP router, and the link from my Internet Service Provider. What command would allow you to establish an eBGP session?
 - a. next-hop-self
 - b. ebgp-multihop 2
 - c. update-source loopback 1
 - d. show ip bgp route
10. What is multihoming?
 - a. Peering with multiple iBGP peers
 - b. Owning more than one public AS
 - c. Peering using more than one loopback interface
 - d. Peering with more than one eBGP peer

Answers to Review Questions

1. d. BGP is a path vector protocol. It makes its decisions based on the AS path of the route.
2. a. interior BGP or iBGP is the IGP version of BGP. It is for use within an AS.
3. c. path vector. Answer A refers to protocols like RIP. Answer B refers to protocols like OSPF.
4. b. **next-hop-self** tells the router receiving eBGP routes to set the "next hop" of the route to itself before advertising it to its iBGP peers.
5. c. Active state is when a TCP session has been attempted, but has not succeeded. The router is "actively" listening in case its peer attempts to establish a TCP session with it.
6. a. A confederation is a collection of Stub ASs.

7. c. The AS path lists each AS a packet must travel through to get to its destination.
8. d. 64,512 - 65,535 (or FFC0 - FFFF) are private AS numbers.
9. b. **ebgp-multihop** allows you to establish an eBGP session with a router that is not directly connected to your router.
10. d. Multihoming means to peers with more than one eBGP peer. It provides multiple entrance/exit links to your AS.

Security, Redundancy and More

We've reached the end of the Layer 3 section. By now, you should have a foundational overview of routing and routing protocols. But there's so much more that a Layer 3 switch can do for you, besides just routing and switching. In this chapter, we're going to discuss a few more features that don't quite fit under the Layer 2 or Layer 3 umbrella.

Security: Access Control Lists (ACL)

Even in the early days of routers, it didn't take designers long to realize that if this appliance was going to decide how a packet was going to get to its destination, it could certainly decide if the packet would get to its destination. Through Access Control Lists (ACL), an administrator could regulate what traffic is permitted or denied through a given path. Permitted packets are forwarded. Denied packets are dropped.

There is one thing that I want to make perfectly clear early in this chapter. ACLs, by themselves, do nothing. If you see an ACL defined in a switch's config, it does not affect the operation of the switch in any way (besides taking up a little extra space within the config). ACLs are a definition. An ACL is a list, often defining what traffic is permitted and what traffic is not. The list does nothing, until it is applied.

In this chapter, we will cover both the defining of ACLs and the applying of ACLs. We will also discuss additional uses for ACLs (beyond permitting and/or denying traffic).

Subnet Mask vs. Wildcard (or Inverse) Mask

The first thing to know is that when you are specifying a range of IP addresses in an ACL, you use what is called a wildcard mask (also referred to as an inverse mask). At first glance, this mask looks similar to the subnet mask, and it functions in a similar way. With a subnet mask, the 1s mark the portion of the IP address that defines the network address, and the 0s mark the portion that defines the host address. With a wildcard mask, the 0s mark the bits in the IP address that don't change. The 1s mark the bits in the IP address that do change.

Let's look at an example. Suppose you had a Class C network—192.168.100.0/24. You want to create an ACL that references all 254 usable addresses in that range. To reference it as a network, you know that you would use 192.168.100.0 followed by a subnet mask of 255.255.255.0. The 1s in the subnet mask demarcate the network portion of the address. The 0s show you the host portion. With an inverse mask, you would use 1s to designate the portion of the address that changes. In this case, you know that all of your desired addresses will start with 192.168.100. It's the last octet that will change. The wildcard mask would be 0.0.0.255.

```
(192) (168) (100) (0)
1100 0000 . 1010 0000 . 0110 0100 . 0000 0000
```

```
(0) (0) (0) (255)
0000 0000 . 0000 0000 . 0000 0000 . 1111 1111
```

In ACLs, “192.168.100.0 0.0.0.255” would include every address from 192.168.100.0 (the network address) through 192.168.100.255 (the broadcast address). The 1s represent the bits that are variable (e.g., that can change).

“Why don't they just use subnet masks?!” I hear you cry. There are certainly many other students of network engineering that have begged that same question. The question has one answer, really: *flexibility*. One of the tricks of the wildcard mask is that it doesn't have to be contiguous. We can place 0s and 1s where we please. It doesn't have to be in a continuous stream.

Let's take another example. You have two networks that you want to identify within an ACL. One is 10.4.1.0/24, and the other is 10.6.1.0/24. You want to be able to reference all addresses in both networks. Now, you could just create two lines: “10.4.1.0 0.0.0.255” and “10.6.1.0 0.0.0.255.” But you're using wildcard masks, and you can be more efficient than that! How about this: “10.4.1.0 0.2.0.255.”

```
(10) (4) (1) (0)
0000 1010 . 0000 0100 . 0000 0001 . 0000 0000
```

```
(0) (2) (0) (255)
0000 0000 . 0000 0010 . 0000 0000 . 1111 1111
```

“What a minute!” you say, “You stuck a 1 in the 15th bit all by itself!” That's true. And that one tells the ACL processor that this bit can change. It has the potential to change the value of the second octet. “10.4.1.0 0.2.0.255” will translate to all of the addresses from 10.4.1.0 through 10.4.1.255 and 10.6.1.0 through 10.6.1.255. When that 15th bit is a value of zero, the second

octet is “4.” When the 15th bit is a value of one, the second octet is “6.” Therefore, this one line (“10.4.1.0 0.2.0.255”) will represent all of the addresses in both networks.

But what if you have four different networks? Let's use 10.4.1.0/24, 10.5.1.0/24, 10.6.1.0/24, and 10.7.1.0/24. Well, take a look at the previous example. We would just need to represent more wildcard bits to the second octet, right? Did you come up with “10.4.1.0 0.3.0.255?” If you did, great! That's the right answer! If not, don't worry. There are certainly no rules stating that you must come up with the most efficient rule. You could still represent all four networks with four separate lines (e.g., “10.4.1.0 0.0.0.255,” “10.5.1.0 0.0.0.255,” etc.). You may find that the flexibility of the wildcard mask may make your life easier. And either way, it always helps to know what your options are.

Numbered Access Lists

In the early days of ACLs, each list was defined by giving it a number. Seems simple enough, right? But we also had to distinguish if this was an ACL for, say, IP, IPX, DECnet, XNS, AppleTalk, etc. How would the router be able to tell them apart?

To solve this problem, each type was given a numeric range. A few included Standard IP (1-99), Extended IP (100-199), Standard IPX (800-899), AppleTalk (600-699), and so on. Each type of network ACL would receive a documented range of 100 numbers (except for Standard IP, which is stuck with 99).

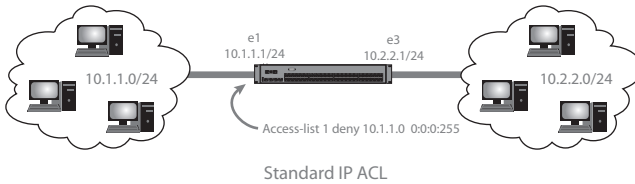
This system worked fine for many years, but it has recently become much less practical. For instance, so many of these other networking protocols are becoming less popular (some are no longer supported on modern equipment). TCP/IP is here to stay. This means that, for many customers, there are a lot of wasted ACL numbers. Even if you're only using IP, you are still restricted to the range of 1-99 for Standard IP and 100-199 for Extended IP. Then, there's the more obvious question: what if we need to define more than 100 ACLs on one switch? Using numbered ACLs, you're out of luck.

Numbered ACLs are still used today. On modern switches, support for protocols other than IP have been removed. If you are configuring numbered ACLs, you have a choice between Standard IP (1-99) and Extended IP (100-199). That's it.

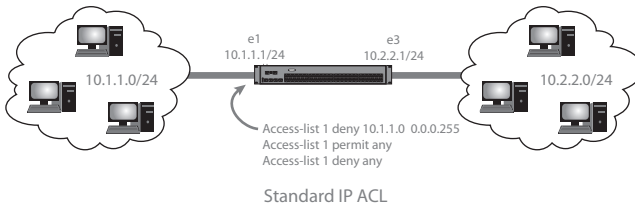
Now, what's all this about “Standard IP” and “Extended IP?” What are they? What's the difference between them? Well, let's talk about that.

Standard IP Access Lists

A Standard IP access list will permit or deny traffic according to the source IP address of the traffic. For example, let's say you had two networks: 10.1.1.0/24 and 10.2.2.0/24. Let's say that 10.2.2.0/24 was the Accounting network, so you wanted to severely limit anyone from communicating to that network (for security's sake). You could create a Standard IP access list that denies traffic coming from 10.1.1.0/24.



As defined, this ACL will actually do more than you intended. At the end of every ACL is an implicit “deny any.” This means that if traffic isn’t explicitly permitted, it’s denied (guilty until proven innocent). This ACL is not only denying traffic from 10.1.1.0/24, but it’s denying traffic from everywhere! Let’s try to define the ACL again, so that it will deny just 10.1.1.0/24, but allow everything else.

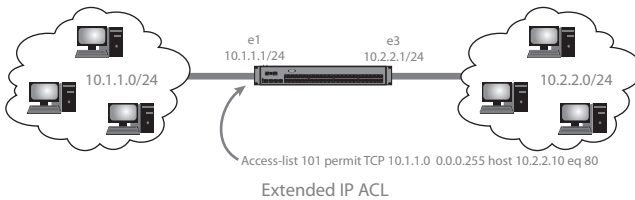


ACLs are read from top to bottom. If a packet doesn’t match the first line, it goes on to the next. If it doesn’t match any lines, it will fall under the implicit “deny any,” and be dropped. In this case, if the packet is coming from 10.1.1.0/24, the first line will match it, and deny it. If it is coming from any other address, it will not match the first line, but will match the second line (“permit any”), and will be allowed through. The “any” keyword actually translates to the address and wildcard mask of “0.0.0.0 255.255.255.255.” The wildcard mask is all 1’s. This means that any of the bits can change. You might find it easier to remember and use the word “any,” but you can certainly use “0.0.0.0 255.255.255.255” as well. They mean the same thing.

Extended IP Access Lists

But what if you want to restrict access using more criteria than just the source IP address? What if you actually want to permit a range of IP addresses, but only for, say, HTTP (TCP 80) traffic? Or what if you want to permit a range of IP addresses, but only if they are bound for a specific destination IP address? This is why we have Extended IP access lists.

Standard IP access lists will match traffic only on the source IP address of the packet. Extended IP access lists will match traffic based on source IP address, destination IP address, and protocol. With that, let’s go back to our example. Let’s say we want the 10.1.1.0/24 network to be able to reach the 10.2.2.0/24 network, but only if they’re communicating using TCP 80 (HTTP) to the web server (say, 10.2.2.10).



Notice that this access list is not “1” anymore. I had to use a number between 100 and 199 because this is an Extended IP access list. Also, notice that this line is considerably longer than the Standard IP list. I had to include much more information. We’ll go more into syntax later, but for now, notice that I had to specify the protocol (“tcp”), the source IP addresses (“10.1.1.0 0.0.0.255”), the destination IP addresses (“host 10.2.2.10”), and I’ve even specified the specific TCP port that is allowed (“eq 80”). By the way, the keyword “host” is similar to the keyword “any.” It actually translates to an address and wildcard mask, but it’s usually easier to use the keyword. In our example, “host 10.2.2.10” would be the same as “10.2.2.10 0.0.0.0.” Notice that the wildcard mask is all zeros. It’s saying that none of the bits will change. It is this address (10.2.2.10) and only this address.

The important thing to take away from all of this is that the Extended IP access list allows you to be much more granular in specifying which traffic is allowed and which is not. Remember, Standard IP access lists only match traffic by source IP address. Extended IP access lists match traffic by source IP address, destination IP address and protocol.

Named Access Lists

Numbered access lists are fine, but what if you want to create more than 100 Extended IP ACLs on a single switch? While this kind of problem doesn’t come up often, there’s also the problem of trying to remember why a certain ACL may have been created in the first place (or what its purpose is). Sure, you may have a document somewhere that details the number of each ACL you’ve created and matches a description. But wouldn’t it be nice if you could simply give it a descriptive name instead of a number?

That’s the Named access list. Now, instead of specifying a number, you create the ACL by giving it a name, and then specifying whether it is a Standard or Extended ACL. That last part is important. Remember that the switch could distinguish between Standard and Extended IP ACLs simply by looking at the ACL number. We’re not using a number anymore, so it’s up to the engineer to specify whether the Named ACL is Standard or Extended.

The name can be anything you like. It is wise to choose a name that means something to you (or your fellow administrators). You cannot change the name once you’ve chosen it. You must delete the old ACL and create a new one (with the new name).

Creating Access Lists

Okay, so let's get to work. Now that we know all these things, how do we actually create the lists?

Numbered Standard IP Access Lists

Let's start with the Numbered Standard IP access lists. These are defined in the global config. The syntax looks like this:

```
BR-Switch#conf t
BR-Switch(config)#access-list 1 deny host 10.1.2.3
BR-Switch(config)#access-list 1 permit any
```

This is a simple two-line access list that prevents any traffic coming from 10.1.2.3, but allows all other traffic. Remember that access lists are read from top to bottom. The first line that matches a packet will be applied. Consider, for instance, if we configured this:

```
BR-Switch#conf t
BR-Switch(config)#access-list 1 permit any
BR-Switch(config)#access-list 1 deny host 10.1.2.3
```

Does this access list deny any traffic? No, actually. Every packet will hit the first line ("permit any") and be allowed through. In fact, that deny line does need to be there at all. Now, what if we go back to our first example, but leave off the permit line:

```
BR-Switch#conf t
BR-Switch(config)#access-list 1 deny host 10.1.2.3
```

Now, the access list actually denies all traffic. Remember that there's an implicit "deny any" at the bottom of the list ("deny ip any any" for Extended IP). If we have an access list that includes only deny lines with no permit lines, no traffic will be allowed through at all. In fact, the above example could just as easily have been written:

```
BR-Switch#conf t
BR-Switch(config)#access-list 1 deny any
```

Remember that there is an implicit "deny any" at the end of the list. Remember that the access list is read from top to bottom. This means it's typically best to put your deny statements close to the top of the list, and your permits later on.

Let's talk about syntax for a minute now. Let's go back to our original example:

```
BR-Switch#conf t
BR-Switch(config)#access-list 1 deny host 10.1.2.3
BR-Switch(config)#access-list 1 permit any
```

You start out in the global config. The first key word you type is **access-list**. This specifies that you are going to define a Numbered ACL. Now, is it a Standard IP or an Extended IP? We don't know yet, we haven't specified the number. That's the next part. Here, we've specified "1." This tells the switch that you are defining a Numbered Standard IP access list. We could have just as easily used "2"

or “10” or “99.” There are many times in which you will need to create multiple different ACLs on a given switch. This is how Numbered ACLs are distinguished. Above, we’ve created “access-list 1.” If we need a different Numbered Standard IP access list, we can create “access-list 2,” or whatever number between 1 and 99.

Let's go to the next part. You've seen the keywords “permit” and “deny.” They're pretty intuitive. The keyword “permit” allows the traffic through, if it is matched. The keyword “deny” drops the packet, if it is matched. There are other commands that may be used here, but this goes beyond the scope of this book. For most all of the ACLs that you will work with, you need only remember “permit” or “deny.”

Finally, we need to specify the source IP addresses. We can do this by choosing a specific IP address with a wildcard mask (e.g., “10.1.1.0 0.0.0.255”). We can also specify a specific single IP address by using the keyword “host” (e.g., “host 10.1.2.3”). Remember, “host” is the same as a wildcard mask of “0.0.0.0” (e.g., “10.1.2.3 0.0.0.0”). Or we can specify “any” IP address. Remember that the keyword “any” means the same thing as “0.0.0.0 255.255.255.255.”

To delete a specific line within the ACL, just put a “no” in front of it (e.g., “no access-list 1 permit any”). To delete an entire ACL, type **no access-list 1**.

That's it. In summary, a Numbered Standard IP access list uses this syntax:

```
access-list <1-99> <permit or deny> <source ip address(es)>
```

Numbered Extended IP Access Lists

Extended IP access lists are a little trickier, because there's so much more to play with. It's a Numbered ACL, so it is configured in the global config. Here's an example:

```
BR-Switch#conf t
BR-Switch(config)#access-list 101 deny icmp any any
BR-Switch(config)#access-list 101 deny udp host 10.2.2.3 eq 53
any
BR-Switch(config)#access-list 101 permit tcp 10.1.1.0
0.0.0.255 host 10.2.2.10 eq 80
BR-Switch(config)#access-list 101 deny ip 10.1.1.0 0.0.0.255
any
BR-Switch(config)#access-list 101 permit ip any any
```

This access lists prevents any ICMP traffic. It denies 10.2.2.3 from sending a segment with a source port of UDP 53 to any address (this is a DNS response packet). It allows the 10.1.1.0/24 network to connect to 10.2.2.10 on TCP 80 (HTTP). It denies any other traffic from 10.1.1.0/24 from going anywhere else. Finally, it allows any address to reach any other address on any protocol.

Here again, this is read from top to bottom, and it has an implicit “deny ip any any” at the bottom. We start by using the keyword “access-list.” This tells the switch that we are about to define a Numbered ACL. Next, we specify the number. In this case, it's “101.” This instantly tells the switch that we are defining

an Extended IP ACL. If the number were less than 100, it would be a Standard IP ACL. We could just as easily have chosen “111” or “199” or even “100.” I chose “101.” Then, again, we must choose to “permit” or “deny” the traffic.

Following this, we must choose the protocol. Remember the Protocol field in the IP header from Chapter 2? No? You might want to flip back and take a look at it. This 8-bit protocol number is a signal to the receiver as to which Layer 3 or Layer 4 protocol created the packet. Well-known numbers here include 1 (ICMP), 6 (TCP), 17 (UDP), and so on. Fortunately for us, most of the possible options can be referenced by name (e.g., “tcp,” “udp,” “icmp,” etc.). Still, if you're not sure of the name, you can always use the number. In other words, we could have written the first line in the example above as “access-list 101 deny 1 any any.” When you perform a “show run” later on, you'll find that the “1” was automatically translated to “icmp.” But what about the keyword “ip?” This tells the access list to match on any protocol.

Now, we need to give it the source address and source port. Source address is given the same way that we've done previously. You can either use an address with a wildcard mask, the keyword “host” followed by a specific IP address, or the keyword “any.” All three of these options are represented in our example. The source port for an initiating connection is usually a random number between 1,024 and 65,535. Notice that in our example (with the exception of the second line), we don't specify a source port. Where ports are concerned, if none are specified, the access list will match any port number. In the example of the third line, we're not really interested in matching the source port. We want to focus on the destination port (which should be a fixed number for an initiating session). Although it may not be obvious, we kept the source port blank, meaning that any source port may be used.

For specifying ports, we can use several options:

- eq — short for “equal”; this means a specific port; for example, “eq 53” means “port 53” and only “port 53”
- gt — short for “greater than”; this means all ports greater than a given number; for example, “gt 1023” means all ports from 1,024 to 65,535
- lt — short for “less than”; this means all ports less than a given number; for example, “lt 1024” would represent all well-known ports (1-1,023)
- neq — short for “not equal”; this means any port except for the one specified; for example, “neq 80” means any port from 1 to 65,535, but not 80
- range — this allows you to specify a contiguous range of port numbers; for example, “range 9000 9005” will match on port numbers 9000, 9001, 9002, 9003, 9004, and 9005

Notice with each example above, I specified a number (with range, I specified two). You may also use the names of the protocols (example: “eq http”). Not all protocols have a name associated with them, and some well-known protocols may not be represented. Use the “?” to see what named options you have available, but you can always use numbers.

Next, we need to specify the destination address and port number. This is done in the same way as the source address and port number. The address can be an IP address followed by a wildcard mask, the keyword “host” followed by a specific IP address, or the keyword “any.” The port number is specified using “eq,” “gt,” “lt,” “neq,” or “range.” It may also be left blank, meaning that any destination port will match. This is used in the second line. We want to match a source port of 53, but the destination port may be anything.

Now, source port and destination port really only play a role in TCP or UDP protocols. Layer 4 port numbers have no place in Layer 3 protocols like ICMP and OSPF. There are additional options granted, depending on the protocol you specified. For example, there are many different ICMP packets. Let's say we wanted to allow everything except Ping (ICMP Echo and ICMP Echo-Reply). We could have written the first line this way:

```
BR-Switch(config)#access-list 101 deny icmp any any echo
BR-Switch(config)#access-list 101 deny icmp any any echo-reply
BR-Switch(config)#access-list 101 permit icmp any any
```

This would allow any ICMP packets, except specifically ICMP Echo and ICMP Echo-Reply packets. In the end, the “?” is your friend. This will always guide you as to what options are available.

To delete a specific line within the ACL, just put a “no” in front of it (e.g., “no access-list 101 deny icmp any any echo-reply”). To delete an entire ACL, type **no access-list 101**.

In summary, the Numbered Extended IP access list syntax looks like this:

```
access-list <100-199> <permit or deny> <protocol> <source ip
address(es)> <source port> <destination ip address(es)>
<destination port>
```

Named Standard IP Access Lists

The form factor for Named access lists is very similar to Numbered. In this case, we start by initially defining the list in the global config. This sends us into a sub-config for Named ACLs:

```
BR-Switch#conf t
BR-Switch(config)#ip access-list standard HostDeny
BR-Switch(config-std-nacl)#deny host 10.1.2.3
BR-Switch(config-std-nacl)#permit any
```

This is a different way to write the same Standard IP access list we used in the Numbered Standard IP access list section. The difference is that instead of a number, we've given the list a name (“HostDeny”). Also, notice that the lines defining the rules do not have “access-list” in front of them. The keyword “access-list” denotes a Numbered ACL.

The function of the ACL is the same. We're denying all traffic coming from a specific IP address: 10.1.2.3, and we're permitting traffic from any other address. To remove a specific line within a Named ACL, make sure you're in

the Named ACL sub-config, and put a “no” in front of the line (e.g., “no permit any”). To delete an entire ACL, from the global config, type **no ip access-list standard HostDeny**, as an example.

The syntax can be summed up this way:

```
ip access-list standard <name>
<permit or deny> <source ip address(es)>
```

Named Extended IP Access Lists

The form factor of Named Extended IP ACLs is very similar to the Numbered ACLs. Begin by initially defining the named ACL in the global config. Then, we define the lines of the ACL in the Named ACL sub-config:

```
BR-Switch#conf t
BR-Switch(config)#ip access-list extended AllowWeb
BR-Switch(config-ext-nacl)#deny icmp any any
BR-Switch(config-ext-nacl)#deny udp host 10.2.2.3 eq 53 any
BR-Switch(config-ext-nacl)#permit tcp 10.1.1.0 0.0.0.255 host
10.2.2.10 eq 80
BR-Switch(config-ext-nacl)#permit tcp host 10.1.1.20 eq 25 any
established
BR-Switch(config-ext-nacl)#deny ip 10.1.1.0 0.0.0.255 any
BR-Switch(config-ext-nacl)#permit ip any any
```

Notice that we used the same ACL rules in our Numbered Extended IP ACL example. We just wrote them as a Named Extended IP ACL. Here, instead of using “101,” we named the ACL “AllowWeb.” The individual lines still specify protocol, source IP address, source port, destination IP address, and destination port. The end result is the same. We deny all ICMP traffic. We deny 10.2.2.3 from sending a UDP segment with a source port of 53 (a DNS response). We allow 10.1.1.0/24 to talk to 10.2.2.10 on TCP 80 (HTTP), but we deny all other traffic from that network. We’ve added a new line to allow traffic from 10.1.1.20 to send to any address with a source TCP port of 25 (SMTP) and a curious keyword “established.” This means that 10.1.1.20 can send traffic with a source TCP port of 25, but only if it is responding to a previously initiated session. When an incoming TCP SYN is received, the responding TCP SYN/ACK will show as part of an established session (and this ACL line will permit it). Finally, we allow all traffic not pertaining to the permit and deny lines above.

To remove a specific line within a Named ACL, make sure you’re in the Named ACL sub-config, and put a “no” in front of the line (e.g., “no deny icmp any any”). To delete an entire ACL, from the global config, type **no ip access-list extended AllowWeb**, as an example.

The syntax looks like this:

```
ip access-list extended <name>
<permit or deny> <protocol> <source ip address(es)> <source
port> <destination ip address(es)> <destination port>
```

Applying Access Lists

Remember, simply creating the lists does nothing. You've defined the rules, but you haven't applied them anywhere. For permitting or denying traffic, you apply the ACL to a specific interface. You do this in the Interface config:

```
BR-Switch#conf t
BR-Switch(config)#int e 1
BR-Switch(config-if-e1000-1)#ip access-group 101 in
```

Here, we've applied a Numbered Extended IP ACL to interface "e 1." How did I know it was Numbered and not Named? Because the **access-group** command specified a number ("101"). How did I know it was Extended IP and not Standard IP? The number specified was between 100 and 199.

But let's take a look at that last keyword: "in." When applying ACLs, you need to specify whether the rules apply to traffic coming into the interface, or traffic going out of the interface. In this case, we chose to apply the rule to traffic coming "in." You can also use the keyword "out" to specify traffic leaving the interface.

What about a Named ACL? Same command:

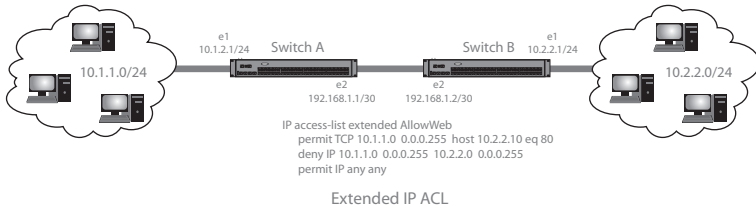
```
BR-Switch#conf t
BR-Switch(config)#int e 1
BR-Switch(config-if-e1000-1)#ip access-group AllowWeb in
```

Notice that it's not important, at this point, whether the ACL is Standard IP or Extended IP. We are simply applying a number or a name. The crucial elements at this level are which ACL is to be applied and in which direction the rules should be applied.

ACLs may be applied to virtual interfaces as well (e.g., ve 1, ve 2, etc.). They are applied the same way. The meaning is a little different. For example, let's say you have a VLAN with 10 interfaces in it, and a ve acting as the default gateway for the VLAN. You've applied an inbound ACL to that ve. This rule is not applied to each of the 10 interfaces individually, but to the ve. This means that the ACL will only be applied when the traffic hits the ve (e.g., when traffic is leaving the VLAN). If the traffic is communicating within the VLAN, the ACL will not be applied. It should also be noted that when you're using a Layer 3 VLAN, you may not apply ACLs to individual interfaces. You must apply the ACL to the ve.

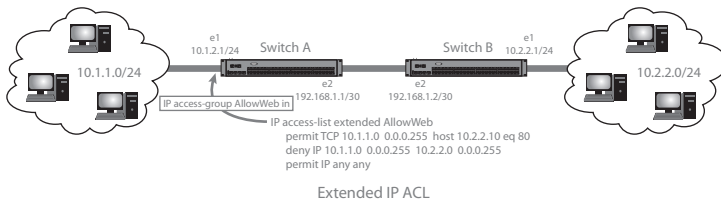
Every interface may only have one inbound ACL and one outbound ACL applied to it. You cannot have more than one ACL applied to a single interface applied in the same direction. You also don't need to apply both an inbound and an outbound. In fact, that's not very common. Usually, only one is chosen that fits the need.

Let's talk a little more about the best place to apply ACLs. Take a look at this example:



From this access list, we want to all 10.1.1.0/24 to reach our web server, 10.2.2.10, using TCP 80 (HTTP), but we don't want to allow any other traffic from 10.1.1.0/24 to 10.2.2.0/24. Finally, we want to allow all other traffic ("permit ip any any"). Where shall we apply this ACL? We could apply it to Switch B e 1 as an outbound ACL. We could apply it to Switch B e 2 as an inbound ACL. We could apply it to Switch A e 2 as an outbound ACL. Finally, we could apply it to Switch A e 1 as an inbound ACL. Where shall we place it?

Conventional ACL wisdom states that we should apply the ACL closest to the source of the traffic we're trying to control. In this case, this would be Switch A e 1, as this is the interface closest to the 10.1.1.0/24 network. Why not apply it to Switch B? Well, you can, but a packet will be traveling through Switch A and Switch B before it gets denied. That's wasting bandwidth space on the lines as well as processing on the switches. The earlier the traffic is stopped, the less resources you expend.



Rule-Based ACLs vs. Flow-Based ACLs

Speaking of resources, let's talk about the two ways in which ACLs are processed by the switch: Rule-Based and Flow-Based.

With Rule-Based ACLs, the defined ACL rules are applied to the interface's CAM (Content Addressable Memory). When a packet is received on that interface, it is compared to the rules within the CAM. According to the rules defined, the packet is either forwarded or dropped. The beauty of this design is that this operation is handled entirely by the interface's hardware. The switch's CPU does nothing. It is free to worry about more critical functions.

With Flow-Based ACLs, the received packet is checked against the session table within the interface's CAM. If there is no match, the packet will be sent to the switch's CPU for analysis. The CPU will apply the ACL rules. If it finds that

the packet should be permitted, it will permit the packet, and update the session table in the interface's CAM. The session table in CAM is a record of all the traffic that the CPU has previously permitted (after applying the ACL rules). It lists the protocol (e.g., TCP, UDP, etc.), source IP address, source port, destination IP address, and destination port of the packet. If the CPU finds that the packet should be denied, the CPU will drop the packet, but it will make no updates to the session table in CAM. CAM, at this point, is only used to record traffic that has been previously permitted by the CPU. If an incoming packet matches an entry in the CAM, the interface hardware will permit the packet. By default, the interface hardware will never deny packets (in Flow-Based ACLs). This can be changed with a global config command **hw-drop-acl-denied-packet**. This command tells the switch to update the session table in CAM for denied traffic as well as permitted traffic.

"Fascinating," you say, "but why are you telling me this?" Flow-Based ACLs are much more CPU-intensive to the switch. If you're pushing a heavy amount of traffic through the switch, and you are applying many Flow-Based ACLs, this could tax the switch's CPU resources. For this reason, it is best to use Rule-Based ACLs wherever you can.

"But how?" you ask. Remember when I said you have to specify direction in applying an ACL? Well, it turns out that all outbound ("out") ACLs are processed as Flow-Based. Inbound ("in") ACLs are processed as Rule-Based (with a few exceptions). A Rule-Based ACL will change to a Flow-Based ACL under the following conditions:

- It is applied as an outbound ACL
- A specific ICMP type is specified (e.g., "icmp any any echo," as opposed to "icmp any any")
- Network Address Translation (NAT) is applied to the interface
- ACL statistics are enabled ("enable-acl-counter")
- ACL logging is enabled

There are other, more specific circumstances as well. For up to date information on the matter, consult Brocade's web site (<http://www.brocade.com>).

To sum up, wherever possible, choose inbound ACLs, and avoid outbound ACLs, particularly for switches that are processing a large amount of traffic.

Strict Mode

When you're using Flow-Based ACLs, the TCP and UDP checks on incoming or outgoing segments aren't as thorough as you might think.

For TCP traffic, only the control segments are checked against the ACLs. Control segments would include SYN, FIN, and RST. Data segments are not checked against the ACLs. This was done to optimize performance. The line and logic is that a data segment would have to have control segments preceding it. TCP must initiate with a three-way handshake, right?

This actually works very well in most situations, but you may find yourself in a lab situation (or an Internet-facing situation) where you want to be sure that every segment is checked. This can be done by issuing the following command:

```
BR-Switch#conf t
BR-Switch(config)#ip strict-acl-tcp
```

For UDP traffic, you don't necessarily have "data" segments or "control" segments. You just have segments. As a result, every UDP segment goes through the same routine in a Flow-Based ACL. The switch checks the session table in CAM. If there's no entry (e.g., there's no record that the switch has let this combination of protocol, source IP address, source port, destination IP address, and destination port through before), the segment will be handed to the CPU to process against the ACL. The CPU will then update the session table accordingly. If there is an entry in the session table, the switch will simply forward the segment. You may find yourself in a situation where you want each UDP segment to be analyzed by the CPU. In this case, you would need to issue this command:

```
BR-Switch#conf t
BR-Switch(config)#ip strict-acl-udp
```

ACL Flow Counters

If you want to gather statistics on how your ACLs are doing, you need to enable ACL statistics. This will provide two counters for your view on the CLI. The first is the flow counter. This will count one match of the session table (or processing by the CPU) of a given session (same protocol, source IP address, source port, destination IP address, and destination port). If the match continues beyond two minutes, the flow count will increase by one again. The idea of the flow counter is to give you an approximate view of the unique session flow of the traffic hitting your ACL (and each of its individual lines). The second counter is the packet counter. This is an exact accounting of how many packets are being matched to the whole ACL, and each individual line in the ACL.

It is important to note that ACL statistics will only function using Flow-Based ACLs. If you are using Rule-Based ACLs, and you enable ACL statistics, the ACLs will automatically become Flow-Based.

To activate ACL statistics, enter the following command:

```
BR-Switch#conf t
BR-Switch(config)#enable-acl-counter
```

Now, you can see the counters by issuing this command:

```
BR-Switch#show access-list 1
Standard IP access list 1 (Total flows: 148, Total packets:
13,324)
deny host 10.1.2.3 (Flows: 56, Packets: 4,973)
permit any (Flows: 92, Packets: 8,351)
```

You can see that host 10.1.2.3 has attempted several times to get through, but the ACLs have denied it. If you're debugging, you may want to reset the counters to start your analysis with an even baseline. To do this:

```
BR-Switch#clear access-list 1
```

Now, don't worry. You're not erasing access-list 1 (that would be “no access-list 1” issued in the global config). You're just clearing the counters.

You have one other analysis option for ACLs. This is the “log” keyword. This keyword generates syslog and SNMP-trap information when packets are matched on an ACL line. Let's look at an example:

```
BR-Switch#conf t
BR-Switch(config)#access-list 1 deny host 10.1.2.3 log
BR-Switch(config)#access-list 1 permit any
```

Notice that we've added the keyword “log” to the end of the first line in the ACL. When the first packet is received by 10.1.2.3, the switch will start a five-minute timer and count all of the packets from 10.1.2.3 within that five minutes. When the timer expires, it sends a single message to the log. This message contains the number of packets received on this ACL line in the last five minutes.

All of the ACL analysis options I've shown you in this section have the same two disadvantages. First, they automatically set your ACLs to Flow-Based. This adds an extra burden to the switch's CPU. Second, they require additional processing by the switch's CPU. Long story short, these functions are typically used for troubleshooting only, and should not be globally enabled on production systems.

MAC Filters

You may even find that you need to filter traffic not by TCP or UDP (Layer 4), not by IP (Layer 3), but by MAC (Layer 2). You have the option through a feature called MAC filters. These work very similarly to ACLs, and the syntax is also a two-step process. First, you define the list. Then, you apply it.

```
BR-Switch#conf t
BR-Switch(config)#mac filter 1 deny any any etype eq 0806
BR-Switch(config)#mac filter 1 deny any 0180.c200.0000
0000.0000.0000
BR-Switch(config)#mac filter 1 deny any any etype lt 0600
BR-Switch(config)#mac filter 1 permit any any etype eq 0800
```

The MAC filter uses the syntax of:

```
mac filter <number> <permit or deny> <source MAC address(es)>
<destination MAC address(es)> <type>
```

The first part of the syntax looks familiar, I'm sure. It starts to get different when you start talking about source and destination addresses. Here, you use a MAC address, not an IP address. The form is in hexadecimal, grouped in three four-digit groups (e.g., “0123.4567.89ab”). As with ACLs, you may use

the keyword “any” to signify any address. You may also use a wildcard mask. This is used the same way. A “1” signifies a bit that may change. A “0” signifies a bit that will not change. There's no “host” option.

The <type> section matches the Ethernet Length/Type Field (see [Chapter 1](#)). We start by defining which type of Ethernet frame we are looking for. In the above example, we're using “etype,” which would be the IEEE 802.3 frame described in Chapter 1. We may also specify “llc” (IEEE 802.2) or “snap.” Next, we can use “eq” (equal), “gt” (greater than), “lt” (less than), and “neq” (not equal) followed by a specific hexadecimal number. This would be the number we expect to find in the Length/Type Field.

In the example above, the first line denies all traffic with a type of 0806. This is ARP. The next line denies any traffic with a destination address of 0180.c200.0000. This is the special multicast address that Spanning Tree Protocol uses to send BPDUs. This filter will prevent BPDUs from being received. The next line denies any traffic with a type less than 0600, which would be any IEEE 802.2 LLC frames. Finally, we're permitting any traffic with a type of 0800, which is IP.

The filter does no good until it's applied. It is applied to the individual interface with the following command:

```
BR-Switch#conf t
BR-Switch(config)#int e 1
BR-Switch(config-if-e1000-1)#mac-filter-group 1
```

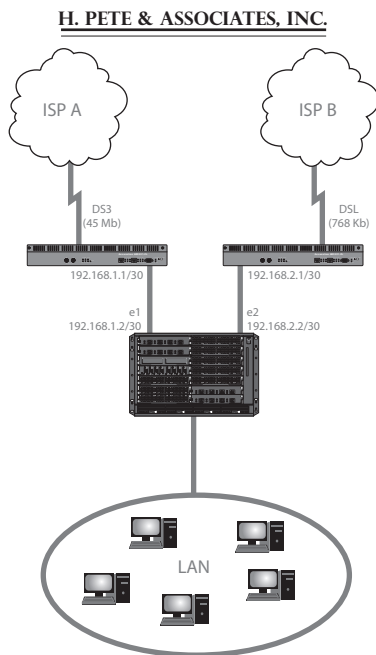
Notice that we didn't specify whether this was inbound or outbound. All MAC filters are inbound. The syntax is simply “mac-filter-group” followed by the filter number (in this case, “1”). All frames that match the filter are permitted through. All frames that do not match the filter are dropped.

Access Control Lists: Not Just For Security

I keep plugging this idea, don't I? The fact is, the deeper you go into your switch's function, the more you're going to find that ACLs can do so much more for you than simply permit or deny traffic. We'll touch on a few of these things in this section. Remember as we go through that an ACL is simply a list to match traffic. It does not inherently block or permit traffic (unless applied to do so).

Policy-Based Routing (PBR)

Policy-Based Routing (PBR) can be used for many things. I'm going to touch on one of the more common things here. We discussed early on that a Layer 3 switch has a routing table. This routing table is used for all traffic coming through that switch. As your infrastructures become more sophisticated, you may find that you'll want to use one “next-hop” router for some traffic, but a different next-hop router for other traffic.



H. Pete & Associates, Inc. does a lot of Internet communication with its partner, Globex Corporation. Globex Corporation hosts a series of servers that are all over the Internet (they're constantly expanding). The unique feature of Globex's servers is that communication always takes place on TCP 31416. To accommodate this communication, H. Pete & Associates bought a dedicated DS-3 (45 Mbps) circuit to the Internet through ISP A. It was also obvious that there would be employees who might need to perform other functions on the Internet (e.g., research, personal, etc.). It was mandated that this DS-3 only be used to communicate with Globex Corporation. For all other communication, H. Pete & Associates has purchased a DSL line (768 Kbps) to ISP B.

How can we fill the requirement to send only Globex-bound traffic through the DS-3, and all other traffic through the DSL line? This is where Policy-Based Routing can help. We need to perform a few steps to enable it:

1. Create an ACL that matches the traffic requiring PBR
2. Create a route map
3. Match the route map to the ACL
4. Define the policy
5. Apply the route map (globally or locally)

Step 1, we need to create an ACL that matches the traffic requiring PBR. Well, the unique feature of communication with Globex Corporation's servers is that they all listen on TCP 31416. If we create an ACL that matches any TCP traffic bound for a destination port of 31416, that should do it:

```
BR-Switch#conf t
BR-Switch(config)#ip access-list extended Globex
BR-Switch(config-ext-nacl)#permit tcp any any eq 31416
```

This is not going to be used to permit or deny traffic. It is being used to define traffic. In this case, it's matching all traffic that has a destination TCP port of 31416 (the unique characteristic of all of Globex's servers).

Step 2, we need to create a route map. It's beyond the scope of this book to go too deep into route maps, but the idea is that they provide a way to manipulate a switch's routing table (beyond simple static routes). Let's start by creating the route map:

```
BR-Switch#conf t
BR-Switch(config)#route-map GlobexRoute permit 10
BR-Switch(config-routemap GlobexRoute)#
```

Route maps are defined in the Global config. Each route map has a unique name. Next, we need a “permit” or “deny.” This describes whether the routing table will learn the route (as a result of the route map) or whether it will not. In this case, we want it to learn the route. And finally, the ending number is an *instance* number. A single route map can have several instances. Each are applied in sequential order. For this example, we will only have one instance. I chose “10,” as it's a safe number (as I may want to add instances prior to this one in the future).

Step 3, we now have to match the ACL with the route map we've created. While still in the Route-map sub-config, we enter this line:

```
BR-Switch(config-routemap GlobexRoute)#match ip address
Globex
```

The **match** command is used to say “match the IP traffic that is defined by this ACL.” In this case, we've told it to match IP traffic that is defined by our Extended IP ACL “Globex.”

Step 4, define the policy. Now that we've told it what to look for, what are we going to tell it to do? In this case, we want to tell the traffic to use ISP A as their default gateway, so that it will use the DS-3. The IP address of the DS-3 router is 192.168.1.1. While still in the Route-map sub-config, we issue this command:

```
BR-Switch(config-routemap GlobexRoute)#set ip next-hop
192.168.1.1
```

The **set** command is used to define the “next-hop” for the traffic. In this case, we specified 192.168.1.1.

Now, we're all done, right? Not yet. Route maps are like ACLs in one respect. You can define them, but until you apply them, they do nothing. So we go to Step 5, applying the route map. As the step states, we can apply this either globally or locally. In this case, we want to apply it globally, so we go to the Global config:

```
BR-Switch#conf t
BR-Switch(config)#ip policy route-map GlobexRoute
```

Now, to make this function the way it was specified, we actually have one more little matter to resolve:

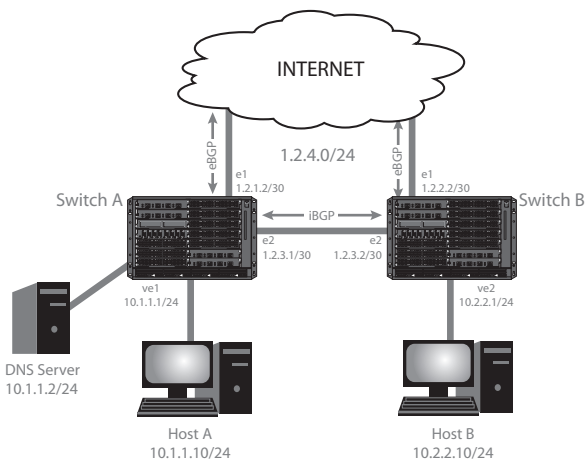
```
BR-Switch#conf t
BR-Switch(config)#ip route 0.0.0.0 0.0.0.0 192.168.2.1
```

Do you see what we just did? We set a default gateway for the router to point all traffic out 192.168.2.1. That's the DSL router. Will that defeat our route map that we just set up? No. The route map will be processed first. If the traffic does not match the ACL (e.g., it doesn't have a destination port of TCP 31416), it will be routed using the routing table. The routing table now has a default gateway that points to the DSL router. If traffic does have a destination TCP port of 31416, it will be given a next hop of 192.168.1.1 (the DS-3 router). This completes the process.

This is a very simple example of how policy-based routing might be used. The important thing to take away from this is that ACLs can be used to simply define traffic, not regulate it.

Network Address Translation (NAT)

Here is another example of ACLs being used to define traffic. Network Address Translation (NAT) was mentioned in Chapter 2. It doesn't require the use of ACLs, but it can be used with ACLs. Let's first go over how to configure traditional NAT.



The DNS server will not only need to get out to the Internet, but the Internet will need to be able to come to it. This requires a one-to-one relationship, which, as you may recall from Chapter 2, describes true NAT. There are two steps involved in this. The first step is to define the translations. We're advertising a public IP address range of 1.2.4.0/24. Let's use 1.2.4.6 as our public address for the DNS server:

```
BR-SwitchA#conf t
BR-SwitchA(config)#ip nat inside source static 10.1.1.2
1.2.4.6
```

The command tells the switch to translate the packets with a source address of 10.1.1.2 with the public address 1.2.4.6. Likewise, the switch also knows to translate inbound packets for 1.2.4.6 to 10.1.1.2. But we haven't activated the NAT yet. This is the second step. To activate the NAT, we need to define the router interfaces involved as either being on the *inside* or the *outside*. On this switch, we will want to designate interface ve 1 as inside (this is the router interface facing the DNS server), but we will want to set e 1 and e 2 as outside. In other words, when traffic from the DNS server leaves e 1 or e 2, NAT will be applied (e.g., the address will be translated). To do this, we enter the following:

```
BR-SwitchA#conf t
BR-SwitchA(config)#int ve 1
BR-SwitchA(config-vif-1)#ip nat inside
BR-SwitchA(config)#int e 1
BR-SwitchA(config-if-e1000-1)#ip nat outside
BR-SwitchA(config)#int e 2
BR-SwitchA(config-if-e1000-2)#ip nat outside
```

The static NAT is now configured. The DNS server can get out to the Internet, and the Internet can reach it as well.

Now, let's say we need to configure PAT (Port Address Translation). We'd like the entire 10.1.1.0/24 network (except for 10.1.1.2, of course) to share a single public IP address, 1.2.4.8. First, we need to create an ACL to define the traffic that we would like to NAT:

```
BR-SwitchA#conf t
BR-SwitchA(config)#ip access-list extended Net10NAT
BR-SwitchA(config-ext-nacl)#deny ip host 10.1.1.2 any
BR-SwitchA(config-ext-nacl)#permit ip 10.1.1.0 0.0.0.255 any
```

Hey, why is there a "deny" line? Well, we wanted to make sure that our DNS server doesn't participate in this pool. After all, the DNS server has its own static NAT address. The second line clearly permits all other addresses from 10.1.1.0/24. Remember that this is not permitting or denying traffic. This is "matching" or "not matching" traffic. Traffic with a source IP address of 10.1.1.2 will not match (as shown by the "deny" line). Traffic with a source IP address in the 10.1.1.0/24 network will match (as shown by the last line).

Next, we need to define the *pool* of addresses that the source may translate to. In this case, the pool only has one address (1.2.4.8). This still needs to be represented as range.

Here's the command:

```
BR-SwitchA#conf t
BR-SwitchA(config)#ip nat pool PATAddress 1.2.4.8 1.2.4.8
```

We've created a pool of addresses that is a range from 1.2.4.8 to 1.2.4.8. In other words, a range of one address. You can certainly use multiple addresses, if you would like. When using a range of addresses, the switch will choose which address to translate using a round-robin technique (e.g., first translation gets the first address in the range; the next translation gets the next address in the range; etc.). In this example, we'll just use one address.

We've created the ACL defining what traffic we want to get translated. We've defined the "pool" of addresses that the traffic may be translated to. Now we need to hook them together:

```
BR-SwitchA#conf t
BR-SwitchA(config)#ip nat inside source list Net10NAT pool
PATAddress overload
```

We're using the same command we used for static NAT, but this time, we've told it to translate the source traffic from an ACL ("list"). Then, we specified the name of the ACL ("Net10NAT"). Next, we told it to use a "pool" of addresses, and we've given it the name of our defined pool ("PAT Address"). But what's "overload?" Overload is the keyword that tells the switch to use PAT. If that keyword is not there, it will perform NAT (one-to-one) on the first address that comes through, and the rest will fail.

Normally, we would also have to define which interfaces are NAT "inside" and "outside," but we've already done that. Host A, and all of his 10.1.1.0/24 counterparts, may now get out to the Internet using the shared PAT address 1.2.4.8.

Now, let's throw a monkey wrench into this happy scene. Let's say that Host A needs to talk to Host B, and it's important that Host A's real address remains intact (e.g., it should not be translated). But e 2 on Switch A is defined as a NAT outside interface. What can we do?

ACLs to the rescue! What if we create a new ACL that looks like this:

```
BR-SwitchA#conf t
BR-SwitchA(config)#ip access-list extended NewNet10NAT
BR-SwitchA(config-ext-nacl)#deny ip host 10.1.1.2 any
BR-SwitchA(config-ext-nacl)#deny ip any 10.2.2.0 0.0.0.255
BR-SwitchA(config-ext-nacl)#permit ip 10.1.1.0 0.0.0.255 any
```

Looks like the old "Net10NAT" ACL, doesn't it? Except we've added one more important line. The line "deny ip any 10.2.2.0 0.0.0.255" will not match traffic with a destination address in the 10.2.2.0/24 network. Therefore, when the NAT is applied, it will not translate this traffic.

Well, that is, after we do this:

```
BR-SwitchA#conf t
BR-SwitchA(config)#no ip nat inside source list Net10NAT pool
PATAddress overload
BR-SwitchA(config)#ip nat inside source list NewNet10NAT pool
PATAddress overload
```

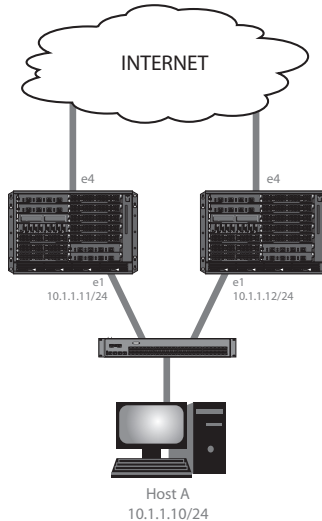
We had to get rid of the old NAT line, and we needed to give it a new one (with the new ACL). Now, everyone's happy! The DNS server is getting out to the Internet, and the Internet is getting to it. Host A and his 10.1.1.0/24 buddies are getting out to the Internet. And Host A can talk to Host B without getting translated. See how well ACLs and NAT can work together?

Redundancy: Virtual Router Redundancy Protocol (VRRP)

Part of a network engineer's job is the never-ending struggle to eliminate single points of failure. Redundancy is the answer, and thanks to RFC 3768, we have a non-proprietary solution to this problem.

What is VRRP?

Virtual Router Redundancy Protocol (VRRP) is a protocol designed to allow two, or more, routers to share a virtual IP address. This allows for default gateway redundancy. Let's look at an example:



Which router should Host A use as its default gateway? Router A? But what if it goes down? We'd have to manually change the default gateway for Host A to point to Router B. That's fine if you have only one host, but what if you have several hundred?

VRRP allows us to configure a virtual IP address that both routers (in this example) can share. Only one will answer for it at a time. The two routers will send “hello” messages back and forth to each other. When the active router fails, the other will assume the responsibility. Let's see how we would set this up.

Configuring VRRP

To set up VRRP, you start out like you're setting up a routing protocol. You start by defining VRRP in the Global config:

```
BR-RouterA#conf t
BR-RouterA(config)#router vrrp
```

Next, you need to configure the interface on Router A:

```
BR-RouterA#conf t
BR-RouterA(config)#int e 1
BR-RouterA(config-if-e1000-1)#ip address 10.1.1.11/24
BR-RouterA(config-if-e1000-1)#ip vrrp vrid 1
BR-RouterA(config-if-e1000-1-vrid-1)#owner track-priority 20
BR-RouterA(config-if-e1000-1-vrid-1)#track-port e 4
BR-RouterA(config-if-e1000-1-vrid-1)#ip-address 10.1.1.11
BR-RouterA(config-if-e1000-1-vrid-1)#activate
```

Let's step through these a bit. First off, we've got to give e 1 an IP address (the diagram shows that it was configured already, but I put it in there just to remind you it was there). The next thing we do is define the Virtual Router ID (VRID). We've given the VRID a number “1” arbitrarily. This is just to identify this specific VRID. A single interface could have more than one VRID.

Next, we defined this router as the owner. In VRRP, the owner is the device that answers for the address. All other devices configured with the same VRID are merely backups, waiting for the opportunity to take over, should the need arise. We've also specified a “track-priority.” By default, the owner of a link has a priority of 255 (priority is an 8-bit number). The track-priority (in VRRP) is the priority number that the router takes on, if the primary link (e 1 in this link) or a track-port fails. In our example, the track-priority is 20.

The next line, “track-port e 4” is a Brocade enhancement to the VRRP protocol. It allows you to track the failure of a port other than the primary port. Look at the diagram again. If e 1 of Router A went down, we'd want VRRP to fail over. That's obvious. But what if e 4 on Router A went down? The e 1 interface would still be up, but the router's function would be useless. This way, we've told VRRP to track e 4. If it goes down, the priority changes (and we'll likely configure Router B with a higher priority than 20; we'll get there in a minute).

Next, we have to give it the virtual IP address. Notice that we use the keyword “ip-address” (with a hyphen). More importantly, notice that the IP address is the exact same IP address we assigned to e 1. With VRRP, the virtual address must be the same address as the owner's physical address.

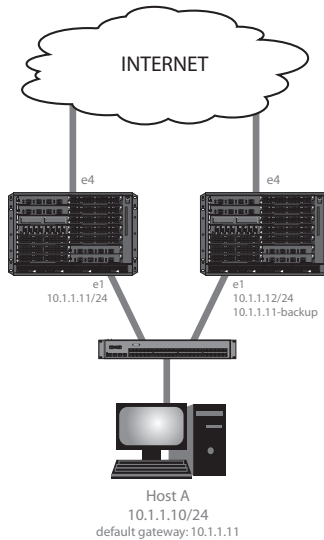
Finally, we have to activate the VRRP VRID. At this point, the interface is answering for the virtual address, but there's no router backing it up. Let's fix that.

```
BR-RouterB#conf t
BR-RouterB(config)#router vrrp
BR-RouterB(config)#int e 1
BR-RouterB(config-if-e1000-1)#ip address 10.1.1.12/24
BR-RouterB(config-if-e1000-1)#ip vrrp vrid 1
BR-RouterB(config-if-e1000-1-vrid-1)#backup priority 100
track-priority 19
BR-RouterB(config-if-e1000-1-vrid-1)#track-port e 4
BR-RouterB(config-if-e1000-1-vrid-1)#ip-address 10.1.1.11
BR-RouterB(config-if-e1000-1-vrid-1)#activate
```

Notice that the configuration is very similar to Router A's. The "backup priority" defines this router's priority number ("100"). As 100 is a lower number than 255 (Router A's priority), Router B will not answer for the virtual address. If Router A fails, Router B will take over. If Router A's e 1 fails, Router B will not get answers to its hello packets, and take over. If Router A's e 4 fails, its priority will change to 20, which is lower than 100. This means, again, Router B would assume command. If Router A's e 4 link were to be restored, Router A's priority would change to 255 again, and, as this is higher than 100, Router A would assume command of the virtual address.

Notice that Router B is also tracking its own e 4 interface. After all, if that goes down, Router B is useless. In this example, if Router B's e 4 fails, its priority will become 19. If Router A's e 4 fails as well, Router A will still answer for the virtual address, because its priority will be 20 (higher than 19). It won't do the host any good, but I just thought I'd walk you through the process.

Now we may assign 10.1.1.11 as Host A's default gateway, and we will be prepared not only if one router fails, but if one Internet link fails as well. The workstation will do nothing.



Hello packets are sent to the multicast address of 224.0.0.18 every second, by default. If this is too often, you may change the interval with this command:

```
BR-Switch#conf t
BR-Switch(config)#int e 1
BR-Switch(config-if-e1000-1)#ip vrrp vrid 1
BR-Switch(config-if-e1000-1-vrid-1)#hello-interval 10
```

This command changed the hello interval to 10 seconds. This will mean a slower failover, but less chatter. This value must be the same on all participating routers.

By default, a backup router will only wait 3.5 seconds before it determines that the owner is dead. This may be adjusted, but this value must be the same on all participating routers. If you would like to adjust this time, the command is:

```
BR-Switch#conf t
BR-Switch(config)#int e 1
BR-Switch(config-if-e1000-1)#ip vrrp vrid 1
BR-Switch(config-if-e1000-1-vrid-1)#dead-interval 5
```

If you are using RIP, you may not want RIP to advertise the virtual address, especially on the backup router. For this situation, use the following command:

```
BR-Switch#conf t
BR-Switch(config)#router rip
BR-Switch(config-rip-router)#use-vrrp-path
```

VRRP vs. VRRP-Extended

There is another flavor of VRRP available. It is Virtual Router Redundancy Protocol-Extended (VRRP-E). This protocol adds several features to VRRP.

- **Ping.** In VRRP, you cannot ping the virtual address. In VRRP-E, you can.
- **Owner/Backup.** In VRRP, you configure one router to be the “owner” and all others to be a backup. In VRRP-E, all routers are backup, and an election is held to decide who is the master (not “owner”). This allows greater flexibility over who is master.
- **IP Address.** In VRRP, the virtual address must be the same physical address as the owner's interface. In VRRP-E, the address may be any address in the same subnet.
- **Hello Packets.** In VRRP, hello packets are sent to 224.0.0.18. In VRRP-E, hello packets are encapsulated in UDP port 8888 and sent to 224.0.0.2 (“all routers” multicast address).
- **Track Priority.** In VRRP, a failure causes the router to assume the value of the track-priority configured. In VRRP-E, a failure causes the router's priority to be deducted by the track-priority. For example, if a router were configured with a priority of 100 (default), and a track-priority of 10, a failure would cause the router's priority to become 90 (100 - 10 = 90).

Configuring VRRP-Extended

Let's see how our previous example would look using VRRP-E:

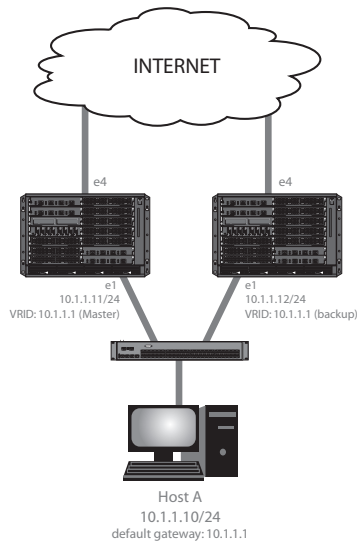
```
BR-RouterA#conf t
BR-RouterA(config)#router vrrp-extended
BR-RouterA(config-vrrpe-router)#exit
BR-RouterA(config)#int e 1
BR-RouterA(config-if-e1000-1)#ip address 10.1.1.11/24
BR-RouterA(config-if-e1000-1)#ip vrrp-extended vrid 1
BR-RouterA(config-if-e1000-1-vrid-1)#backup priority 160
track-priority 20
BR-RouterA(config-if-e1000-1-vrid-1)#track-port e 4
BR-RouterA(config-if-e1000-1-vrid-1)#ip-address 10.1.1.1
BR-RouterA(config-if-e1000-1-vrid-1)#activate
```

Notice that we gave it a different virtual address? We gave it “10.1.1.1.” We didn't specify a subnet mask. This is because we cannot assign a virtual address outside of the interface's own subnet. We've given it a priority of 160 with a track-priority of 20. This means that a failure will cause its priority to drop to 140. The advantage of the deducting track priority is that we can track several links. Each link will bring a deduction in the router's priority, but it may, or may not failover the virtual address. It all depends on who has the higher priority.

Now, let's look at Router B:

```
BR-RouterB#conf t
BR-RouterB(config)#router vrrp-e
BR-RouterB(config-vrrp-e-router)#exit
BR-RouterB(config)#int e 1
BR-RouterB(config-if-e1000-1)#ip address 10.1.1.12/24
BR-RouterB(config-if-e1000-1)#ip vrrp vrid 1
BR-RouterB(config-if-e1000-1-vrid-1)#backup priority 150
track-priority 19
BR-RouterB(config-if-e1000-1-vrid-1)#track-port e 4
BR-RouterB(config-if-e1000-1-vrid-1)#ip-address 10.1.1.1
BR-RouterB(config-if-e1000-1-vrid-1)#activate
```

Notice that both routers are configured as “backup.” In VRRP-E, all routers are backup. An election of priorities decides who is currently “master” (not “owner”). This status can be checked by using the command **show ip vrrp-extended**.



Now, we've assigned Host A with a default gateway of 10.1.1.1, and it will let the routers do the work, as to who handles the traffic. Router A has the higher priority, so it is currently master of the VRID. Router B is standing by, ready to take over should Router A fail or its priority drop below 150.

Traffic Shaping: Quality of Service (QoS)

Quality of Service (QoS) allows the administrator to prioritize traffic, and guarantee bandwidth for certain functions.

Why Quality of Service?

Some applications simply do not function well enough without certain guarantees. For example, an alarm system may require *immediate* delivery (regardless of what may be congesting traffic at the moment). Voice-over-IP (VoIP) may have strict requirements in regards to delay and jitter. Streaming multimedia may require a certain guaranteed level of bandwidth to be usable. This is why Quality of Service is used.

An administrator cannot constantly monitor their LAN in case some unforeseen user decides to congest all of your bandwidth by transferring a collection of DVD ISO images. Though backups have to be run, should they be run at the expense of valuable services having the throughput and delivery they need? In some environments, that's simply unacceptable.

A packet's priority is determined by a couple of things. First, if the packet is an 802.1q-tagged frame, it will have a 3-bit 802.1p priority value (see [Chapter 8](#)). Next, we check on the ToS/IP Precedence field in the IP header. The ToS portion is three bits as well. The larger of these two values is chosen as the packet's priority. It will be a number between 0 and 7.

From here, they are sorted into queues. There are four queues for QoS:

- qosp3 — This is the highest priority queue. This holds packets with a priority of 6 or 7.
- qosp2 — This is the second highest queue. This holds packets with a priority of 4 or 5.
- qosp1 — This is the second lowest queue. This holds packets with a priority of 2 or 3.
- qosp0 — This is the lowest queue. This holds packets with a priority of 0 or 1.

The queues are processed using one of two mechanisms. By default, they are processed by a weighted fair queue (WFQ) system. This system favors the higher queues, but permits traffic from all queues to be processed in a reasonable amount of time. You should consult Brocade's web site

(<http://www.brocade.com/>) for QoS information specific to your device.

Here's an example of a chassis switch's weighted fair queue breakdown:

- qosp3 — 80%
- qosp2 — 15%
- qosp1 — 3.3%
- qosp0 — 1.7%

The other processing mechanism is called *strict*, and with good reason. Essentially, all packets are processed in the higher queues before processing any traffic in the lower queues. Obviously, this could mean that traffic in qosp0, for example, may not be processed for quite some time. Strict processing should be used with caution.

Configuring QoS

By default, the processing mechanism uses weighted fair queuing. To change the processing mechanism to strict, issue the following command in the Global config:

```
BR-Switch#conf t
BR-Switch(config)#qos mechanism strict
```

To change the priority value of traffic coming into a certain switch port, use this command:

```
BR-Switch#conf t
BR-Switch(config)#int e 1
BR-Switch(config-if-e1000-1)#priority 7
```

This command tags all incoming traffic on e 1 with a priority of 7, placing the traffic in qosp3, the highest queue.

You may also assign a value to traffic coming from a given VLAN.

```
BR-Switch#conf t
BR-Switch(config)#vlan 10
BR-Switch(config-vlan-10)#priority 5
```

This command has labeled all traffic in VLAN 10 with a priority of 5, placing it in qosp2, the second highest queue.

The topic of Quality of Service is much broader than the scope of this book. For more information, consult Brocade's web site (<http://www.brocade.com/>).

Summary

- Access Control Lists define a method to match a certain kind of traffic
- ACLs may be applied to an interface to permit or deny traffic
- In applying an ACL to an interface, you must determine which direction the ACL will be applied (inbound or outbound)
- In ACLs, a wildcard mask is used to define an IP address range
- There are two types of IP-based ACLs: Standard and Extended
 - Standard ACLs match traffic using only the source IP address
 - Extended ACLs match traffic using the protocol, source IP address, source port, destination IP address, and destination port
- There are two types of configured ACLs: Numbered and Named

- Numbered ACLs use a number to define each individual ACL (group of rules), and the type of ACL
 - Standard IP ACLs use 1-99
 - Extended IP ACLs use 100-199
- Named ACLs are labeled with a user-determined name
- Named ACLs must be defined manually as either Standard IP or Extended IP
- ACL flow counters and logging may help troubleshoot ACL issues
- Rule-Based ACLs incur the lowest CPU utilization (and thus, best performance)
- Flow-Based ACLs incur a higher CPU penalty
 - All outbound ACLs are Flow-Based
- MAC filters act as a type of Layer 2 ACL, controlling traffic based on MAC addresses and frame types
- Policy-Based Routing provides an ability to direct traffic more granularly
- PBR uses ACLs to match the type of traffic to process
- ACLs may be used to more granularly decide what traffic should be translated (using NAT or PAT)
- The keyword “overload” denotes that the router is to use PAT translation
- VRRP and VRRP-E may be used to provide a redundant gateway
- Quality of Service (QoS) provides a method to prioritize the traffic that is processed
- QoS bases its decision on the higher of:
 - 802.1p priority (found in 802.1q-tagged frames)
 - ToS field (found in the IP header)
- QoS uses four queues to prioritize traffic (from highest to lowest):
 - qosp3
 - qosp2
 - qosp1
 - qosp0

Chapter Review Questions

1. In a switch's config, you see the line "access-list 123 permit ip any any." What kind of ACL is it?
 - a. Named Standard IP ACL
 - b. Numbered Extended IP ACL
 - c. Numbered Standard IP ACL
 - d. Named Extended IP ACL
2. What ACL keyword is needed to allow a TCP response to an initiated session?
 - a. host
 - b. any
 - c. syn-ack
 - d. established
3. Which ACL uses only the source IP address of a packet to permit or deny:
 - a. Extended IP ACL
 - b. Standard IP ACL
 - c. IPX ACL
 - d. MAC filter
4. You try to enter this line: **access-list 15 permit tcp any any**, but you get an error. Why?
 - a. For TCP, you must specify a source port and a destination port
 - b. You're using a Numbered syntax for a Named ACL
 - c. The number indicates a Standard IP ACL, yet the syntax is for an Extended IP ACL
 - d. You did not specify a name for the ACL
5. If a MAC filter denies a certain type of Ethernet frame, what will happen when this frame is received on an interface?
 - a. The switch will immediately drop the frame
 - b. The switch will store the frame in a queue for processing
 - c. The switch will forward the frame
 - d. The switch will generate a syslog and SNMP-trap message

6. qospp0 corresponds to which 802.1p and Brocade priority levels?
 - a. 6 and 7
 - b. 4 and 5
 - c. 2 and 3
 - d. 0 and 1
7. I have a one-line ACL: "access-list 101 deny tcp host 10.1.1.2 any eq 80." After applying it as an inbound ACL to my interface, no traffic is getting through. Why?
 - a. You should have applied it as an outbound ACL
 - b. All ACLs have an implicit "deny all" statement at the end; you have no permits
 - c. You're using Named syntax for a Numbered ACL
 - d. You're using Standard IP syntax for an Extended IP ACL
8. An Extended IP ACL uses what criteria to match traffic:
 - a. Source IP address
 - b. Destination IP address
 - c. Protocol and ports
 - d. All of the above
9. Which type of applied ACLs use the least CPU resources?
 - a. Rule-Based
 - b. Flow-Based
 - c. outbound
 - d. NAT
10. I specify the following source IP address and wildcard mask in an ACL: "172.16.32.0 0.0.7.255." What is the equivalent range in CIDR notation?
 - a. 172.16.32.0/24
 - b. 172.16.32.0/16
 - c. 172.16.32.0/21
 - d. 172.16.32.0/15

Answers to Review Questions

1. b. It's a Numbered ACL because it starts with "access-list" followed by a number. It's an Extended IP ACL because the number is between 100 and 199 (plus the format includes the protocol and destination address).
2. d. The established keyword permits TCP responses to sessions that have already been initiated.
3. b. The Standard IP ACL uses only the source IP address to match traffic.
4. c. The number of the ACL is 15. This would mean a Standard IP ACL. Yet, the protocol and destination address are specified.
5. a. Just like their ACL counterparts, if traffic matches a deny statement, it is immediately dropped.
6. d. qos0 is the lowest queue and corresponds to 0 and 1.
7. b. Always remember that there is an implicit "deny all" at the end of all ACLs. If you have no permits, it will not allow anything through once it is applied.
8. d. It uses protocol, source IP address, source port, destination IP address, and destination port.
9. a. Rule-Based ACLs are the favored ACLs for minimizing CPU resources and should be used whenever possible.
10. c. With a wildcard mask of "0.0.7.255," you've flagged the last 11 bits with a "1" (meaning, that the first 21 bits are marked with a "0"). This is an inverse mask to the subnet mask of 172.16.32.0/21.

Part Four: Layer 4-7 Switching (Load Balancing)

The following chapters are included in Part Four:

- [“Chapter 15: Server Load Balancing \(SLB\)”](#) starting on page 325
- [“Chapter 16: Session Persistence and Transparent Cache Switching”](#) starting on page 355
- [“Chapter 17: Firewall Load Balancing \(FWLB\)”](#) starting on page 373
- [“Chapter 18: Global Server Load Balancing \(GSLB\)”](#) starting on page 387



Server Load Balancing (SLB)

Welcome to Layers 4-7! We're on the home stretch now, and we're about to talk about one of my favorite tools in network engineering: the Brocade ServerIron ADX. This chapter will not only introduce you to the ServerIron ADX series, but it will introduce you to basic load balancing infrastructures. The next three chapters will go a deeper into more sophisticated configurations.

Now, entire books have been written on just the Brocade ServerIron alone. I will make no claims for this section to be comprehensive. However, after reading these chapters, you should feel comfortable designing many of the technologies listed, and you should certainly have a high-level understanding of the methods involved.

NOTE: The four chapters in this section refer to features and hardware specific to the Brocade ServerIron ADX series only, referred to hereinafter as the ServerIron.

The Brocade ServerIron

The ServerIron is an amazing tool. It's not really even fair to confine it to Layers 4-7. It really deals with Layers 2-7! It is, first, a switch. That's all. On top of that foundation, though, it has the ability to process and switch packets based on the OSI Layers all the way up to 7 (Application).

Overview of Features

Some common ways ServerIrons are used include:

- Server Load Balancing (SLB) (keep reading for more on this)
- Policy-Based Load Balancing (see [Chapter 16](#))
- Transparent Cache Switching (TCS) (see [Chapter 16](#))
- Firewall Load Balancing (FWLB) (see [Chapter 17](#))
- ISP Link Load Balancing (LLB)
- Global Server Load Balancing (GSLB) (see [Chapter 18](#))
- Application Security
- Application Availability

Brocade provides a stackable and a chassis version within the ServerIron ADX series, similar to the Layer 2 and Layer 3 switches. The stackable ServerIron ADX functions from a single CPU and includes capabilities of all modules listed in following section, but the stackable version does not have growth capabilities of the chassis versions. The ServerIron ADX chassis versions have growth potential and require that the modules listed in the following section be purchased separately. The minimum requirements for a chassis are one Application Switch Management Module, one Management Module, one Switch Fabric Module, and one Interface Module.

The Application Switch Management Module (ASM)

The ASM blade gives the ServerIron chassis some serious application processing power. Each ServerIron Application Switch Module (ASM8) has four dual-core processors dedicated to processing application traffic. They're the workhorses of the ServerIron chassis.



Depending on the chassis, the maximum number of core supported is 32 (four ASM8 blades in a 10000 Chassis). Chassis ADX ServerIrons have additional slots to provide interfaces.

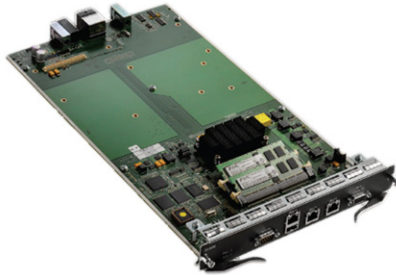
The Switch Fabric Management Module

ServerIron switch fabric modules provide up to 320 Gbps of switching capacity, providing scalability as I/O modules require more bandwidth.



The Management Module

ServerIron management modules have a dual-core processor, one console port, and one USB port, along with space for an optional mezzanine daughter card. The optional daughter card currently provides SSL capabilities, such as SSL termination and SSL-Proxy.

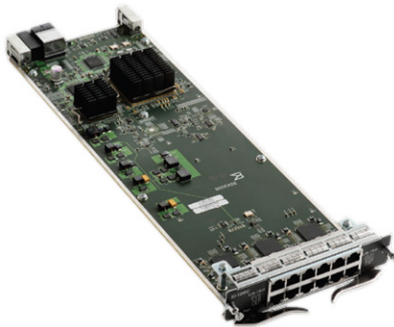


Interface Modules

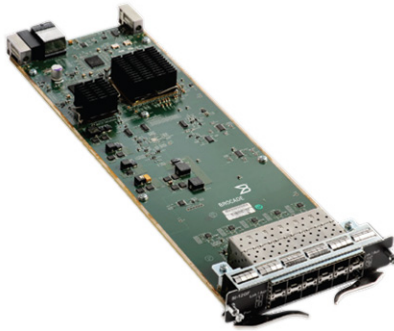
All line card packet processors support Layer 2-3 virtualization, and the ServerIron chassis can scale to support even higher I/O in the future as modular 40 Gbps and 100 Gbps line card interfaces become available.

Three configurations of ServerIron line cards are available:

- 12×1 Gbps copper (RJ45)



- 12×1 Gbps fiber (SFP)



- 4×10 Gbps fiber (XFP)



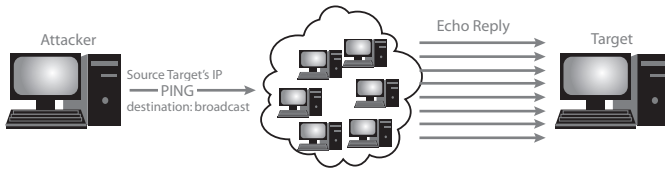
For more specific information on available models and specifications, please see Brocade's web site (<http://www.brocade.com/>).

Attack Protection

With all that processing power, and the fact that the ServerIron is analyzing through Layer 7 of each packet, it seemed only natural to include some protection against common attacks. The attacks we're going to cover here are basic Distributed Denial of Service (DDoS) attacks. These are attacks that are designed to make some remote service or function (often on the Internet) inaccessible.

Smurf Attack (ICMP)

This attack was called “smurf” after the program that was written to initiate the attack. The attacker chooses a target (typically a single IP address). The attack then sends a ping with its source IP address spoofed as the address of the target. This ping would be to a broadcast address of, preferably, a large network. Potentially hundreds of machines would receive the ping, and send their replies to the hapless target (as the original ping “appeared” to come from it).



The target becomes inundated with ICMP Echo Reply packets. This typically uses up all of the target's bandwidth (making the target remotely inaccessible). A truly distributed smurf attack would involve several different networks all replying to the same target.

SYN Attacks

This is a TCP-based attack. The idea is that the attacker is taking advantage of the TCP three-way handshake. Recalling from Chapter 2:

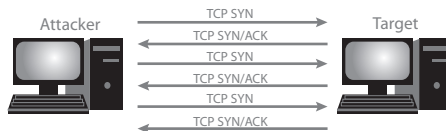
- The client sends a TCP SYN to the server
- The server sends a TCP SYN/ACK to the client
- The client sends a TCP ACK to the server

Thus completes a normal three-way handshake. With a SYN attack, the attacker sends many TCP SYN packets, but deliberately does not send the final ACK. In fact, often an attacker will spoof the source IP address, so that the reply will be sent to an address that cannot be reached (thus, incapable of sending a final ACK or any other message). The exchange stops here:

- The client sends a TCP SYN to the server
- The server sends a TCP SYN/ACK to the client

“What's the harm?” you ask. Every system has a finite number of TCP sessions it can process at one time. The server will keep that session open for quite a while waiting for that final ACK. It will eventually time out, but in the mean time, if the attacker sends many SYNs and all of them are stuck in a waiting state, soon the server will run out of resources to accept further connections.

SYN attacks are not usually focused on using up a target's bandwidth (like smurf attacks). It's more about using up the target's resources. Usual targets would include web servers and e-mail servers, but certainly any TCP service is potentially vulnerable.



The ServerIron has a far greater session capacity than most servers. That's what it's designed to do. It provides a couple of different methods to protect against this kind of attack.

The most basic is actually very similar to the protection against a smurf attack:

Configure syn-proxy in the global mode.

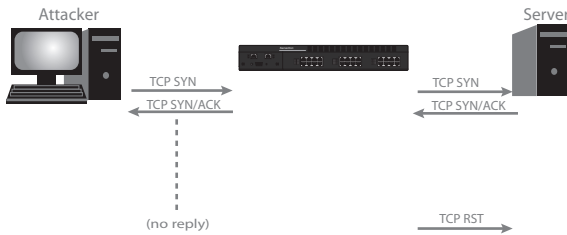
```
ServerIron(config)# ip tcp syn-proxy
Syntax: ip tcp syn-proxy
```

Enable syn-proxy on each interface handling inbound SYN requests (no change here).

```
ServerIron(config)#interface e 3/1
ServerIron(config-if-3/1)# ip tcp syn-proxy in
```

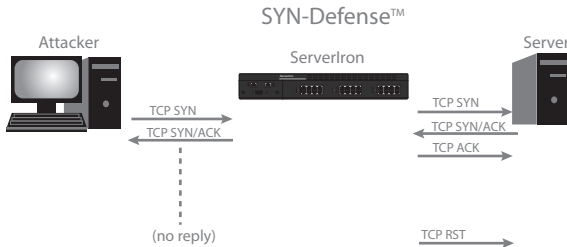
The next method of protection is called SYN-Defense™. There's a variation on SYN-Defense™, referred to as TCP SYN Protection for the stackable models. The concepts are very similar, and they really only differ in one area (which I will highlight).

The idea of SYN-Defense™ is that if the ServerIron is watching the sessions to the real servers anyway, why not monitor the time it takes to complete the three-way handshake?



By default, the ServerIron will wait eight seconds (after seeing that the SYN/ACK was sent). If it doesn't see an ACK from the client, it sends a TCP RST to the real server; thus, closing the session. This is how the stackable ServerIron performs the protection.

The chassis version is very similar, except it adds one more important feature:



The difference is subtle, but important. The ServerIron immediately finishes the three-way handshake on behalf of the client. When the three-way handshake is complete, the server moves the session from its pending connection queue to its established connection queue (which is much larger). If the client

does eventually send an ACK (within the time allotted), the ServerIron forwards it. The real server sees the ACK as a duplicate, and ignores it. If no ACK comes within the time allowed, the ServerIron sends a TCP RST to the real server to terminate the session.

On a chassis or a stackable, this protection is configured the same way. You may either enable it globally, or on an individual port basis. Either way, it must first be defined in the Global config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server syn-def 6
```

This command enables SYN-Defense™ (or TCP SYN Protection on stackables) globally, for all TCP sessions in the device. It has set the timer for six seconds. This is how long the ServerIron will wait from the time the TCP SYN/ACK is sent until it resets the session. The timer may be anything from 0 to 16 seconds. If it is set to 0, the protection is disabled.

If you are using a ServerIron chassis model, there may be circumstances in which you want to enable SYN-Defense™, but you don't want the ServerIron to send the final ACK on behalf of the client. This can be disabled with the following command:

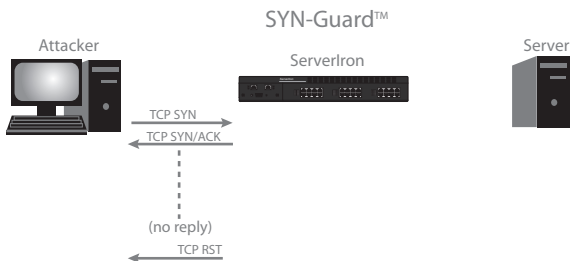
```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server syn-def-dont-send-ack
```

There may be situations where you only want this protection applied to traffic on certain interfaces. In this case, you must first configure it globally first (as we did above). Then:

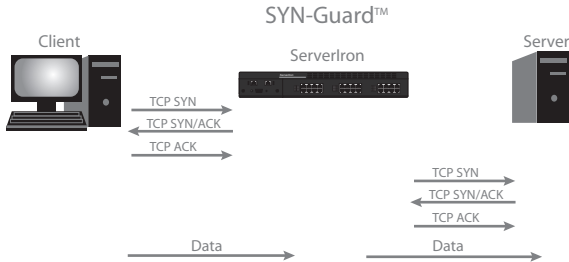
```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server syn-def 6
SLB-ServerIron(config)#int e 1
SLB-ServerIron(config-if-1)#syn-def
SLB-ServerIron(config)#int e 5
SLB-ServerIron(config-if-5)#syn-def
```

In this example, we've enabled the protection, but we've confined it to traffic that is flowing through e 1 and e 5. The interfaces defined will still use the global setting (six seconds, in this example) as its timer.

Another method to defend against SYN attacks is called SYN-Guard™.



Notice something significant about this picture? The real server never saw any traffic. The ServerIron sent a TCP SYN/ACK on behalf of the real server. The ServerIron will wait (by default) eight seconds. If it doesn't see an ACK back from the client, it sends a TCP RST to the client, and clears the session. The real server is oblivious that any transaction has taken place.



With SYN-Guard™ enabled, this is how you would normally see the traffic come through. Notice that the three-way handshake is only handed off to the real server when it has finished between the client and the ServerIron. The ServerIron is acting as a three-way handshake proxy. After the handshake is completed, traffic will pass through uninhibited for that session.

To configure this feature, use the following command in the Global config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#ip tcp syn-proxy 10
```

This has now globally-enabled SYN-Guard™ and set the timer to 10 seconds. This is how long the ServerIron will wait, after having sent the TCP SYN/ACK, to terminate a session.

For more information on current ServerIron protection features, please see Brocade's web site (<http://www.brocade.com/>).

Predictors

The predictor is the method that the ServerIron uses to balance the load across the configured servers. When an incoming packet comes into the ServerIron, it needs to make a decision as to which real server to forward the packet to. The ServerIron provides several different methods to do this:

- **Least connections.** This is the default predictor. This method looks at the current number of connections that each real server has, and picks the one with the least number of connections to forward the next incoming packet to. This method works well, if your real servers are all very similar in their capacity.

- **Round-robin.** This is the simplest of load balancing methods. The incoming packet goes to the next real server in line. When the real servers are bound to the VIP, they are bound in a certain order. It is that order that this predictor will use. If a real server is down, it will not be included in the round-robin rotation.
- **Weighted.** Here, the administrator defines a weight on each real server. This is useful if you have real servers of varying ability. Let's say you have three real servers. One is very new, and very fast. The other is okay. The third is aging rapidly. Ideally, you want more traffic to go to the new server, and less to go to the weakest. You could configure weights like this:
 - real server1 (fast) - weight 10
 - real server2 (medium) - weight 5
 - real server3 (slow) - weight 1

In this example, you would add all of the weights together ($10 + 5 + 1 = 16$). This tells you that out of every sixteen incoming packets the ServerIron receives, server1 will get ten, server2 will get five, and server3 will get one.

- **Server response time only.** The ServerIron uses a special algorithm to watch the response time of the real servers. It watches both health check responses and client responses. It gauges its time by watching TCP SYN and TCP SYN/ACK segments. The server with the fastest response time is usually selected, but if a slower server has not been selected in over one minute, the ServerIron will send the next packet to it. It will use this transaction to remeasure the server's response time.
- **Least connection and server response time weight (formerly "Response Time/Least Connections").** This is similar to weighted. The administrator will manually configure a "least connection weight" and a "server response weight" on each of its real servers. Based on the combination of these weights, the next real server will be selected. Configuring a higher weight biases the ServerIron toward that real server.
- **Least local connections (chassis only).** On an individual BP basis, the ServerIron chooses the real server with the least active client connections. Specifically, the ServerIron selects the real server with the fewest number of active client connections distributed from that specific BP.
- **Least local sessions (chassis only).** On an individual BP basis, the ServerIron chooses the real server with the least number of active client sessions distributed from a specific BP.
- **Dynamic Weighted Predictor.** TrafficWorks provides a dynamic weighted predictor that enables ServerIron to make load balancing decisions using real time server resource usage information, such as CPU utilization and memory consumption. The ServerIron retrieves this information (through the SNMP protocol) from MIBs available on the application servers. To achieve this capability, a software process in ServerIron, named SNMP

manager (also called SNMP client) is used. This process is different from the SNMP agent process (a.k.a. SNMP server process) on the ServerIron. A ServerIron can be configured as both an SNMP agent (that allows management of ServerIron through Network Management System), and an SNMP manager (that facilitates the new SNMP based predictor method). In addition, all the real servers must run the SNMP agent daemon and support MIBs that can be queried by the SNMP manager of the ServerIron. You can fine-tune how traffic is distributed across these real servers by enabling Dynamic Weighted Predictor on the ServerIron. SNMP dynamic predictor is presently not supported for IPv6 traffic.

Configuring SLB

We talked about Server Load Balancing (SLB) in Chapter 3. This is the default method ServerIrons use to load balance to real servers. This requires that all real servers are physically attached to the ServerIron, or that routing is engineered in such a way that all traffic to and from the real servers flows through the ServerIron. All traffic (pertaining to the VIP) must come into and go out of the ServerIron.

We'll need to configure real servers, a virtual server, and we'll go over configuring health checks as well.

Configuring “Real” Servers

Real servers are the servers that we want to load balance to. We need to define them within the ServerIron. We do this in the Global config. Each real server has its own sub-config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real-name rs1 10.1.1.11
SLB-ServerIron(config-rs-rs1)#port http
SLB-ServerIron(config-rs-rs1)#port http url "HEAD /"
SLB-ServerIron(config-rs-rs1)#server real-name rs2 10.1.1.12
SLB-ServerIron(config-rs-rs2)#port http
SLB-ServerIron(config-rs-rs2)#port http url "HEAD /"
SLB-ServerIron(config-rs-rs2)#port smtp
SLB-ServerIron(config-rs-rs2)#port smtp keepalive
SLB-ServerIron(config-rs-rs2)#server real-name rs3 10.1.1.13
SLB-ServerIron(config-rs-rs3)#port http
SLB-ServerIron(config-rs-rs3)#port http url "HEAD /"
SLB-ServerIron(config-rs-rs3)#port smtp
SLB-ServerIron(config-rs-rs3)#port smtp keepalive
```

Here, I've configured three real servers: rs1, rs2, and rs3. The name, by the way, does not need to correspond with any real information on the server. It's purely for your reference within the ServerIron config. It can be any alphanumeric string up to 32 characters. The IP address, of course, is a very important element.

Going back to our initial example, we configured several “port” statements under each real server. Each port statement reflects a protocol that we intend to use on that real server. For rs1, it's just HTTP. For rs2 and rs3, it's both HTTP and SMTP. We'll talk more about the “port http url” and “port smtp keepalive” commands in the health check section.

There are many other settings that may be configured for an individual real server. It's beyond the scope of this book to cover them all, but one more that you should be aware of is “max-conn” (short for “maximum connections”). The ServerIron sets a default value for maximum connections (this varies depending on model and software). This is the maximum number of connections the ServerIron will sustain on an individual real server. When the maximum is reached, no new traffic will be passed to that real server. If the default is more than you know your real server can handle, you may change the value:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real rs1
SLB-ServerIron(config-rs-rs1)#max-conn 150000
```

Here, we've changed the maximum connections allowed for rs1 to be 150,000 (defaults are often 1,000,000 or more).

Configuring Virtual Servers

The virtual server is where the VIP (Virtual IP) is configured. Here, you assign the VIPs address, decide which ports it will be listening on, determine which predictor method is to be used, and which real servers will be used in the load balancing. Virtual servers are configured in much the same way as real servers. Here's an example:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual-name www 1.2.3.4
SLB-ServerIron(config-vs-www)#port http
SLB-ServerIron(config-vs-www)#predictor round-robin
SLB-ServerIron(config-vs-www)#bind http rs1 http
SLB-ServerIron(config-vs-www)#bind http rs2 http
SLB-ServerIron(config)#server virtual-name mail 1.2.3.10
SLB-ServerIron(config-vs-www)#port smtp
SLB-ServerIron(config-vs-www)#port http
SLB-ServerIron(config-vs-www)#predictor least-conn
SLB-ServerIron(config-vs-www)#bind smtp rs2 smtp
SLB-ServerIron(config-vs-www)#bind smtp rs3 smtp
SLB-ServerIron(config-vs-www)#bind http rs3 http
```

Here, we've configured two virtual servers. The first was named “www.” It tells the ServerIron to create a VIP (1.2.3.4). The next line “port http” tells the ServerIron to listen on TCP 80 for that VIP. The next line defines which predictor should be used. In this case, we chose “round-robin.” If nothing is chosen for the predictor, round-robin is the default.

Finally, we are binding real servers to the VIP. This requires a little extra explanation. The syntax is:

```
bind <virtual server port> <real server name> <real server
port>
```

It happens to be a coincidence that the virtual server's port and the real server's port are the same. This gives the user quite a bit of flexibility, though. Say, for example, you wanted the VIP to listen on TCP 9003, but you wanted the communication translated to HTTP before it hit your real servers. In this case, you would use "port 9003" in the virtual server config, and the bind statements would look like "bind 9003 rs1 http."

The second virtual server ("mail") is configured very similarly. It is a different VIP address ("1.2.3.10"), and it is listening on a different protocol ("smtp" or TCP 25), in addition to HTTP. We've configured a different predictor ("least-connections"). And the bind statements are obviously not the same. We're binding different servers ("rs2" and "rs3") to SMTP. Notice that we did not bind rs1. This server did not have a port defined for SMTP in its real server config. Also, we've bound rs3 to HTTP (let's say, that's the web UI for the mail servers). It's important to understand that a single virtual server could be listening on any number of ports. We've chosen two here, as an example. You could configure many more.

Configuring Health Checks

Before we begin, it is always important to keep in mind that when the ServerIron performs health checks, the source IP of the health check will be either the configured management IP of the ServerIron or a more local "server source-ip" (more on these later). If there is a source-ip configured that is within the same broadcast domain as the real server to be checked, the ServerIron will use this address as the source. If not, it will use its management IP address. This may be critical to know when troubleshooting failed health checks. If the ServerIron is using its management IP address, but the real server has no route to reply, the health checks will fail.

In Chapter 3, we talked about a Layer 3, Layer 4, and Layer 7 health check. When real servers are defined, the ServerIron will attempt an initial Layer 3 health check. It will attempt an ARP request for the address. This is sometimes referred to as a Layer 2 health check, since it requests the MAC address of the server. It follows the ARP request with a ping (ICMP Echo). If the real server is not bound to any virtual servers, it will not perform any further health checks until the ARP entry times out. At this point it will perform another ARP and ping. Nothing needs to be done to make these health checks active. They are automatically used. Although it is not recommended, they can be disabled, using the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#no-l3-check
```

When a real server port is bound to the virtual server, regular Layer 3, Layer 4, and (if configured) Layer 7 health checks initiate automatically. Layer 3 health checks are the most frequent. By default, Layer 3 health checks (ARP and ping) are performed every two seconds. Upon failure, it will retry four times before declaring a failed check.

Layer 4 health checks are performed based on the port that was bound. For example, when we created the virtual server “www,” we bound rs1 and rs2 to HTTP. The ServerIron will now be performing Layer 4 health checks on these two servers using TCP 80. To summarize what we covered in Chapter 3, the ServerIron will send a TCP SYN on port 80 to the real server. The real server will send a TCP SYN/ACK back to the ServerIron. The ServerIron will terminate the session by sending a TCP RST. This is a healthy health check.

Layer 4 health checks may also be disabled, but this is also not recommended:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#no server l4-check
```

Generally, Layer 7 health checks are disabled by default. They must be specifically requested in the real server's sub-config. Some well-known protocols automatically add a Layer 7 health check line. HTTP, for example, automatically adds an extra line to the real server config. When a user types “port http”, a second line is also added: port http url “HEAD /.” This defines a Layer 7 health check that performs an HTTP “HEAD /.” If the server's response is an HTTP Status code of 200-299 (or 401), the ServerIron considers the real server to be alive, and it will pass traffic to it. If it gets any other code (or if the operation fails in any other way), the real server will be considered down, and the ServerIron will not pass traffic to it. The HTTP command used for the health check may be changed to whatever the user chooses. Some common examples are:

```
port http url "GET /index.html"
port http url "GET /image.gif"
port http url "GET /some/specific/application.cgi"
```

But what about SMTP? We can't use a “port smtp url” command for SMTP. For SMTP, and many of the well-known ports, you need only add one line to the real server's config: “port smtp keepalive.” This tells the ServerIron that you'd like to perform a Layer 7 health check for that protocol. For SMTP, the ServerIron will look for the opening SMTP banner followed by a 220 status code. If it sees these, it will issue an SMTP “Quit” command, and reset the TCP connection. This is a successful Layer 7 health check for SMTP.

There are many more variations for Layer 7 health checks. For protocols that are not well-known, custom health checks may be created, but this goes beyond the scope of this book. For more information, consult Brocade's web site (<http://www.brocade.com/>).

Reassign Threshold

No matter how well we configure our health checks, there's always the probability that the service on the real server will fail between health checks. The Serverlrons have one more safeguard for that. This service is only available for TCP traffic, and it is not available in DSR configurations (because the Serverlron never sees the return traffic).

When a real server does not respond to (by default) 21 TCP SYN packets in a row, the Reassign Threshold is reached, and the real server's port is considered down. Each real server port has a Reassign Threshold assigned to it. It starts at 0. For each missed TCP SYN packet, the counter is increased. If the Serverlron sees a TCP SYN/ACK reply, the counter is reset to 0. If the counter makes it all the way to the threshold (21, by default), the real server port is considered down, and no new traffic will be sent to it.

The Reassign Threshold is disabled by default. It can be enabled with the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server reassign-threshold 25
```

This command not only enables the Reassign Threshold, but, in this case, it has set the threshold at 25 failed packets. This value may be anything from 6 to 4,000.

No-fast-bringup

An improvement was made to the way real server ports are added back into rotation. In the early days, a failed real server port had to pass a Layer 4 health check and, if one was enabled, a Layer 7 health check before being added back into rotation. The process now will check a Layer 4 health check, and, if it's successful, regardless of whether there is a Layer 7 health check enabled, it will add the real server port back into rotation. After adding it into rotation, if there is a Layer 7 health check enabled, it will perform that as well. This is to speed up the restore process. The assumption is that if it's passing Layer 4, it will pass Layer 7.

The problem is that if you are depending on your Layer 7 health check to determine the up or down status of your real server port, the Layer 4 health check may pass, and your real server port may receive traffic briefly, until the Layer 7 health check is tried. In this situation, it would be better to go back to the old system, and not activate the real server port until you're certain the Layer 4 and the Layer 7 health check have succeeded. You can do this with the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server no-fast-bringup
```

SLB show Commands

To troubleshoot SLB, there are three very handy commands. Here's the first one:

```
SLB-ServerIron#show server virtual
Virtual Servers Info
Name: www                               State: Enabled
IP:1.2.3.4: 1
Pred: round-robin                       ACL-Id: 0
TotalConn: 157541
Port   State   Sticky  Concur  Proxy  DSR   CurConn  TotConn
PeakConn
-----
default enabled NO      NO      NO      NO    0         0
0
http   enabled NO      NO      NO      NO    55        157541
12477
```

This command gives you information about the virtual server configured. You see its name, IP address, predictor, connection information, and so on. The second command is:

```
SLB-ServerIron#show server real
Real Servers Info
=====
State - ACT:active, ENB:enabled, FAL:failed, TST:test,
SUS:suspect,
      GDN:grace-dn, DIS:disabled, UNK:unknown, UNB:unbind,
      AWU:await-unbind, AWD: await-shutdown
Name: rsl                               State: Active
IP:10.1.1.11: 1
Mac: 0001.0203.0405                     Weight: 0                               MaxConn:
2000000
SrcNAT: not-cfg, not-op                 DstNAT: not-cfg, not-op                 Serv-
Rsts: 0
Port  St  Ms  CurConn  TotConn  Rx-pkts  Tx-pkts  Rx-octet
Tx-octet  Reas
-----
default UNB 0 0      0      0      0      0      0
0
http   ACT 0 0      0      0      0      0      0
0
Server Total 0      0      0      0      0      0
0
```

State of Real Server

The real server, and the individual real server ports, can be in one of several different states. For example:

The real server's state is declared in the field just to the right of the real server's name (above example: "State: Active"). It could be in one of the following states:

- **Enabled (ENB).** The real server is configured, but the real server has failed the ARP check. The ServerIron cannot reach this real server.
- **Failed (FAL).** The real server has failed the Layer 3 ICMP health check.
- **Testing (TST).** The ARP check has succeeded, but the ICMP check has not. We've sent an ICMP Echo Request, but we haven't yet received a reply.
- **Suspect (SUS).** The ServerIron keeps track of the time that elapses from when the real server receives a packet and when it responds. If this time increases to 3 or 4 seconds, the ServerIron will send a ping. If a reply is not received, another ping is sent, but the real server is put in "Suspect" state. If a reply is received, the real server is put into Active. If the maximum number of retried pings (a configurable parameter) fails, the real server is put into Failed state.
- **Grace-dn (GDN).** A force-shutdown command has been issued, and the ServerIron is gracefully disabling the real server.
- **Active (ACT).** Layer 3 health checks (both ARP and ping) have passed. The real server should be available to receive traffic.

State of Real Server Ports

Each individual real server port may be in a different state. The possibilities are:

- **Enabled (ENB).** This means the same as the real server Enabled state. The real server's been configured, but there is no ARP response.
- **Failed (FAL).** The real server port has failed a Layer 4 or Layer 7 health check.
- **Testing (TST).** The ServerIron is trying its Layer 4 or Layer 7 health check again. Layer 3 would still have been successful, but it is trying Layer 4 or Layer 7.
- **Suspect (SUS).** As in the real server Suspect state, the ServerIron keeps track of how long a real server responds between an incoming packet and a reply. If this gap grows to 3 or 4 seconds, the ServerIron will send another Layer 4 health check. If a Layer 7 health check is enabled, it will send that, too. This happens regardless of when the next scheduled health check is.
- **Grace-dn (GDN).** The force-shutdown option has been issued, and the real server's services are in the process of being gracefully disabled.

- **Active (ACT).** The real server port has passed Layer 3, Layer 4, and, if enabled, Layer 7 health checks, and is ready to receive traffic.
- **Unbind (UNB).** The real server port is not bound to any virtual server.
- **Await-Unbind (AWU).** Someone has unbound this real server port from the virtual server, but the real server port was still servicing existing sessions. The ServerIron is gracefully expiring these sessions before officially unbinding the port. No new traffic is received in this state.
- **Await-shutdown (AWD).** Similar to Await-Unbind. A shutdown was issued to a real server, and the ServerIron is in the process of gracefully disabling services.
- **Disabled (DIS).** The real server port has been disabled in the configuration.

The real server port fields are:

- **Port.** The real server ports that are defined in the config
- **St (State).** The state of the real server port (see above)
- **Ms (Master Port State).** This pertains only to track ports (beyond the scope of this book)
- **CurConn (Current Client Connections).** The number of client connections currently being serviced by the real server port; a “connection” consists of two sessions: the client-to-server session and the server-to-client session
- **TotConn (Total Client Connections).** The total number of connections this real server port has serviced since: a) the real server port was bound; b) the ServerIron was last rebooted; or c) the counters were last cleared
- **Rx-pkts (Received packets).** The number of packets the ServerIron has received from the real server; this does not include health check responses; in DSR bindings, this statistic would show nothing (as the ServerIron never sees the responses)
- **Tx-pkts (Transmitted packets).** The number of packets the ServerIron has sent to the real server port; this does not include health checks
- **Rx-octet (Received octets).** The number of “octets” (8-bit groups, or “bytes”) the ServerIron has received from the real server; this does not include health check responses; in DSR bindings, this statistic would show nothing (as the ServerIron never sees the responses)
- **Tx-octet (Transmitted octets).** The number of “octets” (8-bit groups, or “bytes”) the ServerIron has sent to the real server port; this does not include health checks
- **Reas (Reassign Counter).** The number of consecutive times a real server port has failed to respond to a TCP SYN packet; this counter does not apply to a DSR binding (as the ServerIron would never see a response)

To get a high level of which real servers are bound to virtual servers and what their current state is, use the third helpful command:

```
SLB-ServerIron#show server bind
Bind info
Virtual server: www                               Status: enabled IP:
1.2.3.4
      http -----> rs1: 10.1.1.11,  http (Active)
                      rs2: 10.1.1.12,  http (Active)
```

Notice that the output of this command shows the virtual server, the name, and the VIP address. It then details each of the bound real servers, their name and address, the bound port, and finally, the current state of the bound port.

Configuring Source NAT

Let's look back at Chapter 3. Source NAT is used when the load-balanced servers are not directly connected to the load balancer (or when all traffic to and from the real servers is not routed through the load balancer). The load balancer can still hand off to the real server, but the return traffic must come back through the load balancer to be properly translated to the VIP. To do this, Source NAT makes the packets that go to the real server appear to come from the ServerIron itself. The real server returns the packets to the ServerIron, and the ServerIron translates it to the proper VIP and sends it back to the outside client.

The first thing that needs to be configured is a “source-ip” address. This is a virtual address that you assign to the ServerIron so that it can reach the remote servers for health checks and for passing traffic. This will also be the IP address that appears as the source address for traffic that is passed to the real server (that is using Source NAT). The ServerIron limits the number of source IPs to 64.

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server source-ip 10.20.30.3
255.255.255.0 10.20.30.1
```

Notice that we've specified an IP address, subnet mask, and a default gateway. Next, we need to enable Source NAT. This can be either enabled globally or locally. If you enable it globally, it will affect all real servers. Use the following command from within the Global config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server source-nat
```

If you have some real servers you would like to use Source NAT, and others that you do not, you'll want to enable Source NAT locally. This is done in the real server sub-config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real rs1
SLB-ServerIron(config-rs-rs1)#source-nat
```

That's all there is to it. You are now using Source NAT.

Configuring Direct Server Return (DSR)

Once again, let's reflect on what we learned in Chapter 3. Direct Server Return (DSR) is an alternate solution to the problem of not having your real servers directly connected to the load balancer. In this scenario, the ServerIron passes traffic to the real server, and the real server responds directly back to the client (without the ServerIron ever seeing the return traffic). How is it able to do this?

For DSR to work, on each real server, you must have a loopback address configured. This loopback address must be the exact same IP address as the VIP. You also must make sure that your real server does not answer ARP requests for this address. This process is different for every real server's operating system. For help, consult Brocade's web site (<http://www.brocade.com/>).

Once the loopback address is configured, the ServerIron, when it receives an incoming packet for the VIP, changes only the destination MAC address. It changes the address to be the MAC address of the real server it wishes to send to. When the real server receives the packet, it sees that it is destined for the VIP address. Because the real server has a loopback with the same IP address as the VIP, it knows to accept the packet. In addition, since the packet was received to that address, the real server will reply as having come from that address. To the end user, it looks as if they have communicated with one host.

To configure the ServerIron, there are two more steps. First, you need to configure a "source-ip", so that the ServerIron will be able to reach the remote server for health checks and to pass traffic. Remember, in DSR, the ServerIron only changes the destination MAC address. If it cannot reach the real server in the same broadcast domain, the ServerIron is not going to be able to pass traffic. This means that the ServerIron is going to need at least one physical interface that is in the same VLAN as the real servers. The "source-ip" is configured this way:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server source-ip 10.20.30.3
255.255.255.0 10.20.30.1
```

Finally, enable DSR in the virtual server. This function is unique to the virtual server. In fact, it's unique to a specific port on the virtual server. Here's an example:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual-name www 1.2.3.4
SLB-ServerIron(config-vs-www)#port http
SLB-ServerIron(config-vs-www)#port http dsr
SLB-ServerIron(config-vs-www)#predictor round-robin
SLB-ServerIron(config-vs-www)#bind http rs1 http
SLB-ServerIron(config-vs-www)#bind http rs2 http
```

That's all you need to add. Now, when the virtual server “www” hands off to rs1 or rs2, it will use DSR. The health checks on rs1 and rs2 will know that DSR is being used as well. If the loopback does not exist on these real servers, they will fail the health check.

Does this mean that you could have one virtual server port (say, http) using DSR, and a different virtual server port (say, smtp) on the same virtual server that is not using DSR? Yes, that's what this means. DSR is unique to the virtual server port.

Configuring Multiple VIPs to the Same Real Server

There are times when you may want to bind the same real server port to different virtual servers. Let's say we have our virtual servers “www” and “mail”, but we want to create a new virtual server called “images”:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual-name images 1.2.3.50
SLB-ServerIron(config-vs-www)#port http
SLB-ServerIron(config-vs-www)#predictor least-conn
SLB-ServerIron(config-vs-www)#bind http rs1 http
```

This port is already bound to another virtual server.

```
SLB-ServerIron(config-vs-www)#
```

What happened? Oh, that's right. We've already bound HTTP on rs1 with virtual server “www.” What can we do? We need to use rs1. Can't it be bound to both?

Yes, but it requires an additional port: a shadow port. This is a fake port that we will add to the real server sub-config. Its sole purpose is to allow us to bind the same real server to different virtual servers on the same port. When choosing a shadow port, it's wise to pick a number that you know you aren't going to use later. For example:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real rs1
SLB-ServerIron(config-rs-rs1)#port 12345
```

Now wait a minute! Isn't the ServerIron going to try a Layer 4 health check going to TCP or UDP port 12345? Think back. What determines the Layer 4 health check? This only happens when a real server port is bound, right? Let's take a look at that process. Let's go back to “images”:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual images
SLB-ServerIron(config-vs-www)#port http
SLB-ServerIron(config-vs-www)#no port http translate
SLB-ServerIron(config-vs-www)#bind http rs1 12345
```

Okay, so you've bound HTTP to rs1's port 12345. Won't that mean that the virtual server will listen on TCP 80 (HTTP) and the real server will listen on TCP or UDP 12345? Wait a minute. We've added another line. What does that say? "no port http translate"? It means that we've specifically instructed the virtual server not to translate the port. This means that the virtual server will still speak to the real server using HTTP. The "port 12345" only means something within the confines of the ServerIron config.

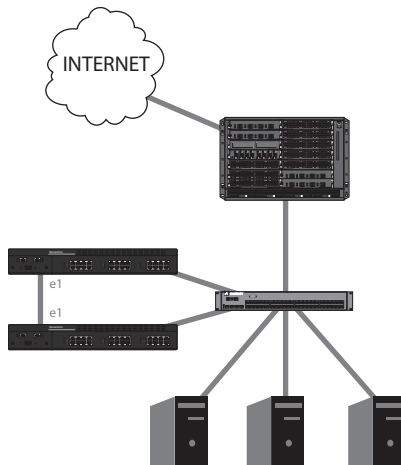
A real server's port may only be bound once. How can we get around this? By creating a fake port (or shadow port). When we bind the port, though, we need to make sure that the virtual server does not translate the port. Otherwise, the ServerIron will attempt to communicate to the real server on the fake port you've created.

You can create as many shadow ports as you need. In this example, we've just created one. We only needed to bind rs1's http port to one other virtual server. With additional shadow ports, though, we could bind rs1's http port to many different virtual servers. This will become important when we start to look at ServerIron redundancy.

Hot-Standby Redundancy

We always want to make sure that we eliminate as many single points of failure as we can. Since ServerIrons are an important part of the infrastructure, we need to be protected in the case of a failure.

One way we can do this is with Hot-Standby Redundancy. In this case, we would attach the servers to a central layer-2 switch (as opposed to a ServerIron directly). We would then provide two ServerIrons. We would connect each ServerIron to the same layer-2 switch, and then, to each other.



Hot-Standby uses an Active/Standby solution (see [Chapter 5](#)). One ServerIron is active. The other ServerIron simply waits for the first ServerIron to fail. The two ServerIrons check on each other via the “sync link”, the cable connecting their respective e 1 interfaces. Just so that it's clear, I chose e 1 arbitrarily. You may use any port you wish, and they do not have to match (e.g., e 3 on ServerIron, e 5 on the other). The sync link is used to send and receive hello messages between the two ServerIrons. It is also used to exchange state information. Hello messages flow 10 times per second over this link.

To configure Hot-Standby, we'll start by creating a VLAN that will house the sync link. On both ServerIrons, we'll configure VLAN 111 for the sync link. We will also disable Spanning Tree Protocol in this VLAN:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#vlan 111
SLB-ServerIron(config-vlan-111)#untag e 1
SLB-ServerIron(config-vlan-111)#no spanning-tree
```

Next, we should make sure that the configurations on both ServerIrons are identical. Same real servers. Same virtual servers. The only thing that should be different is their Management IP address.

Next, we'll need to determine what the chassis MAC address is of each ServerIron. On a chassis model, this can be found using the command “show chassis.” It will be listed as the “Boot Prom MAC:.” For a stackable model, it will be the MAC address of the lowest-numbered interface (usually e 1). The “show interface” command will give you this address. You choose one of these MAC addresses to be the master. Let's say you choose MAC address 0001.0203.0405. You configure this MAC address as part of enabling Hot-Standby. This is done with the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server backup e 1 0001.0203.0405 vlan-
id 111
```

Once we've issued this on both ServerIrons, Hot-Standby is enabled. One ServerIron will assume the active role (and answer for the defined MAC address). The other will wait for a failure. If a failure is detected over the sync link, the remaining ServerIron will attempt to communicate to its partner over the data link (the link used to communicate to the real servers), just to make sure that it wasn't just a sync link failure. If it was a sync link failure, hello packets will continue over this data link until the sync link is restored. If the ServerIron gets no response over the data link as well, the active ServerIron is considered inactive (failed).

In the event of a failure, the other ServerIron will assume the MAC address and active role for real and virtual servers. Because session state information is exchanged, this process should be completely transparent to both the real servers, and the clients. To check the status of the redundancy, you can use “show server backup” on either ServerIron.

If you want to assign one ServerIron to always be the active ServerIron, you can issue one other command (on the ServerIron you want to be active). If this command is on a ServerIron, that ServerIron becomes the active ServerIron.

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server backup-preference 5
```

Okay, what about the 5? Let's say the active ServerIron fails. We repair it, and bring it back up. Instead of just seizing control right away, the ServerIron will wait a certain number of minutes to make sure that its state information is accurate. The "5" is 5 minutes. This value can be anything from 5 to 30 minutes.

Symmetric Server Load Balancing

The big difference between Hot-Standby and Symmetric SLB is that Hot-Standby provides active/standby for the entire ServerIron where Symmetric SLB provides active/standby per virtual server. Each virtual server may have its own active/standby redundancy.

To configure Symmetric SLB, we should start by making certain that the session state information will be properly shared. This is configured at the port level. When a failover occurs on a stackable, existing sessions will need to be re-established by the client.

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server port 80
SLB-ServerIron(config-port-80)#session-sync
```

We've provided the ability to sync sessions for HTTP (TCP 80) traffic. Should any other port be participating in active/standby, this command ("session-sync") should be in its global "server port" config as well.

Active/Standby

To configure a virtual server as active/standby, you merely need to add one line to the virtual server config:

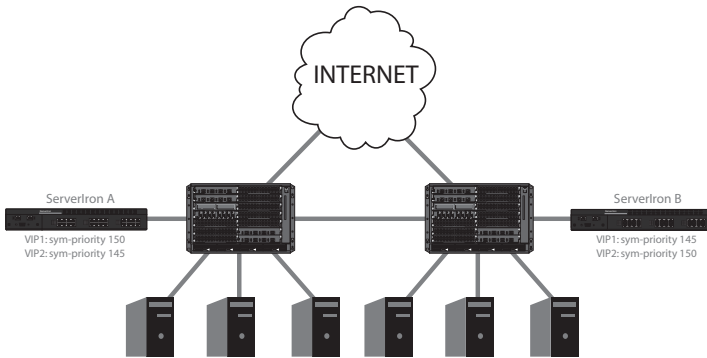
```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual www
SLB-ServerIron(config-vs-www)#sym-priority 200
```

The "sym-priority" command tells the ServerIron that it will be participating in Symmetric SLB with another ServerIron. The command sets the "priority" for this particular ServerIron (in relation to the virtual server "www"). The ServerIron will look for any other ServerIron that may have the "www" virtual server configured (more specifically, it will look for the IP address of the virtual server). Out of all the ServerIrons that may have this virtual server configured, the ServerIron with the highest "sym-priority" configured will become "active." This means that this ServerIron will actively answer for this virtual server. Any others will wait until they may have the highest sym-priority (e.g., should the active ServerIron fail).

A “failure” should be better defined here as well. An obvious failure would be if the ServerIron loses power or network connectivity. But a failure will also occur if all of the real servers bound to the virtual server fail their health checks. If the active ServerIron (for a given virtual server) stops answering for that virtual server, another sym-priority election takes place. The ServerIron with the same virtual server with the highest sym-priority assumes the active role. All others assume the “standby” role.

Active/Active

You may want to review Chapter 3 for this one. Active/Active can be a difficult concept to grasp. This is most easily demonstrated with two ServerIrons, though you could use many. One method creates alternating Active/Standby pairs. Essentially, we create two VIPs. Both ServerIrons are configured with both VIPs.



We would configure ServerIron A with a higher sym-priority for, say, VIP1, and a lower sym-priority for VIP2. Alternately, we would configure ServerIron B with a higher sym-priority for VIP2 and a lower sym-priority for VIP1.

The final element is that we would need our clients to use both IP addresses. As mentioned in Chapter 3, this is commonly done by using DNS. DNS has a natural round-robin effect when having one name resolve to multiple IP addresses. Both VIPs will receive traffic and each VIP has a backup in case of failure.

If you're using traditional SLB (e.g., your real servers are directly connected to the ServerIrons), there is one other method that can be used. It is called “true Active/Active.” In this method, you configure only one VIP on both ServerIrons, and each one has a different sym-priority. The difference is that you enable this command in the virtual server config on each ServerIron:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual www
SLB-ServerIron(config-vs-www)#sym-active
```

In this scenario, ServerIron A still actively answers for incoming traffic (as it has the higher sym-priority), but it hands off some traffic to ServerIron B. ServerIron A communicates much more closely with ServerIron B to make sure that each ServerIron has approximately the same CPU load. ServerIron A offloads accordingly to keep things balanced.

Remote Servers

As Source NAT and DSR have shown us, a real server does not have to be directly attached to the ServerIron to be load balanced. Now, let's see this logic in its extreme. Remote servers are load-balanced servers that are typically not in the same network as the ServerIron. Often, these are not even in the same data center. Sometimes, they're not even in the same country. Remote servers are often used for disaster recovery. In other words, they are a server of last resort. By default, they are treated this way when bound to the virtual server.

Remote servers are configured very similarly to real servers. In fact, the only difference is the initial setup:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server source-ip 1.2.3.100
255.255.255.0 0.0.0.0
SLB-ServerIron(config)#server remote-name remotel 123.1.2.3
SLB-ServerIron(config-rs-remotel)#source-nat
SLB-ServerIron(config-rs-remotel)#port http
```

Instead of using “server real-name” to define the server, I used “server remote-name.” This defines the server as a remote server. Notice that I also configured “source-nat.” This is the only load balancing that could work to a server this remote. Also, notice that I have configured a publicly-routable “source-ip” for the ServerIron. I've specified its default gateway as 0.0.0.0. This indicates using the configured default gateway for the management IP. This saves you from having to have a publicly-routable address as the ServerIron's management IP address. Instead, it uses the source-ip when performing health checks and passing traffic to the remote server.

When this remote server is bound to the virtual server (with the rest of the real servers), it will be health checked, but it will not receive traffic. It only receives traffic after all the bound real servers have failed. Again, this is a server of last resort.

Primary and Backup Servers

There may be times where you don't really need to configure a remote server. Maybe you just want one or two of your real servers to be considered “last resort.” Or maybe you want some remote servers to be active participants with the real servers (instead of “last resort”). For this, we use Primary and Backup server configurations.

Primary and Backup are triggered at the virtual server port level. Once triggered, all servers (real or remote) are considered primary, except for those that are specifically marked as “backup.”

To mark a server as “backup”:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real rs1
SLB-ServerIron(config-rs-rs1)#backup
```

This command does nothing until the Primary/Backup configuration is triggered at the virtual server port level. To trigger this configuration:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual www
SLB-ServerIron(config-vs-www)#port http lb-pri-servers
```

If there are no “backup” servers defined, all the bound real servers are used. “Backup” servers can always be added later, if desired.

Clones

Nope. I'm not talking about sheep. I'm talking about real servers. Let's say you need to configure some real servers, a lot of real servers. For example, let's say you're configuring real servers for a farm of 200 servers. You can already feel your wrists cramping, can't you?

Well, Brocade's stepped in to help with a cloning function. You can clone an existing real server. The clone copies all of the base “port” commands (not Layer 7 checks, etc.), and it automatically binds the newly-created real server into the same virtual server (and in the same way). To clone:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real rs1
SLB-ServerIron(config-rs-rs1)#clone-server rs2 10.1.1.12
```

Summary

- The Brocade ServerIron provides switching on Layers 2-7
- The minimum requirements for a ServerIron chassis are one Application Switch Management Module, one Management Module, one Switch Fabric Module, and one Interface Module.
- Barrel Processors (BP) are dedicated to load balancing functions
- The ServerIron offers unique protection against TCP SYN attacks with SYN-Defense and SYN-Guard
- Predictors are the method the load balancer uses to balance the load across real servers
 - Least Connections
 - Round-Robin
 - Weighted

-
- Server response time only
 - Least connection and server response time weight (formerly “Response Time/Least Connections”)
 - Least local connections (chassis only)
 - Least local sessions (chassis only)
 - Real servers are the servers to be load balanced
 - Real servers may be configured to listen on many different ports
 - Virtual servers are the VIPs; they define a virtual address that the ServerIron listens on to load balance traffic
 - Virtual servers may be configured to listen on many different ports
 - Real server ports are bound to virtual servers
 - Layer 3 health checks consist of ARP and ping (ICMP Echo)
 - Layer 4 health checks are enabled automatically when the real server port is bound to a virtual server
 - Layer 7 health checks are disabled by default
 - When Layer 7 health checks are enabled, they occur automatically when the real server port is bound to a virtual server
 - The key commands for troubleshooting SLB are:
 - show server virtual
 - show server real
 - show server bind
 - A “source-ip” enables a ServerIron to perform health checks and pass traffic to servers that are not directly connected; this is a requirement for Source NAT, DSR, and remote servers
 - Source NAT may be configured globally or per real server
 - DSR requires that a loopback be configured on the real server with the same IP address as the VIP to which it is bound
 - DSR is unique to the virtual server port
 - A single real server port may be bound to many virtual servers through shadow ports
 - Hot-Standby Redundancy provides ServerIron active/standby redundancy for two ServerIrons
 - Symmetric SLB provides virtual server redundancy across multiple ServerIrons
 - Symmetric SLB provides both active/standby and active/active redundancy

- Remote servers are servers that are more than a single router hop away
- Remote servers are used as a last resort, unless Primary and Backup servers are enabled
- Primary and Backup servers are used to define servers of last resort

Chapter Review Questions

1. I've defined "port http" on a real server, and I have already bound it to one virtual server. Can I bind HTTP on this real server to another virtual server?
 - a. No. You must first unbind it from the first virtual server
 - b. Yes, just "bind http rs http" on the second virtual server
 - c. No. You must create another real server
 - d. Yes, but you must create an additional shadow port on the real server and use that
2. Which predictor simply passes traffic to the next server, regardless of load or weight?
 - a. Round-robin
 - b. Least-connections
 - c. Weighted
 - d. Server response time only
3. In Hot-Standby Redundancy, what will happen if the sync link goes down?
 - a. Hot-Standby Redundancy will be disabled
 - b. The ServerIrons will continue hello messages across the data link
 - c. Both ServerIrons will assume an active role
 - d. None of the above
4. I've enabled Primary/Backup with the virtual server command "port http lb-pri-servers." How do I define a real server as "Primary"?
 - a. The command "primary" in the real server config
 - b. Don't enter the command "backup" in the real server config
 - c. The command "primary" in the remote server config
 - d. The command "port http primary" in the real server config

5. Which command enables Symmetric SLB in the virtual server config?
 - a. sym-priority
 - b. sym-slb
 - c. server backup
 - d. sym-standby
6. By default, how many retries does a Layer 3 health check perform before declaring a failed health check?
 - a. 2
 - b. 3
 - c. 4
 - d. 1
7. I've configured a loopback address on my web servers. How would I enable DSR?
 - a. In the real server config, "port http dsr"
 - b. In the virtual server config, "port http dsr"
 - c. In the real server config, "enable dsr"
 - d. In the virtual server config, "dsr"
8. For well-known TCP/UDP protocols, what keyword would I use to enable Layer 7 health checks on my real server?
 - a. port url
 - b. l7
 - c. keepalive
 - d. url
9. In Source NAT, what address will my real server see as the source IP address of the incoming traffic?
 - a. The ServerIron's management IP address
 - b. The ServerIron's source-ip address
 - c. The VIP address
 - d. The client's IP address

10. I'm using DSR. When I perform a "show server real", I don't see any "Rx-pkts" or "Rx-octets", but I see lots of "Tx-pkts" and "Tx-octets." Why?
 - a. You forgot to configure a loopback on your real server
 - b. You didn't enable DSR in the real server config
 - c. Your real server port is not active
 - d. In DSR, the ServerIron never sees return traffic; these counters are useless

Answers to Review Questions

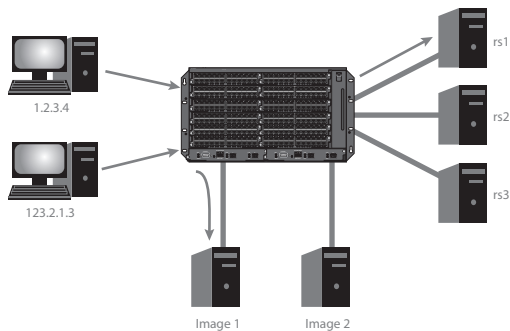
1. d. This is where you would use a shadow port. You are binding to a different port on the real server, but you are binding the same port (HTTP, in this case) on the virtual server. Just make sure you've got "no port http translate" in your virtual server config.
2. a. Round-robin just picks the next one in line. If a server is down, it skips it, but other than that, it blindly sends to the next server.
3. b. The ServerIrons continue their hello messages across the data link until the sync link is restored. If the data link between the ServerIrons goes down as well, answer C would be correct.
4. b. All servers (including remote) are considered primary, unless they are specifically configured as "backup."
5. a. "sym-priority" is the magic word. This is added to the virtual server config on all participating ServerIrons.
6. c. It is 4.
7. b. DSR is unique to the virtual server port, so it is enabled in the virtual server config. The correct commands in this case are "port http dsr."
8. c. Tricky one. Most people think of the "port http url" command available for HTTP. But for all well-known Layer 7 protocols, you can use the keyword "keepalive" (e.g., "port smtp keepalive") to trigger their unique Layer 7 health check.
9. b. The ServerIron's source-ip. This is the whole reason Source NAT works. If you missed this question, review Chapter 3. When the ServerIron receives the packet, it substitutes the source-ip address (that is most "local" to the real server) as the source IP of the packet (as well as substituting the IP of the real server as the destination IP).
10. d. So many engineers (and their bosses) have required an explanation of this. DSR is unique in that it's the only solution where the ServerIron does not see all the traffic. Given that, certain predictors and statistics will not behave necessarily as expected.

Session Persistence and Transparent Cache Switching

We've covered SLB and various redundancy methods in the previous chapter. Let's explore load balancing options in a bit more detail.

Policy-based Server Load Balancing (PBSLB)

Policy-based Server Load Balancing permits you to load balance to certain groups of real servers depending on the source IP address of the client.



This is often looked at as the ability to load balance based on source IP address, destination IP address and port because the switching decision is tied to the virtual server port.

Policy-based SLB highlights:

- Policy-based SLB is enabled for individual ports on virtual servers
- Because policy-based SLB is enabled on a per-VIP basis, some VIPs configured on the ServerIron ADX can have policy-based SLB enabled, while others do not
- Policy-based SLB can exist on a standalone device or in high-availability configurations, such as active-standby, symmetric, and active-active configurations
- Policy-based SLB can coexist with other ServerIron ADX features, including FWLB, NAT, and TCS
- Policy-based SLB cannot coexist on the same VIP with Layer 7 switching features, including URL switching and cookie switching

The first thing to do is to create groups by assigning each involved real server port to a relevant group. This is done in the real server config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real rs1
SLB-ServerIron(config-rs-rs1)#port http group-id 1 1
```

Here, we've assigned real server "rs1's" HTTP port to Group 1. There are two "1s" because a group-id is always configured in a range. This should be read as "assigning group-id from 1 to 1." If we want to assign this real server port to groups 1 through 5, we would use "port http group-id 1 5."

In our example above, let's say that we want HTTP for real server rs1 to be a member of Group1. We want HTTP for real servers rs1, rs2, and rs3 to be members of Group 2, and we want our HTTP image servers, image1 and image2, to be members of Group 3. Configure it like this:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real rs1
SLB-ServerIron(config-rs-rs1)#port http group-id 1 2
SLB-ServerIron(config-rs-rs1)#server real rs2
SLB-ServerIron(config-rs-rs2)#port http group-id 2 2
SLB-ServerIron(config-rs-rs2)#server real rs3
SLB-ServerIron(config-rs-rs3)#port http group-id 2 2
SLB-ServerIron(config-rs-rs3)#server real image1
SLB-ServerIron(config-rs-image1)#port http group-id 3 3
SLB-ServerIron(config-rs-image1)#server real image2
SLB-ServerIron(config-rs-image2)#port http group-id 3 3
```

Now, let's say that we want the specific host 1.2.3.4 to always be directed to Group 1 (rs1) when it connects to our virtual server. But, we want any hosts coming from the 123.2.1.0/24 network to be directed to the image servers in Group 3 (image1 and image2). Any other source IP should be directed to Group 2 (rs1, rs2, and rs3). Now define the policy:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server pbslb add 1.2.3.4 1
SLB-ServerIron(config)#server pbslb add 123.2.1.0/24 3
SLB-ServerIron(config)#server pbslb default-group-id 2
```

The policy states that all traffic from coming from 1.2.3.4 will be directed to real server Group 1. All traffic coming from the network 123.2.1.0/24 will be directed to real server Group 3, and any other traffic (the defined “default-group-id”) will be directed to real server Group 2. That's what we want. Now, how do we enable it?

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual-name www 1.1.1.3
SLB-ServerIron(config-vs-www)#port http
SLB-ServerIron(config-vs-www)#port http sw-l4-pbslb
SLB-ServerIron(config-vs-www)#bind http rs1 http
SLB-ServerIron(config-vs-www)#bind http rs2 http
SLB-ServerIron(config-vs-www)#bind http rs3 http
SLB-ServerIron(config-vs-www)#bind http image1 http
SLB-ServerIron(config-vs-www)#bind http image2 http
```

Notice that we've bound all five real servers to the virtual servers. You would think this means that load balancing would take place on all five servers. If it weren't for the “sw-l4-pbslb,” that would be the case. However, because of that line, the servers will be load balanced according to our defined policy.

Session Persistence

The Worldwide Web has become more and more a part of everyone's daily life. As a result, businesses and other organizations understand how important it is to make sure their Web sites are always up. One of the more conventional ways to do this is to use a farm of servers to support the Web site. If one server experiences difficulty, the site is still available to the user. Serverlrons are a major part of this type of infrastructure.

But what if we have a specific web application or function? What if a client's transaction starts on one server, but then gets load balanced to a different server? The new server doesn't have any information about the transaction its predecessor just started. The problem I'm describing is referred to as Session Persistence, and it continues to be one of the foremost struggles of web designers and infrastructure engineers. We need the redundancy of multiple servers, but we also need certain functions to be performed on a persistent machine (once initiated).

The Brocade ServerIron provides several methods to combat this problem. We'll cover a few of them in this chapter.

Concurrent

Some applications require a client to create multiple sessions. One of the most popular examples of this is passive FTP (File Transfer Protocol). For this to function, the client makes an initial connection to the server using TCP 21. The server then listens on a random TCP port above 1023, and instructs the client (over the existing TCP 21 session) to make another connection to the server on this given port. There are many Java-based applications that function very similarly. These are sometimes referred to as *parallel connections*.

In standard load balancing, an incoming TCP session is handed off to one real server (decided by the predictor). A second incoming TCP session (as described above) is handed off to another real server. This could be the same real server (depending on the predictor and circumstances), but odds are good that it will be a different real server. Let's say, we started the passive FTP session (TCP 21) to the virtual server and were handed off to rs-ftp1. The real server (rs-ftp1) tells the client to connect to it again using TCP 1048 (to which it is now listening). We initiate a connection to the virtual server again using TCP 1048. Now, the virtual server isn't listening on TCP 1048. It's only listening on TCP 21. And even if it were listening on TCP 1048, what if it handed us off to rs-ftp2? That real server doesn't know about our initiated connection with rs-ftp1, and it's certainly not listening on TCP 1048.

Using Concurrent connections, we're instructing the virtual server that once an initial session is made (and as long as it's still active), it should send all requests (regardless of the port) from this client IP address to the same real server. By default, concurrent connections are disabled.

Sticky

Sticky is very similar to Concurrent. The difference is that, instead of forwarding all requests (regardless of port) to the same real server, it just forwards to the same destination port. For example, when you request a Web page, you initiate at least one TCP session using HTTP (TCP 80). What many people don't realize is that it is not uncommon for a client (requesting one page) to actually initiate several TCP sessions. Perhaps the images are stored at a different site. Perhaps a dynamic HTML application is requested differently than the page that houses it. These successive requests all have the possibility of being assigned to a different real server in your virtual server pool.

Let's say, you've got a specialized web page application that sends the client through a series of web pages. It's important that this client stay with the same real server throughout the process. How can we make sure that happens? We make the virtual server's HTTP port "sticky." This means that as long as the requests are coming from the same source IP address (to the same destination port), the client will be directed to the same real server.

When a sticky session is initiated, a timer is created. When communication stops from the client, the timer starts. If it reaches its configurable expiration (defined by “sticky age”), the client will no longer stick to the same real server. If it hasn't quite reached its configurable expiration, and the client initiates more traffic, the timer will restart.

By default, sticky sessions are not enabled on virtual server ports, with the exception of Secure Sockets Layer (SSL). Due to the way SSL negotiates a session (involving several TCP sessions), it will not function in any way other than sticky. The ServerIron automatically adds the sticky line to the virtual server config when SSL is defined.

Configurable TCP/UDP Application Groups

This is also referred to as “port tracking.” This function is very similar to concurrent and sticky. Like concurrent, it accounts for applications that require the client to make several successive connections for the application to function. For this reason, we need to make sure that all of those connections reach the same real server.

In our concurrent example, we talked about passive FTP. It requires the client to make a second connection on a random port. For Application Groups, the client is required to make a second connection on a *fixed* or *known* port. In other words, we can anticipate on what port the second client connection will be made.

All Configurable TCP/UDP Application Groups have one port that is considered the “primary.” This is the port that the client initially connects to. You can define your Application Group with up to four other ports that the client may successively initiate connections to. Because Application Groups are defined, the ServerIron makes sure that the same source IP's successive connections are handed off to the same real server.

One of the things that makes Application Groups similar to sticky is that they use the same timer (the “sticky age” timer). When the client connection starts, the ServerIron watches communication coming from the client. If the client communication stops (even briefly), the timer starts. If it reaches the time defined by the “sticky age” setting, the ServerIron no longer sends successive communication from that client to the same real server. However, if the timer's expiration has not been reached, and the client sends additional communication, the timer is reset.

Cookies

Make mine chocolate chip, please. What? Oh! You're still here! Excuse me. How embarrassing!

What were we talking about? Ah, yes! Cookies! In the mid-1990's, the competing web browsers of the day brought us a new feature: HTTP cookies. These are simply chunks of text that are sent from the web server to the client. Once the client has received a cookie, it automatically includes the cookie in future communications to that same web site. This is often used to track an individ-

ual session. For example, a client goes to a website and logs in (with a username and password) to a special area of the site. The web server verifies the username and password authentication, assigns a cryptic session ID and sends this ID to the client in an HTTP cookie. When the client requests further web pages in this protected area of the site, the requests will contain, in the HTTP header, the cookie (with the session ID). The receiving web server will be able to verify that it's a current session that was properly authenticated.

This works if your client is connecting to the same individual web server each time, but what if your web servers are being load balanced? Well, you've got a couple of options:

Cookie Switching

With cookie switching, you can configure a value (an ID) to each individual participating real server in your ServerIron's configuration. Then you would configure your individual web servers to hand their clients a cookie containing the ID (assigned to the particular server). When a client makes an initial connection, the ServerIron load balances it according to the predictor. The server that it hands off to gives the client a cookie that contains the real server's ID. The next packets from the client contain this cookie. The ServerIron looks at the cookie of the incoming packet, and directs the connection to the real server belonging to that cookie.

To configure cookie switching, you need to configure IDs on the real servers:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real rs1
SLB-ServerIron(config-rs-rs1)#port http server-id 2001
SLB-ServerIron(config-rs-rs1)#server real rs2
SLB-ServerIron(config-rs-rs2)#port http server-id 2002
```

Notice that the ID is tied to the real server *port*. It is certainly possible to assign multiple IDs to the same real server (say, a unique ID for each port?). The “server-id” can be any number between 1024 and 2047. The important thing is to make the numbers unique to each real server. The use of higher numbers within this group is recommended. For instance, I chose to start at 2000 and correspond the ID numbers with the real server's name (e.g., rs1 - 2001, rs2 - 2002, etc.).

To enable cookie switching on your virtual server, you must first create a rule and a policy:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#csw-rule rule1 header "Cookie" search
"ServerID="
SLB-ServerIron(config)#csw-policy policy1
SLB-ServerIron(config-csw-policy1)#match rule1 persist offset
0 length 0 group-or-server-id
SLB-ServerIron(config-csw-policy1)#default forward 1
SLB-ServerIron(config-csw-policy1)#default rewrite insert-
cookie "ServerID" "*" "*" age 60
```

Now, you must enable the rule on the VIP:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual-name www 1.2.3.4
SLB-ServerIron(config-vs-www)#port http
SLB-ServerIron(config-vs-www)#port http csw-policy policy1
SLB-ServerIron(config-vs-www)#port http csw
SLB-ServerIron(config-vs-www)#bind http rs1 http rs2 http
```

You can see that we've defined the name of the cookie to which the ServerIron should pay attention. We've called it "ServerID," but you're welcome to call it whatever you'd like (just make sure you use the same name when you configure your servers). Finally, we enable cookie-switching. Notice that we've enabled it to the virtual server port. It is certainly possible to use cookie switching on one port in a virtual server, but not on another port.

Before enabling cookie switching, you need to configure the real servers. How you do this depends greatly on the platform and software you are using on your web server. Consult your documentation. The idea is that you'll want your real server to send the client a cookie (named "ServerID," in this example) with a value corresponding to the values we've defined in the ServerIron. For example, rs1 should be sending a cookie named ServerID that has a value of 2001. Likewise, rs2 should be sending a cookie named ServerID that has a value of 2002.

Now, sometimes you're dealing with a web application that is not easily tampered with. It may be a commercial package that your system is running. That still doesn't take away from the fact that you need it to generate a cookie for your clients. Or does it?

Thanks to Brocade, there is a little variation on cookie switching called *cookie insertion*. It works almost exactly the same way as cookie switching. The difference is that it's the ServerIron that generates the cookie. You would still assign IDs to each real server. This time, after the client makes its initial connection, the ServerIron intercepts the server's reply and adds a cookie (containing the real server's ID) to the reply packet's header. When the client continues its communication, it includes the cookie it received from the ServerIron. The ServerIron sees the cookie, and directs the traffic to the real server where it belongs. The real server sees the cookie as well, but it ignores it. If you have an application that will not ignore it, you can also configure the ServerIron to remove the cookie before it delivers an incoming packet to the real server.

This is accomplished by adding the following commands to the previous configuration:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual-name www 1.2.3.4
SLB-ServerIron(config-vs-www)# port rewrite cookie-insert
"ServerID"
SLB-ServerIron(config)#csw-policy policy1
default forward 1
default rewrite insert-cookie "ServerID" "*" "*" age 60
```

If you need to delete the cookie before the real server sees it, you must add this line to the csw-policy command as well:

```
SLB-ServerIron(config-vs-www)#match r1 rewrite delete-cookie
```

Deleting the cookie requires the ServerIron to recalculate the full checksum of the data packet. This could induce some overhead that could impact performance on your load balancer. You should use this command only when absolutely necessary. Again, most web applications will simply ignore a foreign cookie.

Cookie Hashing

With cookie hashing, you're not assigning IDs to your real servers. In fact, you change nothing on your ServerIron's configuration (other than enabling cookie hashing). To use cookie hashing, your web servers will need to assign a cookie to the client. The ServerIron performs a checksum-like operation on the entire cookie, creating a hash value. It stores this hash value in the *hash bucket*, associating the hash value with that particular real server. Whenever connections come in, the ServerIron compares the hash value of their cookie to the hash bucket. If there's a match, the ServerIron directs the traffic to the real server to which the hash belongs.

To enable cookie hashing, you need to add the following to your virtual server config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#csw-rule r1 header "cookie" search
"ServerId"
SLB-ServerIron(config)#csw-policy p1
SLB-ServerIron(config-csw-p1)#match r1 persist offset 0 length
0
SLB-ServerIron(config)#server virtual-name www 1.2.3.4
SLB-ServerIron(config-vs-www)#port http csw-policy p1
SLB-ServerIron(config-vs-www)#port http csw
```

Note that this configuration is tied to the virtual server port.

SSL ID Based Persistence

To establish an SSL session, the client and server must exchange certain parameters (encryption, decryption, etc.). As part of this negotiation process, the server sends the client an SSL ID. With SSL ID Based Persistence enabled on your ServerIron, the ServerIron makes a note of this ID and ties it to the real server that sent it. All future incoming communication for that session contains that SSL ID. The ServerIron now always sends traffic containing a specific SSL ID to the real server that created it.

To enable SSL ID Based Persistence, you need to add only one line in your virtual server's config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual-name www 1.2.3.4
SLB-ServerIron(config-vs-www)#port ssl
SLB-ServerIron(config-vs-www)#port ssl session-id-switch
```

URL Switching

URL switching provides the capability to load balance based on Layer 7 information. Specifically, we're referring to the user-requested Uniform Resource Locator (URL). The URL is the full path of the item being requested. For example, Brocade's main web site is <http://www.brocade.com/>. The web page discussing the latest ServerIron products is <http://www.brocade.com/products-solutions/products/application-delivery/serveriron-adx-series/>.

Notice that it's similar, but there's some additional information beyond the first single slash (/). Technically, the "www.brocade.com" is considered the *Server Request*, and the "/products-solutions/products/application-delivery/serveriron-adx-series/" is considered the *URL*.

In a nut shell, URL switching allows the ServerIron to load balance to some real servers if one type of URL is received, to other real servers if another type of URL is received, and to yet others if another type of URL is received. When the client requests a web page, the URL the client is requesting is in the HTTP header. According to policies that you define, the ServerIron will load balance to different servers based on the on the URL in the incoming HTTP header.

Let's say we have six real servers. Two are general web servers. Two are image servers. And two are CGI application servers. We'd like to bind all six of these servers to the same virtual server, but we only want static HTML content (located in the "/www" directory) to be served by the web servers, images served by the image servers, and CGI applications handled by the application servers.

There are a set of rules that define the conditions for URL switching. We start by creating URL rules and specify a matching method. The method "prefix" tells the policy to look at the beginning of the URL. The next line tells the policy what to look for. In this case, we told it to look for any URLs starting with "/www". The next rule is using the method "suffix". This is telling the rule to look at the very end of the URL. We know that our images are all GIF images, and they all have the filename suffix ".gif". The last rule method is called "pattern". This tells the policy to look for whatever we specify anywhere within the URL, in this case `cgi`.

```
SLB-ServerIron#conf t
SLB-ServerIron(config)# csw-rule r1 url prefix "/www"
SLB-ServerIron(config)# csw-rule r2 url suffix ".gif"
SLB-ServerIron(config)# csw-rule r3 url pattern "cgi"
SLB-ServerIron(config)#csw-policy p1
SLB-ServerIron(config-csw-p1)# match r1 forward 1
SLB-ServerIron(config-csw-p1)# match r2 forward 2
SLB-ServerIron(config-csw-p1)# match r3 forward 3
SLB-ServerIron(config-csw-p1)#default forward 1
```

It's important to be careful to place default policies or default groups in your CSW, but look out for loops.

Now, we need to define the real servers in their respective groups:

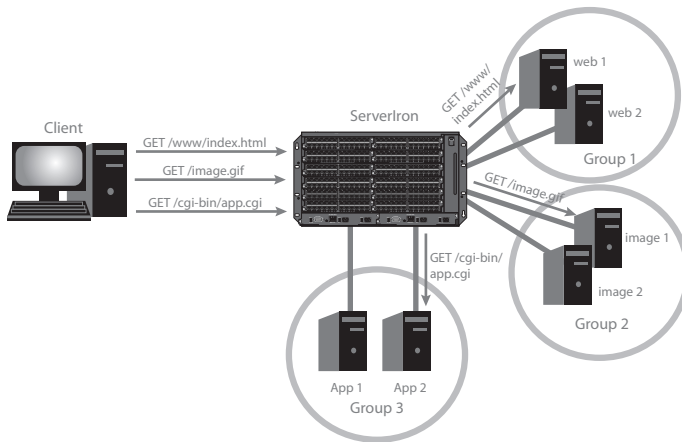
```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real web1
SLB-ServerIron(config-rs-web1)#port http group-id 1 1
SLB-ServerIron(config-rs-web1)#server real web2
SLB-ServerIron(config-rs-web2)#port http group-id 1 1
SLB-ServerIron(config-rs-web2)#server real image1
SLB-ServerIron(config-rs-image1)#port http group-id 2 2
SLB-ServerIron(config-rs-image1)#server real image2
SLB-ServerIron(config-rs-image2)#port http group-id 2 2
SLB-ServerIron(config-rs-image2)#server real appl
SLB-ServerIron(config-rs-appl)#port http group-id 3 3
SLB-ServerIron(config-rs-appl)#server real app2
SLB-ServerIron(config-rs-app2)#port http group-id 3 3
```

You can see that these are the same group commands that we introduced in Policy-based SLB. Finally, we need to apply the CSW policy:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server virtual-name www 1.2.3.4
SLB-ServerIron(config-vs-www)#port http
SLB-ServerIron(config-vs-www)#port http csw-policy p1
SLB-ServerIron(config-vs-www)#port http csw
SLB-ServerIron(config-vs-www)#bind http web1 http web2 http
SLB-ServerIron(config-vs-www)#bind http image1 http image2
http
SLB-ServerIron(config-vs-www)#bind http appl http app2 http
```

We've enabled URL switching with the command **port http url-switch**. Notice that this is tied to the virtual server port. Next, we specified which URL map to use. Notice that we only specified one. All three maps that we've defined are combined in a daisy chain:

- general — check for “/www” at the beginning of the URL
 - if it's there, load balance to server Group 1
 - if not, check against URL map “image”
- image — check for “gif” at the end of the URL
 - if it's there, load balance to server Group 2
 - if not, check against URL map “app”
- app — check for “/cgi-bin/” anywhere in the URL
 - if it's there, load balance to server Group 3
 - if not, load balance to server Group 1



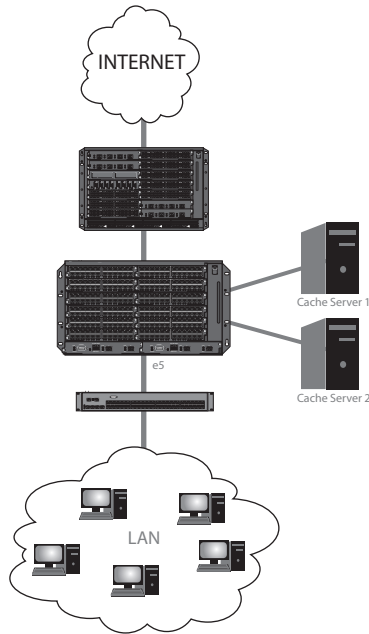
There is much more flexibility with URL switching than we can cover in this book. For more information, see Brocade's web site (<http://www.brocade.com/>).

What is Transparent Cache Switching?

Before we answer this question, let's look at its parts. A cache server is a server that processes web requests more efficiently than by directing every request to the Internet. For example, if you have 50 LAN users all requesting the same web page, you could forward all 50 requests out to the Internet. Or, you could forward one request to the Internet, store the resulting page on a local cache server, and redirect the other 49 requests to the local cache server.

The act of redirecting the other 49 requests is considered Cache Switching. You are making a decision as to whether you will forward a request to the Internet (as the item is not yet on the cache server), or redirect the request to the local cache server. This is often the secondary function of a proxy server in many corporate LANs. The proxy server serves as both a proxy server and a cache server. This requires clients to configure their workstations so that all web requests are funneled through the proxy server.

But what if you don't want to use a proxy server, and you don't want to change all of your user's workstation configurations? This is where the *transparent* piece of it comes in to play. In Transparent Cache Switching (TCS), the ServerIron intercepts the web request transparently and decides whether the request should be forwarded to the Internet, or to the cache server. This entire process is transparent to the client.



Configuring Transparent Cache Switching

There are three basic steps to configure TCS:

1. Enable TCS
2. Define the cache servers
3. Assign the defined cache servers to cache-groups

Enabling TCS

TCS is defined to be either global (entire ServerIron) or local (per interface). To configure TCS globally:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#ip policy 1 cache tcp 80 global
```

TCS is enabled by defining an IP policy. We've chosen ip policy "1," but this could be any number from 1 to 64. The keyword "cache" is what defines this as enabling TCS. The "tcp 80" defines what type of traffic we want to analyze for caching. In this case, it would be TCP port 80 (HTTP)-web requests. Finally, the keyword "global" indicates that we want to enable this function ServerIron-wide.

To configure TCS local to a specific interface, use the following commands:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#ip policy 1 cache tcp 80 local
SLB-ServerIron(config)#int e 3
SLB-ServerIron(config-if-3)#ip-policy 1
```

Here, we've applied TCS only to interface e 3.

Defining Cache Servers

Cache servers are defined in a very similar way to real servers and virtual servers:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server cache-name Cache1 10.2.3.4
```

The “server cache-name” defines this as a cache server. I've given it the name “Cache1.” Finally, we've listed its IP address.

Cache-Groups

A cache-group is a collection of cache servers. It is applied to the interfaces that are accepting web requests. In our example, this is the interface that connects the ServerIron to the LAN (e5). Up to four cache-groups may be defined on a single ServerIron. Each cache-group may contain up to 254 servers. To configure the cache-group:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server cache-group 1
SLB-ServerIron(config-tc-1)#cache Cache1
SLB-ServerIron(config-tc-1)#cache Cache2
```

This group (“1”) contains two cache servers (“Cache1” and “Cache2”). Finally, we need to apply this group to the interface sending web requests to the ServerIron:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#int e 5
SLB-ServerIron(config-if-5)#cache-group 1
```

This can be applied to as many interfaces as you need. In our example, there is only one interface: e 5.

By default, all interfaces are applied with cache-group 1 (making the example unnecessary). If you want to remove this application from any interface, you can either assign a different cache-group to the interface, or, in the interface config, enter **no cache-group 1**.

TCS is now configured. Verify the configuration and caching statistics using the command **show cache**.

Policy-based Caching

Normally, the ServerIron tries to cache any web site that is requested. Some web sites, however, should not be cached. Perhaps their site's information is too dynamic. Whatever the case, you can use Policy-based Caching to decide which web requests to always forward to the Internet.

This is a two step process. The first step is to define the filters. You want to define which destination IP addresses should not be retrieved from cache. Here is an example:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#ip filter 1 deny any 123.1.2.3
255.255.255.255 tcp eq 80
SLB-ServerIron(config)#ip filter 2 deny any 213.2.3.4
255.255.255.255 tcp eq 80
SLB-ServerIron(config)#ip filter 1024 permit any any
```

Don't let the "deny" fool you. These are not ACLs. They are filters. A "deny" here means that the ServerIron will never attempt to pass requests for these servers to the cache servers. It will always forward them to the Internet. In this example, we have set aside the web servers at "123.1.2.3" and "213.2.3.4." Web requests to these servers should always be forwarded to the Internet (never to the cache servers). Notice that we did not use a wildcard mask, but a 1s mask (like the subnet mask). Here, a mask of all 1s means "this IP address and only this IP address."

Finally, the last line (which is customarily assigned filter number 1024; the last filter) has a global "permit any any." This means that all other destinations beside these two servers should be forwarded to the cache servers whenever possible. This is important. Like ACLs, filters take on an implicit "deny all" at the end unless a "permit" is defined. If we leave that last line off, all web requests are forwarded to the Internet, and the cache servers would never be used. This last line doesn't have to be filter 1024 (it could have been "3," in our example), but it's a good practice. It leaves you free to add additional filters if necessary, without having to worry about that last line.

I said there were two steps. The second step is to apply these policies to the cache servers. This is done with the keyword "filter-match," and it is done in the cache servers' config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server cache-name Cache1 10.2.3.4
SLB-ServerIron(config-rs-Cache1)#filter-match
```

Now the policy is applied. The ServerIron knows not to forward web requests for 123.1.2.3 and 213.2.3.4 to the cache servers (but to forward all other destinations).

Summary

- Policy-based SLB allows the ServerIron to load balance incoming requests to specific real servers based on the source IP address of the request
- Concurrent persistence allows a client to make multiple successive connections to the same real server on multiple unknown destination ports
- Sticky persistence provides the ability to load balance a client to the same real server with which it started the session
- Configurable TCP/UDP Application Groups provide the ability for a client to make multiple successive connections (up to 4) to the same real server on multiple predefined destination ports
- Cookie switching allows the ServerIron to load balance a client to the same real server based on the cookie the client received from the real server
- Cookie insertion allows the ServerIron to offer the client the cookie needed (instead of the web server) for cookie switching
- Cookie hashing provides the ability for the ServerIron to load balance based on custom cookies generated by the real servers
- SSL ID Based persistence load balances a client to the same SSL real server to which the client made its initial connection
- URL switching provides the ability to load balance to specific real servers (or groups of real servers) based on the contents of the requesting URL
- Transparent Cache Switching provides the ability to use the speed and efficiency of cache servers without having to configure the clients
- TCS is “transparent” to the client; it never has to know that its web request may have been forwarded to a cache server, rather than the Internet
- TCS allows for up to four cache-groups with up to 254 cache servers in each group
- All interfaces are applied to cache-group 1 by default
- Policy-based Caching allows you to define which destination web sites should never be retrieved from a cache server

Chapter Review Questions

1. All interfaces on the ServerIron are, by default, a member of cache-group:
 - a. 2
 - b. 5
 - c. 1
 - d. 3
2. URL switching requires defined policies to match the contents of the URL. This type of policy is called a:
 - a. ip filter
 - b. CSW policy
 - c. ip policy url
 - d. port http url
3. Which persistence method provides the ability for a client to connect to the same real server on multiple unknown destination ports?
 - a. Sticky
 - b. SSL ID Based
 - c. Concurrent
 - d. Application Groups
4. On a Brocade ServerIron, how many cache-groups may be defined?
 - a. 4
 - b. 3
 - c. 8
 - d. 0
5. Which keyword is used to apply defined policies to the cache servers in Policy-based Caching?
 - a. filter-match
 - b. ip filter
 - c. cache-group 1
 - d. keepalive

6. Which persistence method provides the ability to load balance a client based on matching the cookie it received from the real server with a value defined in the ServerIron config?
 - a. cookie switching
 - b. cookie hashing
 - c. cookie insertion
 - d. cookie caching
7. What keyword is used to enable TCS?
 - a. ip filter
 - b. ip policy
 - c. tcs enable
 - d. port tcs-switching
8. Which persistence method provides the ability to switch based on the client's cookie, without having to configure cookies on the real servers themselves?
 - a. cookie insertion
 - b. cookie switching
 - c. cookie caching
 - d. Application Groups
9. Which method permits the ServerIron to load balance, using predefined policies, based on the source IP address?
 - a. Policy-based Caching
 - b. SSL ID Based
 - c. Application Groups
 - d. Policy-based SLB
10. Which persistence method provides load balancing based on the client's cookie, without configuring any IDs in the ServerIron's real server configs?
 - a. cookie switching
 - b. cookie insertion
 - c. cookie hashing
 - d. cookie caching

Answers to Review Questions

1. c. All interfaces are, by default, applied to cache-group 1.
2. b. url-maps are used to define URL switching policies.
3. c. For applications such as passive FTP, where additional sessions are required on unknown ports, Concurrent persistence is the answer.
4. a. Only 4 cache-groups may be defined on a single ServerIron.
5. a. **filter-match** applies the defined policies in the cache servers' config.
6. a. Cookie switching. The key here is that the cookie was received from the real server. This rules out cookie insertion, where the cookie is received from the ServerIron itself. It is matched to a value defined in the ServerIron config. This rules out cookie hashing, which matches to a value in the hash bucket. And cookie caching doesn't exist.
7. b. **ip policy** is the command that is used to enable TCS, either globally or locally.
8. a. Cookie insertion. Cookie switching requires the real servers to be configured to send cookies. Notice I didn't include cookie hashing in this list, as it, too, could fit this description.
9. d. Policy-based SLB. The important thing to remember is that it provides source IP-based load balancing due to predefined policies. Session persistence methods based on the source IP (sticky, Application Groups, etc.) have no predefined IP-based policy.
10. c. Cookie hashing. Cookie switching and cookie insertion both require IDs to be configured in the ServerIron's real server configs.

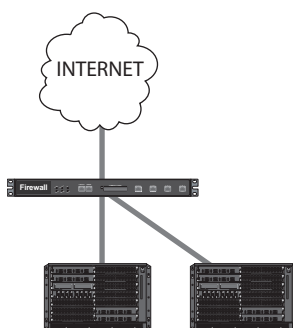
Firewall Load Balancing (FWLB)

We've explored many of the SLB and SLB-related functions of the ServerIron. We've seen how it can provide caching services transparently. Now let's see what ServerIrons can do for firewalls.

Why Would I Want to Load Balance Firewalls?

Firewalls are a necessary protection method, especially for networks connected to the Internet. The world is filled with people with malicious intent, or who simply have too much time on their hands. To protect your network, you'll want to funnel all of the traffic that's coming in from the Internet (and going out to the Internet) through a firewall, that can analyze packets and discard those that are undesirable.

What's the problem? Notice that I said "a" firewall. What does that tell you? As necessary as a firewall is, if all traffic to and from the Internet passes through a single firewall, the firewall becomes a single point of failure.



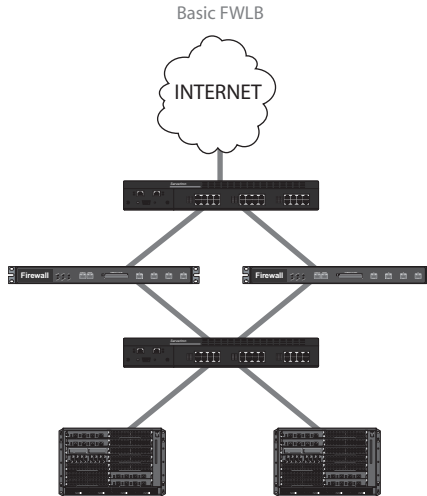
If the firewall fails, all traffic to and from the Internet would cease.

Many firewall manufacturers provide active/standby (and even active/active) solutions for their products to help eliminate the single point of failure problem. The problem is that these solutions usually use just two firewalls. What if your needs exceed the capability of those two firewalls? You've got to buy two

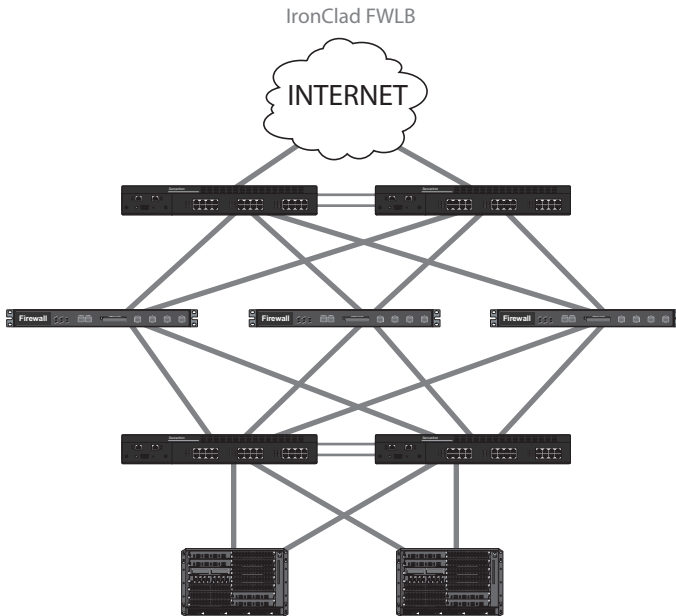
bigger firewalls and replace the originals. Firewalls are expensive appliances. Besides it's theoretically possible that your needs may become so great that not even two of the most expensive firewalls will be enough. You need the ability to expand your network to meet with the growing needs of your infrastructure.

Now, you could buy two, three, or even four firewalls, but how do you direct traffic to use any of them? Could we use them all actively? Or would we have to use just one and wait for it to fail? This is where Firewall Load Balancing (FWLB) comes in. The ServerIron takes on the responsibility to route traffic into and out of multiple firewalls. This allows for a near-infinitely expanding firewall infrastructure. The ServerIron stackable and chassis models can load balance up to 16 firewalls.

Brocade offers two solutions for FWLB: *basic* and *IronClad*. The basic solution provides the ability to expand the firewall infrastructure, utilizing many firewalls, but it does not provide full redundancy. In this case, ServerIrons themselves become single points of failure.



The solution? IronClad FWLB. This provides a scalable firewall infrastructure as well as redundancy.



Configuring FWLB

In either the basic or the IronClad solution, you need at least one ServerIron *outside* the firewall (to load balance incoming traffic), and at least one ServerIron *inside* the firewall (to load balance outgoing traffic). When configuring IronClad, you configure a pair of ServerIrons outside the firewall, and a pair of ServerIrons on the inside. The pairs must be either both chassis models or both stackables. The ServerIron FWLB solution currently supports a maximum of 16 firewalls.

IronClad can be configured as either active/standby or active/active.

To configure FWLB, you go through six steps:

1. Define the Firewall Servers
2. Define the Firewall Group
3. Assign the Firewall Servers to the Firewall Group
4. Configure the Firewall Paths
5. Enable FWLB
6. Configure Static MAC entries (optional, dependent on configuration)

You need to perform these steps on both the Outside ServerIron(s), and the Inside ServerIron(s).

Defining the Firewall Servers

By “firewall servers,” we mean the actual firewalls themselves. These are defined in a very similar way to real servers and cache servers:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server fw-name FW1 123.1.2.1
```

Like the real servers and the cache servers, we define each firewall to be load balanced.

Defining the Firewall Group

Similar to TCS, we need to define a firewall group. In TCS, the default cache group is 1. In FWLB, the only firewall group is 2. It seems kind of silly to have to define the only group available, but keep in mind that not all ServerIrons are configured to use FWLB. Also, it turns out that FWLB and TCS share resources. You can't have four cache groups and a firewall group on the same ServerIron. You can have four groups total (three cache groups and a firewall group). Also, a cache group cannot have the same number as a firewall group, so it would never be possible to have a cache group of 2.

To define the firewall group:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server fw-group 2
```

Assigning the Firewall Servers to a Firewall Group

Do you see a relationship yet between TCS and FWLB? Here, we make our defined firewall servers members of our firewall group. To assign the firewall servers to the group:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server fw-group 2
SLB-ServerIron(config-tc-2)#fw FW1
SLB-ServerIron(config-tc-2)#fw FW2
```

We've added to firewalls to our firewall group: FW1 and FW2. This configuration will load balance traffic across these two firewalls equally.

Configure Firewall Paths

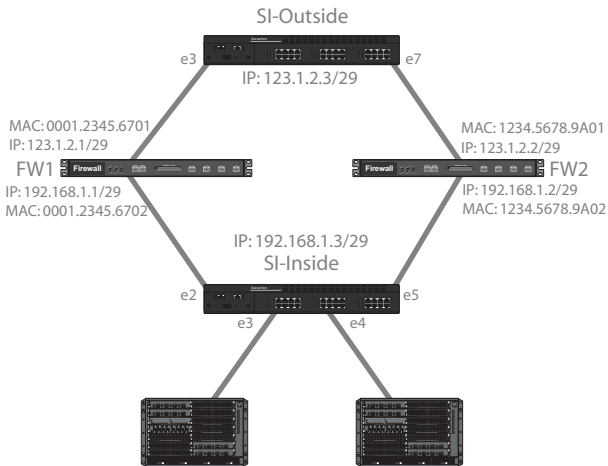
When we configure paths, we are telling one ServerIron how to get to the other ServerIron, through the firewalls. You can define up to 32 paths on ServerIron models.

NOTE: ServerIron software release 12.2.1 or later provides support for up to 64 paths on ServerIron ADX switches.

The paths are necessary so that the ServerIron can be sure that a given session (source IP and destination IP hash) is processed by the same firewall. When an incoming connection is received, the ServerIron creates a hash value (consisting of the source and destination IP address), and matches it to the

hash table. If a match exists, it will send the connection to the corresponding firewall. If it does not exist, the ServerIron selects the receiving firewall, and updates the hash table with the new entry.

Let's take a look at a more detailed example:



We've defined FW1 and FW2. We've defined them as members of fw-group 2. Now, we need to define paths. On SI-Outside, we need to define two paths (one through FW1 and one through FW2) to get to SI-Inside. Likewise, on SI-Inside, we need to define two paths (one through FW1 and one through FW2) to get to SI-Outside. Let's configure:

```
SLB-SI-Outside#conf t
SLB-SI-Outside(config)#server fw-group 2
SLB-SI-Outside(config-tc-2)#fwall-info 1 3 192.168.1.3
123.1.2.1
SLB-SI-Outside(config-tc-2)#fwall-info 2 7 192.168.1.3
123.1.2.2
```

The syntax of the “fwall-info” command is:

```
fwall-info <path number> <port number> <other ServerIron IP>
<next hop IP>
```

The first number is the path number. You will start with one, and move consecutively through (no gaps) the numbers until reaching 32. The next number is the outgoing interface number. In the first path, the interface facing FW1 is e 3. The interface facing FW2 is e 7 (as noted in the second path). The first IP address is the IP address of the opposite ServerIron. Since we're on SI-Outside, we want to reach the IP address of SI-Inside. Finally, we tell it the next hop IP address to get there. In this example, the first path directs SI-Outside through interface e 1 to 123.1.2.1 (FW1). The second path directs SI-Outside through interface e 2 to 123.1.2.2 (FW2).

Now, let's configure SI-Inside's paths:

```
SLB-SI-Inside#conf t
SLB-SI-Inside(config)#server fw-group 2
SLB-SI-Inside(config-tc-2)#fwall-info 1 2 123.1.2.3
192.168.1.1
SLB-SI-Inside(config-tc-2)#fwall-info 2 5 123.1.2.3
192.168.1.2
```

Configure Static MAC Entries For Each Firewall Server

In addition to paths, we need to configure static MAC entries on the Serverlrns for the interfaces on the firewalls. Here's what we would configure on SI-Outside:

```
SLB-SI-Outside#conf t
SLB-SI-Outside(config)#static-mac-address 0001.2345.6701 e 3
high-priority router-type
SLB-SI-Outside(config)#static-mac-address 1234.5678.9a01 e 7
high-priority router-type
```

Notice that we've tied the MAC address to the interface that is facing it. The first entry is for the outside interface of FW1. It is facing e 3 of SI-Outside. The second entry is for the outside interface of FW2. It is facing e 7 of SI-Outside. We've also added two more keywords. The keyword "high-priority" refers to QoS traffic. Setting this to "high-priority" means that we will use this static MAC entry for all traffic, regardless of the QoS priority ("high-priority" indicates everything from the highest priority queue to the lowest). The "router-type" parameter indicates that we are handing off to a Layer 3 device. The meaning of these two keywords is not as important as remembering that they must always be included. They will always be "high-priority router-type."

FWLB Predictors

Just like SLB, FWLB uses predictors to decide which firewall to load balance new connections. Here, the variety is not nearly as large as SLB. You have two choices:

- **total-least-conn (Total Least Connections).** This is the default predictor. Basically, it looks at which firewall is managing the least total connections, and it hands off the incoming connection to it.
- **per-service-least-conn (Per Service Least Connections).** This is similar to total-least-conn, but with this one, it looks for the total number of connections on a specific application. Say, for instance, that a web request was coming in. The Serverlrn knows that FW1 is handling 80 HTTP connections, 15 DNS connections, and 5 SMTP connections (100 total). The Serverlrn also knows that FW2 is handling 70 HTTP connections, 30 DNS connections, and 20 SMTP connections (120 total). Even though FW2 has more total connections, it has less HTTP connections (70 on FW2 versus 80 on FW1). The incoming HTTP request will be sent to FW2.

As we mentioned above, the “total-least-conn” predictor is enabled by default. If you would like to use the “per-service-least-conn” predictor, enter the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server fw-group 2
SLB-ServerIron(config-tc-2)#fw-predictor per-service-least-conn
```

FWLB show Commands

There are two key FWLB show commands that are helpful in making certain that your FWLB configuration is functioning the way you expect it to. The first is:

```
SLB-ServerIron#show fw-group
Firewall-group 2 has 2 members Admin-status = Enabled
Hash_info: Dest_mask = 255.255.255.255 Src_mask = 255.255.255.255
Firewall Server Name          Admin-st Hash-distribution
FW1                           6         0
FW2                           6         0
Traffic From<->to Firewall Servers
=====
Name: FW1                    IP: 123.1.2.1          State: 6   Groups = 2
Host->Fw-Server              Fw-Server->Host
      State                      Packets      Octets
Packets      Octets
Fw-Server    active              3877         1045140
186262  196008627
Total
186262  196008627
Name: FW2                    IP: 123.1.2.2          State: 6   Groups = 2
Host->Fw-Server              Fw-Server->Host
      State                      Packets      Octets
Packets      Octets
Fw-Server    active              1497         168408
0            0
Total
0            0
```

The State of the firewall server is determined by a Layer 3 (ping) health check that the ServerIron performs. Here, the same rules apply as the real server in SLB. The Administrative State can be one of the following:

- 0 - disabled
- 1 - enabled
- 2 - failed
- 3 - testing
- 4 - suspect
- 6 - active

The other command will help you to verify your configured paths:

```
SLB-ServerIron#show server fw-path
Firewall Server Path Info
Number of Fwall      =      2

```

Target-ip	Next-hop-ip	Port	Path	Status	Tx	Rx	State
192.168.1.3	123.1.2.1	3	1	1	1	1	0
192.168.1.3	123.1.2.2	7	2	1	1	1	0

The Status field will show a “0” if the link is down, and a “1” if the link is up. Likewise, the transmit side (Tx) or the receive side (Rx) can be either “0” (down) or “1” (up). The State applies to IronClad FWLB. A State of “3” means that the ServerIron at the other end of the path is in standby mode. A State of “5” means that the ServerIron at the other end of the path is in active mode. If IronClad FWLB is not configured, the State will always be “0.”

Fine-Tuning FWLB

There are a few additional commands that you should be aware of. These are some additional options to help fine-tune FWLB.

First, you can control the rate at which the ServerIron hands off connections to the firewall. If you have a weaker firewall, this will be something that you'll want to implement to prevent the firewall from being overwhelmed. The command is unique to the firewall server, and is as follows:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server fw FW1
SLB-ServerIron(config-rs-FW1)#max-tcp-conn-rate 1000
```

The “1000” indicates the maximum number of TCP connections per second the ServerIron will send to FW1. This value can be any number from 1 to 65,535. There is also a **max-udp-conn-rate** command that works the same way for UDP traffic.

Second, we move on to the Layer 3 health check. By default, the ServerIron checks the health of the firewall by sending a ping every 400 milliseconds (5 times every two seconds). If the ServerIron receives one or more responses within 1.2 seconds, the ServerIron deems the path healthy. If it does not get a

response, it will try again three more times (on chassis models; 8 more times on stackables) before declaring the path down. The retry number is configurable with the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server fw-group 2
SLB-ServerIron(config-tc-2)#fw-health-check icmp 15
```

The ServerIron will now retry its health check 15 times before declaring a path down. This number can be between 3 and 31.

Third, if a ServerIron receives incoming traffic, but the firewall it was going to load balance to has already reach its maximum connections, it will adjust its hashing mechanism and select an alternative firewall. You may also instruct the ServerIron to simply drop incoming packets when a firewall has reached its maximum connections. This is done with the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server fw-group 2
SLB-ServerIron(config-tc-2)#fw-exceed-max-drop
```

Stateful FWLB

A ServerIron performs *stateful* FWLB by creating and using session entries for source and destination traffic flows and associating each flow with a specific firewall. When a ServerIron receives a packet that needs to go through a firewall, the ServerIron checks to see whether it has an existing session entry for the packet in the following manner:

- If the ServerIron does not have a session entry with the packet's source and destination addresses, the ServerIron creates one. To create the session entry, the ServerIron selects the firewall that has the fewest open sessions with the ServerIron and associates the source and destination addresses of the packet with that firewall. The ServerIron also sends the session information to the other ServerIron in the high-availability pair, so that the other ServerIron does not need to create a new session for the same traffic flow.
- If the ServerIron already has a session entry for the packet, the ServerIron forwards the traffic to the firewall in the session entry. All packets with the same source and destination addresses are forwarded to the same firewall. Since the ServerIrons in a high-availability pair exchange session information, the same firewall is used regardless of which ServerIron receives the traffic to be forwarded.

For stateful FWLB, you associate certain types of traffic with a particular firewall. This is currently only supported with TCP or UDP traffic. The following example associates HTTP and FTP traffic with FW1, and SMTP and POP3 traffic with FW2:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server fw FW1
SLB-ServerIron(config-rs-FW1)#port http
SLB-ServerIron(config-rs-FW1)#port ftp
SLB-ServerIron(config-rs-FW1)#server fw FW2
SLB-ServerIron(config-rs-FW2)#port smtp
SLB-ServerIron(config-rs-FW2)#port pop3
```

Summary

- Firewall Load Balancing (FWLB) provides firewall redundancy and the ability to expand firewall capacity, by incorporating multiple firewalls working in parallel
- FWLB functions by having at least one ServerIron on the outside of the firewall load balancing incoming traffic, and at least one ServerIron on the inside of the firewall load balancing outgoing traffic
- Basic FWLB requires only one ServerIron on the outside of the firewalls and one ServerIron on the inside of the firewalls; this provides expandability, but not redundancy (the ServerIrons become a single point of failure)
- IronClad FWLB requires two or more ServerIrons on the outside of the firewalls and two or more ServerIrons on the inside of the firewalls; this provides expandability and redundancy
- Firewalls need to be defined as firewall servers in the ServerIron config
- Firewall servers are made members of a firewall group
- On a single ServerIron, there is only one group: Group 2
- For FWLB to function, you must configure paths and static MAC entries, so that the outside ServerIrons know how to reach the inside ServerIrons through the firewalls (and vice versa)
- FWLB uses a hash (comprising of the source IP and destination IP of the connection) to ensure that the same session is serviced by the same firewall
- For new connections, FWLB uses one of two predictors ("total-least-conn" by default) to decide which firewall will receive the request
- Stateful FWLB can be used with certain types of application traffic through specific firewalls

Chapter Review Questions

1. What is the minimum number of ServerIrons required for FWLB?
 - a. 1
 - b. 2
 - c. 3
 - d. 4
2. If FWLB and TCS are being used, how many cache groups may be created?
 - a. 1
 - b. 2
 - c. 3
 - d. 4
3. What is the group ID for the firewall group?
 - a. 1
 - b. 2
 - c. 3
 - d. 4
4. Using chassis ServerIrons, how many firewalls can be load balanced?
 - a. 8
 - b. 10
 - c. 12
 - d. 16
5. What is the minimum number of ServerIrons required for IronClad FWLB?
 - a. 1
 - b. 2
 - c. 3
 - d. 4
6. In FWLB, what does the configured firewall “path” describe?
 - a. the path to the Internet
 - b. the path to the firewall
 - c. the path to the LAN
 - d. the path from one ServerIron to the opposite ServerIron

7. What is the maximum number of paths that can be supported?
 - a. 2
 - b. 36
 - c. 32
 - d. 16
8. Assuming Stateful FWLB is enabled on all my FWLB ServerIrons (in both directions), what command would I use to direct all SSL traffic to FW1?
 - a. SLB-ServerIron(config-rs-FW1)#port ssl
 - b. SLB-ServerIron(config-rs)#ip policy 1 ssl
 - c. SLB-ServerIron(config-rs-FW1)#filter-match ssl
 - d. SLB-ServerIron(config)#port ssl
9. What command would I use to define a firewall in the ServerIron's config?
 - a. server real FW1 1.2.3.4
 - b. server fw-name FW1 1.2.3.4
 - c. server fwall-info FW1 1.2.3.4
 - d. server firewall FW1 1.2.3.4
10. What command is used to prevent the firewall from being overwhelmed?
 - a. limit connections
 - b. max-tcp-conn-rate
 - c. max-conn
 - d. max-rate

Answers to Review Questions

1. b. The simplest configuration for FWLB (Basic) requires one ServerIron on the outside of the firewalls, and one ServerIron on the inside of the firewalls.
2. c. TCS and FWLB dip from the same resource pool of four groups. If FWLB is enabled, that's one less that TCS can use.
3. b. Firewall Group 2 is the only group configurable on any ServerIron.
4. d. 16 firewalls can be load balanced with chassis ServerIrons.
5. d. For IronClad, you need at least two ServerIrons on the outside of the firewalls, and at least two ServerIrons on the inside of the firewalls.
6. d. The firewall path describes the path that the ServerIron must take to reach the ServerIron on the opposite side of the firewalls.
7. c. 32 paths are currently supported.

8. a. Tricky one. **port ssl** is the right command, but D is not correct, because it's being issued in the Global config, not the firewall server config.
9. b. **server fw-name <name> <ip-address>** is the correct syntax.
10. b. If you have a weaker firewall, this will be something that you'll want to implement to prevent the firewall from being overwhelmed.

Global Server Load Balancing (GSLB)

We've certainly seen many of the features provided by the Brocade ServerIron. We've studied the intricacies of SLB, and have been introduced to TCS and FWLB. Now it's time to take a look at one more acronym.

What is Global Server Load Balancing?

Global Server Load Balancing (GSLB) allows the ServerIron to serve as a DNS proxy for your sites. It makes a particularly intelligent proxy. It uses response time, health status, geographic location, and other criteria so that it may send the requesting client the IP address of the server that will serve them best.

Among the GSLB benefits are:

- No connection delay; the client must make a DNS query anyway; GSLB simply manipulates the IP address to which the name resolves
- Geographic awareness; this is based on the IP address of the client sending the DNS request
- Distributed site performance awareness; GSLB is aware of all of your sites and how well they are performing
- Fair site selection
- Statistical site performance measurements in a way that minimizes the impact of traffic spikes
- Best performing sites are favored, but not inundated (e.g., they receive more traffic, but not all traffic)
- All IP protocols (utilizing a name) are supported

Domain Name Service (DNS)

What's this DNS I keep mentioning? Well, I'll forewarn you that plenty of books have been written on this topic alone. We'll skim the high points here, but I'll give you some good places to go for more information at the end.

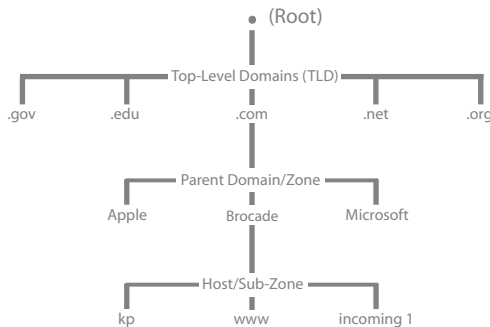
Domain Name Service (DNS) is a service that provides the ability to resolve a name to an IP address, and vice versa. On the Internet, you would submit a name (e.g., `www.brocade.com`) and, through DNS, you would get a response (e.g., `63.236.63.244`). Now your packets and communications have a Layer 3 address that they can use to communicate across the Internet.

The concept of resolving names to Layer 3 addresses is actually as old as the Internet itself (perhaps older). Name resolution started by keeping entries in a file. In a way, this file served much the same purpose as a phone book (though, not nearly as large). It was simply a text file that listed a name (e.g., `www.brocade.com`) and the IP address it belongs to (e.g., `63.236.63.244`). This system lasted for about 20 years, and is still in use today. Most operating systems provide a “hosts” or “hosts.txt” file that you can edit and provide your own local resolutions.

In 1983, a man by the name of Paul Mockapetris came up with the concept of DNS, and wrote the first implementation. It is now referred to in the IETF RFC's as RFC 1034 and RFC 1035 (written in 1987). In 1984 and 1985, a UNIX-based software program called BIND (Berkeley Internet Name Domain) was created to service DNS. This has remained one of the most popular programs for serving DNS.

DNS Hierarchy

DNS functions based on a hierarchy. The “hosts” file system didn't work too well, because it required every individual workstation to know how to resolve every address on the Internet. That would have been fine in the 1970's, but now, the file would be huge (and would have to be constantly updated). DNS doesn't function on the concept of centrally storing all of the names that need to be resolved. Instead, it uses a hierarchy to distribute the resolution to a more specific structure.



Not too many people realize that a Fully-Qualified Domain Name (FQDN) should properly be written with a period at the end of the name (e.g., `www.brocade.com.`). That period represents the Root of the DNS hierarchy. A name consists of a series of groups that are separated by a period. For example, let's

take the name “www.brocade.com.” This address is referring to a host named “www,” who is part of the “brocade” domain, which is part of the “com” Top-Level Domain (TLD), which is part of the Root domain (“.”).

The DNS Root domain currently consists of 13 servers. Ten of these servers are located in the US. One is in Stockholm, Sweden. One is in London, UK. And one is in Tokyo, Japan. Every server on the Internet that can make recursive DNS queries knows the IP addresses of these 13 servers.

Let's talk a little more about that word “recursive.” Every individual workstation that wishes to communicate with other servers on the Internet has at least one DNS server configured. This is the IP address of a server that this workstation will ask to resolve names to IP addresses (and vice versa). This server does not know all of the names or all of the IP addresses. It simply knows who to ask if it does not know.

So a client wants to know what the IP address is for “www.brocade.com.” It sends this request to its configured DNS server (usually, a server that is hosted by its local ISP). Let's say, that server does not know what the IP address is for “www.brocade.com.” It will make a recursive DNS query. Let's walk through what happens next:



- The local DNS server sends a query for “www.brocade.com” to one of the Root servers. It already knows the IP addresses of the Root servers, because all recursive DNS servers do. They're stored in a text file on the server.
- The Root server doesn't know the address (nor should it), but it does know the IP address of the servers that host the “com” TLD. It replies to the local DNS server, and tells it to check with the “com” TLD servers (giving the addresses of the servers).
- The local DNS server again sends a query for “www.brocade.com,” but this time, it sends it to the one of the addresses (for the “com” TLD servers) that it received from the Root server.
- The “com” TLD server replies to the local DNS server saying that it doesn't know how to resolve the name, either, but it does know the IP addresses of the servers that manage DNS for “brocade.com.” It gives the IP addresses of these servers.

- The local DNS server, once again, sends a query for “www.brocade.com.” It sends the query to one of the IP addresses of the servers that manage DNS for “brocade.com” (the addresses it got from its last reply).
- The servers that manage DNS for “brocade.com” do know how to resolve “www.brocade.com,” and they reply with “63.236.63.244” (the requested IP resolution).

Now, that seems like a lot of work (and it is), but it also decentralizes the data. The only servers that know how to resolve “www.brocade.com” are the DNS servers that manage that domain. Now wouldn't it be more efficient for the local DNS server to just start by querying the “brocade.com” DNS servers? Sure, but how would it know to start there? Every DNS server on the Internet would have to have the IP address for every domain name (and sub-domains) out there. Again, this would be a *monstrous* file and an administrative nightmare to make sure that it was up to date on all DNS servers. The Root servers don't change that often. They stay pretty constant. TLD servers change more frequently, but still, not that often. Parent domain and sub-domain DNS servers change all the time. This is fine, because they only have to update their information in one place: the TLD servers. This is usually done by the registrar (the company that registered the domain name).

When the local DNS server receives this resolution, it will keep it in its local cache. That way, if someone else makes a query asking for “www.brocade.com,” it will simply hand the client the answer, rather than initiate another recursive query. But how long will the local DNS server keep that name resolution? What if it changes? That's up to the administrator of DNS. When the DNS record is defined, there is a Time-To-Live (TTL) value assigned to it. This tells any other DNS server how long this resolution is good for. If the TTL expires, the “other” DNS servers will know that they have to perform the recursive query again, should anyone request the name.

DNS Domains/DNS Zones

DNS Domains and DNS Zones are often used interchangeably. It's hard to blame people who do this. It's often difficult to define the difference. We've started the DNS hierarchy with the “.” (root). The next level (to the left) is the TLD (“com,” “net,” “org,” etc.). The next division in a name (to the left) is usually what people refer to as the “domain.” In the example “www.brocade.com,” the right-hand “.” is the root, “com” is the TLD, and “brocade” is the domain (often, this would be read as “brocade.com”).

Now, there are those that consider a “zone” to be wherever administration changes. This would make the Root domain a zone. The TLD domain would also be a zone. In our example, “brocade.com” could also be considered a zone. How is it different from a domain? What if you saw this name: “www.internal.research.brocade.com.” Wait a minute. So, “brocade.com” is the domain, but what's “research,” “internal,” and “www?” They are zones. They still belong to brocade.com, but they have their own tables of names and resolution. Some even have their own DNS servers that host the “sub-domain.”

A zone describes the name servers that serve the zone, the mail servers that server the zone, and the individual host resolutions. This is collected in what is referred to as the *DNS zone file* or *zone database*.

The A Record

The A record is the line in the zone file that matches an individual name to an IP address. The “A” is for “address.” In our example for “www.brocade.com,” the DNS servers that are authoritative for “brocade.com” would have a zone file for “brocade.com.” In this zone file would be an A record that might look something like this:

```
www      IN      A      63.236.63.244
```

The “IN” is an outdated keyword standing for “Internet” (as opposed to local name resolution). The “A” specifies that this is an A record. The “www” is the individual host name. Now, how do we know that this is resolving “www.brocade.com” and not some other “www?” Because this line is in the “brocade.com” zone file. Finally, the IP address that it should resolve to is also listed.

Additional Information About DNS

For more information about DNS, you might look to:

- RFC 1034 and RFC 1035; these describe DNS structure and implementation; they can be read online at <http://www.ietf.org/>
- The Internet Assigned Numbers Authority (IANA) has several pages dealing with DNS and TLDs; they can be reached at <http://www.iana.org/>
- ISC (<http://www.isc.org/>) — they are the keepers of the BIND DNS server software; they also have many documents and information relating to DNS
- DNS and BIND by Cricket Liu and Paul Albitz (Published by O'Reilly); to date, this is considered the authority on DNS

Enter: Global Server Load Balancing (GSLB)

GSLB acts as a DNS proxy. We mentioned that before. But what good does that do me? Let's say that you have three geographically-diverse sites. All three sites serve the same web content. You want clients to use all three sites. “No problem,” you say, “I'll just use DNS round-robin.”

DNS round-robin is when you use the same A record to resolve several IP addresses. Say, for example, that www.something.com resolved to 1.2.3.4, 2.3.4.5, and 3.4.5.6. Now, someone doing a lookup using the **host** or **nslookup** command is going to see all three IP addresses returned. A client trying to connect to the site will simply take the first IP address it sees. Most DNS servers, by default, will rotate which IP address is first. For example, for the first person to request “www.something.com,” they may see “1.2.3.4, 2.3.4.5, 3.4.5.6.” The next person to request “www.something.com” might see “2.3.4.5, 3.4.5.6, 1.2.3.4.” You see where this is going? The next person to

request “www.something.com” might see “3.4.5.6, 1.2.3.4, 2.3.4.5.” Each time the client will pick the first one on the list. It's the DNS server that's using round-robin to decide which IP address is seen first.

Well, that seems reasonable, doesn't it? Problem solved. I'll just put each of my three sites' IP addresses in the A record, and let DNS serve them up. What if a site goes down? DNS won't make a change. It will keep handing off requests. Theoretically, if one of your three sites became unavailable, you would lose approximately 33% of your incoming traffic. About one-third of your clients would get a connection failure when trying to reach your site. Now, what if one of your sites was temporarily experiencing problems and was performing much slower than usual? Again, about 33% of your incoming clients would see this congestion, and experience it as well.

What if you had an agent who handled DNS much more intelligently? What if you had something that checked the health of the site, checked the responsiveness of the site, and maybe even checked the geographic location of the client in relation to the site being requested? And what if this something used that information to decide to which IP address to resolve the name to? Enter GSLB.

That's what it does. It adds an intelligent layer to DNS. It makes decisions on the fly as to which might be the best IP address to resolve to, given the circumstances. If a site is down, it will no longer resolve the name to that site's IP address (until it comes back up). If a site is not responding well, it will favor resolving to other sites, until that site is responding better. If a client in London is making a request, it will resolve the name to the IP address of the site in London, rather than the site in South Korea. It's as if we've got an operator working 24x7 watching every DNS request and doing his or her best to change the reply to suit the needs of the client. That's GSLB.

How Does GSLB Decide?

GSLB uses several metrics to decide which site would be best for the client. They will be presented to you in their default order. This can be changed. In default order, they are as follows:

Check 1: Server Health

If the remote site is dead, GSLB doesn't want to serve that IP address. To avoid this scenario, it performs a health check. It will perform a Layer 4 health check and, if configured, a Layer 7 health check. If the site fails the health check, it will not be considered for the IP resolution.

So, to what IP is this health check being performed? It's to whatever IP address you configured. It could be a virtual server on the remote site's ServerIron. It could be the actual IP address of a remote host. It could be a VIP contrived by some third-party appliance. It's the site that the client would see, if you resolved it to that site's IP address.

Now, I mentioned that this order of checking could be changed. Brocade (and I) recommend that if you do change it, keep the Server Health check first. You don't want the possibility of being directed to a site that is down, merely because the site's health wasn't the first thing checked.

Check 2: Session Capacity Threshold

Here, the GSLB ServerIron will query the ServerIrons at each of the remote sites to obtain the current number of TCP and UDP sessions, and the maximum number of sessions the ServerIrons are capable of running. Basically, it's checking how much the site can handle, and how close it is to reaching that maximum.

The Session Capacity Threshold is the percentage "busy" the site can be before it is considered "too busy." By default, this threshold is 90%. If a site's current number of sessions is less than 90% of its maximum sessions, it will still be considered for IP resolution. If it is higher than 90%, it will be considered "too busy" (or too close to being too busy).

Check 3: Round-Trip Time (RTT)

With this check, the GSLB ServerIron will track the speed of the TCP three-way handshake. It works likes this:

- The GSLB ServerIron receives a DNS request from a client, say, on network 1.2.3.0/24.
- The GSLB ServerIron checks a database of cached entries that contains information about each previous resolution. Let's say, it doesn't have any entries for the 1.2.3.0/24 network (or that it had entries, and they've aged out).
- The GSLB ServerIron will reply with a resolution (based on other criteria).
- The GSLB ServerIron will now query the ServerIron of the site to which it directed the client. It requests the Round-Trip Time (RTT) for the client it just sent there. The RTT is the time between the first TCP SYN it received from the client, and the TCP ACK (finishing the handshake) that it received from the client.

With multiple entries in the GSLB ServerIron's database, it will be able to compare and contrast RTT's and choose the fastest for the client that is requesting. In comparing two sites, the GSLB ServerIron will only choose one over the other if the difference is beyond the RTT Tolerance. By default, this value is 10%. In other words, if there's less than 10% difference in RTT between two sites, they are considered the same.

Also, this check is of little value if there's only one site that reports an RTT. There's no other site to compare it to. To collect initial data for a client range, this check is ignored for a small percentage of the requests from a given network. This is called the RTT Explore percentage, and by default, it is 5%. This means that, for a given client network, the first 5% of the requests will not consider RTT.

Check 4: Geographic Location of the Server

If all things are equal up to this point, the GSLB ServerIron will attempt to choose the site that suits the client geographically. To do this, the ServerIron must know where the client is, and where the site is.

For the client, it bases this on the IP address of the DNS server that made the request. The ServerIron can narrow down its geographic location by continent (North America, South America, Asia, or Europe).

For the remote site, it determines the geographic location by one of three methods:

- If the site address is a host or an appliance (a physical address, not a VIP), it will determine the region using that address
- If the site address is a VIP, the geographic region is based on the management IP address of the ServerIron hosting the VIP
- In the GSLB ServerIron, you can explicitly specify the region, presuming that the above criteria may not lead to the correct region; this setting supersedes the two previously-mentioned methods

Now that we know the geographic location of the client and the geographic location of the sites, the GSLB ServerIron will favor sites within the same geographic region.

Check 5: Available Session Capacity

If, after the previous four checks, you end up with two or more sites that are within the same region, the next thing the GSLB ServerIron checks is the Available Session Capacity. This is similar to the Session Capacity Threshold, in that the GSLB ServerIron looks at the remote site ServerIron's session information. Whereas, the Session Capacity Threshold checks to see if a site is too busy, the Available Session Capacity checks to see how busy (or how available) a site is.

This check compares how many unused session entries each site has. To prevent frequent, and unnecessary, changes in site preference, this check uses a tolerance percentage. By default, this percentage is 10%. For example, if ServerIron A has 1,000,000 sessions available, and ServerIron B has 800,000 sessions available, the GSLB ServerIron will prefer ServerIron A. The difference between the two is 200,000 sessions (1,000,000 - 800,000). Ten percent of 1,000,000 is 100,000, so this difference is outside of the tolerance percentage. However, if ServerIron A has 1,000,000 sessions available, and ServerIron B has 990,000 sessions available, the difference is only 10,000. This is less than the tolerance percentage (1,000,000 x .10 = 100,000). Now, ServerIron A and ServerIron B will be preferred equally. The tolerance percentage is configurable.

Check 6: Flashback

The year is 1983... Oh, sorry. Wrong flashback. If we still have multiple sites to choose from at this point, we'll base the next decision on the Flashback time. This is the response time of the site during the GSLB ServerIron's health check. It will favor the site with the fastest time.

This uses a tolerance percentage, too. By default, it's 10%. This means that the Flashback response times must be greater than a 10% variance. Otherwise, the GSLB will favor them equally.

For Flashback time, the GSLB ServerIron will first compare the speeds of the Layer 7 health check response (if it's configured). If there's not a big enough variance there, it will use the Layer 4 health check as a tie breaker.

Check 7a: Least Response Selection

Let's say we've passed through all six of these checks, and we still have more than one site to choose from! Well, the GSLB ServerIron has to make a decision. It needs a "tie-breaker." So it decides to send it to the site that it has chosen the least. For example, let's say you have three sites. The GSLB ServerIron has resolved requests to Site 1: 50% of the time, Site 2: 30% of the time, and Site 3: 20% of the time. If it comes down to Least Response, it will resolve to Site 3. Site 3 has been chosen the least often for DNS resolutions at this point.

Check 7b: Round-Robin Selection

Let's say, for whatever reason, you didn't want the last criteria to be Least Response Selection. Least Response is enabled by default. It can be overridden, if you enable Round-Robin. If this is enabled, it will take the place (at the end of the criteria) of Least Response.

Round-Robin works the same way it has worked with everything else we've described. It blindly takes the next site in line. If it resolved to Site 1 last time, it'll resolve to Site 2 this time, and so on. You would use Round-Robin if you didn't want the GSLB ServerIron to favor sites that have recently been enabled, or sites that had gone down, but have now been restored.

Affinity

Affinity allows you to set a GSLB preference to a particular site, given a certain range of clients. For example, let's say that you know clients from the 123.1.2.0/24 network would always be best served to go to Site 3. You would define a GSLB affinity for clients from 123.1.2.0/24 to always resolve to Site 3, unless it is down. In this sense, affinity acts kind of like Check 1.5. The Server Health check still supersedes it, but, if it's defined, it will take precedence over all others. You can define up to 50 affinities.

Administrative Preference

Administrative Preference is kind of like Check 6.5. It's considered right before Least Response. Administrative Preference is disabled by default. When it is enabled, all sites have a default administrative preference of 128. The preference for each site may be set at any number from 0 to 255. A higher number indicates a higher GSLB preference.

Why would you want to do this? Well, here are some good reasons:

- You can temporarily disable a site without changing any other configuration pieces. Let's say, your site has a back-end database. The database at one site is becoming particularly congested (a situation that would not reveal itself in the other checks). Or let's say you were performing a scheduled maintenance on a site. You could temporarily disable client access to the site simply by setting a low administrative preference (say, 0).
- Let's say you have a GSLB ServerIron that's also acting as the ServerIron for one of the sites. You may want this ServerIron to deliberately prefer itself. To do this, you could set the administrative preference to 255.

Configuring GSLB

To configure GSLB, you need to start by making sure the ServerIron is able to health check to remote sites. You can do this by configuring a source-ip:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server source-ip 123.1.2.3
255.255.255.0 123.1.2.1
```

Next, you define the ServerIron as a DNS proxy to your actual DNS servers. You'll recognize this setup from SLB:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#server real-name dns-server1
123.1.2.11
SLB-ServerIron(config-rs-dns-server1)#port dns proxy
SLB-ServerIron(config-rs-dns-server1)#server real-name dns-
server2 123.1.2.12
SLB-ServerIron(config-rs-dns-server2)#port dns proxy
SLB-ServerIron(config-rs-dns-server2)#exit
SLB-ServerIron(config)#server virtual-name DNSProxy
123.1.2.10
SLB-ServerIron(config-vs-DNSProxy)#port dns
SLB-ServerIron(config-vs-DNSProxy)#bind dns dns-server1 dns
dns-server2 dns
```

It's configured just like we're load balancing DNS servers with SLB, with one important difference. In the real server configs, we used the words "port dns proxy." This indicates that we want the ServerIron to act as a DNS proxy for the real servers. Now, the SLB rules for real servers apply here. If these DNS servers are not directly connected to the ServerIron, you'll need to use Source NAT (as the ServerIron is acting as a proxy, DSR would not work).

It's the address of the VIP (123.1.2.10) that you should use as your domain's DNS server. When a domain is registered with an Internet registrar, it requires at least one DNS server (often two) to list as the authority for this domain. For our domain, we want to make sure our registrar is pointing to 123.1.2.10 as this address.

Next, we need to define the remote sites involved in the GSLB. These are configured like this:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb site london
SLB-ServerIron(config-gslb-site-london)#si-name vip1
213.10.20.100
SLB-ServerIron(config-gslb-site-london)#si-name vip2
213.10.20.200
SLB-ServerIron(config-gslb-site-london)#exit
SLB-ServerIron(config)#gslb site seoul
SLB-ServerIron(config-gslb-site-seoul)#si-name vip1
220.200.20.100
SLB-ServerIron(config-gslb-site-seoul)#si-name vip2
220.200.20.200
SLB-ServerIron(config-gslb-site-seoul)#exit
SLB-ServerIron(config)#gslb site slc
SLB-ServerIron(config-gslb-site-slc)#si-name vip1 123.1.2.100
SLB-ServerIron(config-gslb-site-slc)#si-name vip2 123.1.2.200
```

We've configured three sites: "london," "seoul," and "slc." Each site happens to be configured with two VIPs. Let's say that each site is using active/active symmetric SLB. All of these IP addresses are virtual servers configured on their respective ServerIrons. This is also where you configure an administrative preference. The syntax would be, for example, "si-name vip1 213.10.20.100 255." This sets this specific VIP with an administrative preference of 255.

Next, we need to configure our zone file. This will detail what our domain name is, and what host name we are resolving for these sites.

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb dns zone-name brocade.com
SLB-ServerIron(config-gslb-dns-brocade.com)#host-info www
http
SLB-ServerIron(config-gslb-dns-brocade.com)#host-info ftp ftp
```

Here, we've defined two names: "www.brocade.com" and "ftp.brocade.com." After each definition, we've specified what service these VIPs will perform. In the case of "www," it's HTTP (TCP 80). In the case of "ftp," it's, coincidentally, FTP (TCP 21).

Finally, you'll want to make sure that the GSLB protocol is enabled on all of the ServerIrons involved. This is done in the global config:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb protocol
```


This allows the ServerIrons to share GSLB information. This information is shared using TCP 182, by default. You'll need to make sure that this communication is permitted through any relevant firewalls. Plus, you'll want to make sure that the source-ip of the GSLB ServerIron is allowed to health check the sites.

Fine-Tuning DNS Parameters

There are more parameters (than we have space for) to tamper with in GSLB. Here are some handy favorites.

First, if a site fails a health check, the IP address is still included in the DNS response. It's just not at the top of the list. If you would like the IP address removed all together (when a site fails health check), enter the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb policy
SLB-ServerIron(config-gslb-policy)#dns active-only
```

Next, GSLB will give a DNS reply containing all of the IP addresses of all of the sites defined. It just puts the IP address of the site that it has chosen at the top of the list. You can set GSLB so that it only sends one IP address (the address it has chosen; the “best” address) in the reply. This is done with the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb policy
SLB-ServerIron(config-gslb-policy)#dns best-only
```

Next, the GSLB ServerIron will periodically verify its host and zone information with the real DNS servers that are configured. By default, it makes this check every 30 seconds. You can change the interval with this command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb policy
SLB-ServerIron(config-gslb-policy)#dns check-interval 60
```

This command has changed the interval to 60 seconds (every minute). This can be any number from 0 to 1,000,000,000 seconds.

Next, remember our discussion of TTL in DNS? Each A record can have its own TTL. By default, GSLB supersedes any defined TTL with its own. It does this so it can maintain best performance, as the sites change. By default, GSLB will change the TTL of an A record to 10 seconds. You can change the TTL with this command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb policy
SLB-ServerIron(config-gslb-policy)#dns ttl 30
```

This changes the TTL value to 30 seconds. This value can be anything from 0 to 1,000,000,000 seconds. You can turn off this feature and let the DNS servers themselves manage TTLs by using “no dns ttl” in the gslb policy.

Next, there may be times when you need to override the information in the DNS servers. This is done with the “DNS override” feature. You define a list of IP addresses to which a given host name should resolve. GSLB will then ignore resolution information for that host and make its decision from the list you’ve defined:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb dns zone-name brocade.com
SLB-ServerIron(config-gslb-dns-brocade.com)#host-info www
http
SLB-ServerIron(config-gslb-dns-brocade.com)#host-info www ip-
list 123.123.1.11
SLB-ServerIron(config-gslb-dns-brocade.com)#host-info www ip-
list 123.123.1.12
SLB-ServerIron(config-gslb-dns-brocade.com)#host-info www ip-
list 123.123.1.13
```

Here, we’ve defined the standard resolve line “host-info www http,” but we’ve also added an “ip-list” of three additional IP addresses. These addresses will not be used unless DNS override is enabled. Enable it by entering this command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb policy
SLB-ServerIron(config-gslb-policy)#dns override
```

At this point, the resolving for the name “www” will only include the IP addresses we’ve defined in the list. The GSLB will use the same appropriate criteria to decide which IP is listed first.

And finally, by default, the ServerIron will forward requests to the DNS server (using SLB, as we defined previously). It is possible to have the ServerIron cache the responses it gets from the DNS server. This will provide an even faster response (as the request doesn’t have to be forwarded). To enable this feature, use the following command:

```
SLB-ServerIron#conf t
SLB-ServerIron(config)#gslb policy
SLB-ServerIron(config-gslb-policy)#dns cache-proxy
```

GSLB show Commands

Here’s a list of handy show commands to help you troubleshoot your GSLB configuration:

- **show gslb default** — Displays information about the various GSLB policies; specifically, the default configuration of these policies
- **show gslb policy** — Displays the current information about the various GSLB policies; this should be the same as “show gslb default,” unless changes have been made

- `rshow <ip> server real` — Displays the “show server real” information from a remote ServerIron (using the GSLB protocol); the “<ip>” would be the IP address of the remote ServerIron
- `show gslb site` — Displays information about the sites that have been defined in GSLB, including session information, administrative preference, and geographic location
- `show gslb dns zone` — Displays the zone entries that are defined in the GSLB configuration
- `show gslb dns detail` — Displays even more information than “show gslb dns zone,” including selection percentages, and Flashback statistics

Summary

- GSLB acts as an intelligent DNS proxy for multiple sites
- DNS resolves names to IP addresses
- A DNS “A” record resolves one name to, usually, one IP address
- A DNS “A” record may resolve a single name to many IP addresses; by default, most DNS servers will change the order of the multiple IP addresses returned in a round-robin fashion; most clients will accept the first address as the address to use
- GSLB uses several methods to decide which IP address to list first in its DNS reply:
 - Server Health — if the site is down, it's not chosen
 - Affinity — if there's a defined affinity to a particular site from a certain source, that site will be chosen
 - Session Capacity Threshold — if the site is too busy, it's not chosen
 - Round-Trip Time (RTT) — if the site responds to the client slower than another site, it is not chosen
 - Geographic Location of the Server — if the site is in a different continent than the client, it is not chosen
 - Available Session Capacity — which site is the least busy?
 - Flashback — which site responds the quickest to my health check?
 - Administrative Preference — if defined, this value determines which site will be chosen; the higher preference is chosen
 - Least Response Selection/Round-Robin Selection — if Round-Robin is defined, GSLB chooses the site based on Round-Robin; otherwise, by default, it chooses the site that has been chosen the least

Chapter Review Questions

1. What does the Flashback selection check?
 - a. the response speed from the remote site to the client
 - b. the response time to the GSLB ServerIron's health check
 - c. the busiest site
 - d. the least busy site
2. What GSLB policy allows manual host entries in the ServerIron config to supersede the information on the DNS servers?
 - a. dns override
 - b. dns proxy
 - c. dns supersede
 - d. override dns
3. By default, Administrative Preference makes the GSLB decision:
 - a. always, superseding all else
 - b. if there's still a tie after Server Health, but before Session Capacity Threshold
 - c. if there's still a tie after Flashback, but before Least Response/Round-Robin
 - d. if there's still a tie before Server Health
4. The very top of the DNS hierarchy is referred to as:
 - a. Root Domain
 - b. Top-Level Domain
 - c. Parent Domain
 - d. Sub-Domain
5. The Round-Trip Time is:
 - a. the Layer 4 health check from the GSLB ServerIron to the remote site
 - b. the TCP three-way handshake time between the client network and the remote site
 - c. the Layer 3 health check from the remote site to the GSLB ServerIron
 - d. the Layer 7 health check from the GSLB ServerIron to the remote site

6. Names like “com,” “net,” “edu,” and “gov” are all examples of:
 - a. Root Domains
 - b. Parent Domains
 - c. Parent Zones
 - d. Top-Level Domains
7. Affinity provides the ability to influence the GSLB decision based on:
 - a. the source IP address of the DNS request
 - b. the geographic location
 - c. a manually-configured weight
 - d. none of the above
8. The Geographic Location narrows the location of an IP address to the nearest:
 - a. city
 - b. county
 - c. country
 - d. continent
9. What is the default Session Capacity Threshold?
 - a. 90%
 - b. 80%
 - c. 50%
 - d. 10%
10. By default, affinity is considered:
 - a. immediately
 - b. following Server Health check
 - c. following Least Response
 - d. following Flashback

Answers to Review Questions

1. b. Flashback is the GSLB ServerIron's health check results. Specifically, it is the time that the site has taken to respond to the health check.
2. a. dns override.
3. c. Administrative Preference is only considered if there's still a tie at the Flashback check.
4. a. Root Domain (represented by a “.”).

5. b. The Round-Trip Time is the time difference between when the ServerIron of the remote site sees the first TCP SYN and the final TCP ACK of the initial three-way handshake.
6. d. These are Top-Level Domains (TLD).
7. a. Affinity makes its decision based on the source IP address of the host making the DNS request (usually a local DNS server to the actual client).
8. d. Continent. Specifically, either North America, South America, Europe, or Asia.
9. a. 90%.
10. b. By default, the Server Health is always considered first.

Glossary

10Base-T An Ethernet specification that is included in the IEEE 802.3 standard. It provides speeds of up to 10 Mbps over twisted-pair copper cables (minimum category 3 required).

100Base-T An Ethernet specification that is based on the IEEE 802.3u standard. It provides speeds of up to 100 Mbps over twisted-pair copper cables (minimum category 5 required).

1000BaseSX An Ethernet specification that is based on the IEEE 802.3z standard. It provides speeds of up to 1 Gbps (or 1000 Mbps) over multimode fiber-optic cables.

1000BaseTX An Ethernet specification that is based on the IEEE 802.3z standard. It provides speeds of up to 1 Gbps (or 1000 Mbps) over twisted-pair copper cables (minimum category 5e required).

AAA Authentication, Authorization, and Accounting. This is a system to provide secure access and accountability to a network device. See also: accounting and authorization.

ABR Area Border Router. This is an OSPF router that has one interface as a member of one area, and a different interface as a member of a different area. The router sits, logically, on the border between one area and another.

Accounting This is the third "A" in "AAA." Accounting provides a way to see when an authenticated user logged in, when they logged out, and what they did while they were logged in. Some examples include syslog and SNMP traps.

ACK Short for "Acknowledgment." In TCP, it is a special segment that is sent to acknowledge that data was received. It is also the last segment in the initial 3-way handshake of a TCP session, and the last segment of a properly-closed TCP session.

ACL Access Control Lists. These are a list of rules that define what type of traffic matches a certain criteria. They are often used to control traffic flow (e.g., permitting and denying certain types of traffic), but they are also used in Policy-Based Routing, NAT, and many other functions.

Administrative distance When a router has multiple entries of the same destination network in its route table, this method is used by the router to decide which entry to use. Administrative distance assigns a number from 0 to 255 to each routing method (e.g., 1 for static, 120 for RIP, etc.). The lower the number, the more favored the entry is.

Application Layer This is the seventh layer of the OSI Reference Model. It is the layer that receives data created by the user, and performs some function with it. Examples of Layer 7 TCP/IP protocols include HTTP, FTP, and SSH.

Area In OSPF, an area defines a collection of OSPF routers. The size of the area (e.g., how many routers) is determined by the network engineer.

ARP Address Resolution Protocol. This is a protocol that matches an IP address (Layer 3) with a MAC address (Layer 2).

ARPANET Advanced Research Projects Agency Network. This was a network, developed in 1969, that many feel is the predecessor to the modern Internet. It started with just four nodes: three in California, and one in Utah.

AS Autonomous System. In OSPF, an Autonomous System defines a collection of OSPF areas. A router that is considered to be on the border of the autonomous system will have one interface as a member of an OSPF area, and a different interface participating in a different routing protocol. In BGP, an autonomous system defines a collection of routers that are managed by the same administrative entity. Each AS is assigned a number between 0 and 65,535.

ASBR Autonomous System Boundary Router. This is an OSPF router which has at least one interface participating in an OSPF area, and at least one other interface participating in some other routing protocol (e.g., RIP, BGP, etc.).

ASCII American Standard Code for Information Interchange. This is a method of representing each character, number, or punctuation with a code number. For example, the decimal number 65, in ASCII, corresponds to the capital letter "A." The decimal number 66, in ASCII, corresponds to the capital letter "B," and so on.

ASIC Application-Specific Integrated Circuit. For Brocade Switches, an ASIC provides some processing power to the individual interface that is independent of the switch's CPU.

ATM Asynchronous Transfer Mode. This is a Layer 2 protocol that involves breaking up packets into fixed-length cells.

Attenuation This is the distance at which a signal will become so weak that it will be unrecognizable by the receiver. When discussing networking over copper wire, attenuation is the distance at which the electrical signal degrades too much to be properly recognized. When discussing networking over fiber optics, attenuation is the distance at which the laser signal degrades too much to be properly recognized. Attenuation defines the limitation of the length of the physical medium.

Authentication This is the first "A" in "AAA." It defines how a user is authenticated. Some examples include local user accounts, RADIUS, or TACACS.

Authorization This is the second "A" in "AAA." It defines what an authenticated user may do. Privilege levels are defined here. Some users may only be able to read information. Others may be able to change the configuration, and so on.

Autonegotiation In Ethernet, autonegotiation is the process in which a NIC automatically determines what speed and duplex setting would be most optimal for the device it's connecting to.

Backbone In OSPF, the backbone area is Area 0. Every OSPF design must have a backbone, and all areas in the design must have a connection to this area. Traffic that is traveling from one area to another must traverse the backbone.

Backplane The electrical path within a switch that allows an interface to send signals to another interface.

Base Layer-3 One of the three types of software that may be run on many Brocade switches. This code offers standard Layer 3 functionality, but not higher-end features like OSPF, BGP, NAT, etc.

BDR Backup Designated Router. In a broadcast, multi-access OSPF area, this router acts as the failover router for the Designated Router.

BGP Border Gateway Protocol. This is a routing protocol that is designed for large complex WANs. Most notably, it is the routing protocol that is used by routers participating in the Internet. It is a path vector routing protocol. Rather than basing its routing decisions on the next-hop router, it bases the decision on the neighboring AS. It is well-suited to accommodate routing tables containing hundreds of thousands of routes.

BigIron This is the middle-grade chassis Layer 2-3 switch of the Brocade line. It is well-suited as a core switch, and it is ideal for BGP.

Binary This is a number system that is based on the number two. Exactly two single digits are allowed within this system: 0 and 1. This is the lowest-level language involved in networking and computing.

Bit In binary, this is a single binary digit. It can be a 0 or a 1.

BP Barrel Processor. This is a processor specifically designed for Layer 7 switching. It resides within the WSM of a ServerIron chassis. A single WSM may have several BPs.

BPDU Bridge Protocol Data Unit. This is a special frame used by Spanning Tree Protocol to find Layer 2 loops, and to share information between switches.

Broadcast address This is a special address that is sent to all hosts in a given broadcast domain. In Layer 2 (Ethernet), the address is FF:FF:FF:FF:FF:FF. In Layer 3, the broadcast IP address is the last possible address of the subnet. The universal broadcast IP address is 255.255.255.255.

Broadcast domain This defines how far a broadcast will travel. At Layer 2, this can be organized physically, or by using VLANs. At Layer 3, this can be organized using subnets.

Byte A collection of eight bits. A byte may represent any decimal number from 0 to 255.

CAM Content Addressable Memory. This is special memory that is used by the ASIC.

Chassis This is a switch body design that is modular. It allows blades or modules to be added to the device to expand its capabilities.

CIDR Classless Inter-Domain Routing. This is a method of describing a subnet. It uses the network address followed by a "/" and the number of bits in the network portion of the address. For example, the network 192.168.4.0 with a subnet mask of 255.255.255.0 would be represented as 192.168.4.0/24.

CLI Command Line Interface. This is the text-based interface used to access a switch's commands. This can be seen by connecting to a switch's serial console or by remotely connecting to a switch using a command-line based protocol, like Telnet or SSH.

Confederation In BGP, a confederation is a collection of stub ASs.

Config Short for "configuration." This is the list of commands that change the switch's behavior from its default settings. There are two types of configs. The Startup config is loaded when the switch is booted. The Running config is actively used while the switch is operating.

Convergence This describes the process of all units having finished sharing all of the information they have to share. In Spanning Tree Protocol, convergence describes the process of all switches having tested all paths and finishing the root switch election. In Routing Protocols, convergence describes all routers having shared their routes with all other participating routers.

Cookie In HTTP, a cookie is a value that is assigned by the server to the client. The client sends the value back to the server for all subsequent communication. Many different pieces of information may be tracked in a cookie. It is very commonly used for session persistence.

CSMA/CD Carrier Sense Multiple Access/Collision Detection. This is a method used by the Ethernet protocol to send traffic. Carrier Sense is the ability to detect a link (e.g., is there a medium for me to use? Copper? Fiber? Air?). Multiple Access means that many different nodes can access that medium. Collision Detection means that it's aware if two different nodes try to use the medium at the same time.

Data Link Layer This is the second layer of the OSI Reference Model. This Layer is the final preparation point for the data to be translated to a physical signal. All data collected through the layers is formed into frames.

DCB Data Center Bridging. The DCB effort undertaken by the IEEE 802.1 work group adds new extensions to bridging and Ethernet, so that it becomes capable of converging LAN and storage traffic on a single link. Often, you hear that new DCB features will make Ethernet "like Fibre Channel." That's true, because the new features being added to Ethernet are solving issues that FC faced in the past and successfully resolved. IEEE is expected to complete its work on the components of DCB by the end of 2010. The new enhancements are PFC, ETS, and DCBX.

DCBX Data Center Bridge eXchange. The role of DCBX is to allow two adjacent nodes to exchange information about themselves and what they support. If both nodes support PFC and ETS, then a lossless link can be established to support the requirements of FCoE.

DDoS Distributed Denial of Service. This describes a type of attack, in which an attacker uses many different network nodes to overload a victim.

Default gateway This is a special entry in a device's routing table. If the device needs to reach an IP address for which it does not know the next-hop (e.g., the IP address is not in its routing table), it will forward the packet to the default gateway (if it is defined). It is a "gateway of last resort."

DHCP Dynamic Host Configuration Protocol. This is a protocol that is used to assign IP addresses, subnet masks, default gateways, and other network configuration items automatically to hosts within a broadcast domain.

Distance vector This is a routing protocol that chooses its best route by determining the least number of routers (hops) to get to the destination.

DNS Domain Name System. This is a protocol that is designed to match a host name to an IP address, and vice versa.

DR Designated Router. In a broadcast multi-access OSPF area, this is the router that will keep the master copy of the area's OSPF database. All other routers in the area will sync to this master. This router is decided on by an election. The router in second place becomes the Backup Designated Router (BDR), which will take over should the DR become unavailable.

DSR Direct Server Return. This is a method of load balancing in which the request is received by the load balancer, handed off to the real server, and finally, the real server replies directly to the client (not through the load balancer). This provides maximum bandwidth to the client.

Dual-mode In IEEE 802.1q tagging, dual-mode permits a single port to be both an untagged member of one VLAN, and a tagged member of many VLANs.

Edgelron This is the inexpensive low-end Layer 2 switch in the Brocade Line. These make ideal edge switches. They come in a stackable body.

eBGP External Border Gateway Protocol. This describes a BGP environment that communicates from one AS to another. The Internet uses eBGP to traverse it.

EGP Exterior Gateway Protocol. This may describe any routing protocol that communicates between administrative entities. This also describes a specific routing protocol (EGP) that was an Internet routing protocol predecessor to BGP.

EMI Electromagnetic Interference. This describes any external interference that may obstruct or degrade communication on the electrical network medium. This is an important issue with copper-based network mediums. It is overcome by shielding, and by distancing the cable from sources of EMI.

Enable mode This describes the mode in the CLI in which the user has access to the full range of CLI commands. This is where changes in the switch's configuration take place.

Ethernet This is a protocol defined in IEEE 802.3. It defines a method to transmit data using CSMA/CD. It is easily the most popular Layer 2 protocol.

ETS Enhanced Transmission Selection. ETS is used with Priority-based Flow Control. ETS establishes priorities and bandwidth limitations so that all types of traffic receive the priority and bandwidth they require for the proper operation of the server and all applications.

FastIron This is the low-end Layer 2-3 switch model of the Brocade line. These switches make good core and edge switches. They come in both chassis and stackable models.

FCoE Fibre Channel over Ethernet. FCoE refers to the ability to carry Fibre Channel traffic directly over a lossless Ethernet infrastructure known as Data Center Ethernet.

FIN Finish. This TCP segment is part of the four-way segment exchange that closes a session. It indicates the end of the session.

Firewall This is a network device that monitors traffic traveling through it. It will often permit or deny traffic based on criteria defined by the administrator. Depending on the model, it may be able to regulate traffic based on information contained in all seven OSI Layers.

Flash A special kind of memory that maintains its data even when power is lost to the device. The switch's software and Startup config are stored here. Brocade switches provide a primary and a secondary flash, so that you may store two different software images. A switch may boot from either primary or secondary flash.

Flow control This is the concept of regulating how fast data is transmitted. This is commonly associated with TCP, as it provides this service at Layer 4. This process helps to avoid overwhelming equipment that may be under a heavy load or may have a lesser capacity.

FQDN Fully-Qualified Domain Name. In DNS, this is a host name that displays the entire path of the name. For example, a relative name might be "www." The FQDN version of this name might be "www.brocade.com."

Frame This is the data unit of the Data Link Layer. In Ethernet, it is typically 1,518 bytes long (including the header).

FTP File Transfer Protocol. This is a Layer 7 protocol that provides the ability to transmit and receive files over the network.

Full-duplex This is a method of communication over a medium that provides simultaneous transmitting and receiving of data.

Full Router This describes a version of software for Brocade switches that provides the full range of Layer 3 functionality. This version provides all the features of Base Layer-3 code, but adds OSPF, BGP, NAT, PBR, and many more features.

FWLB Firewall Load Balancing. Using ServerIron, this is the process of load balancing inbound and outbound traffic through multiple firewalls. This greatly expands the capacity of the firewalls.

Gateway A gateway is a network device that provides a path from one subnet into another. This device is often a router, and will often have one interface in each subnet.

GBIC Gigabit Interface Converter. This is a module that decides which medium (e.g., copper, fiber, long-haul fiber, etc.) the modular gigabit port of a switch will use.

GSLB Global Server Load Balancing. This is a DNS proxy service provided by ServerIron. This allows load distribution and high-availability based on a geographic site level.

Half-duplex This is a method of communication of a medium that provides only either transmit or receive at any given time. You cannot transmit and receive at the same time. One must happen, and then the other.

Health check This is the process that a ServerIron uses to determine whether a remote server is available to receive incoming connections. The ServerIron provides a health check at Layer 3, Layer 4, and Layer 7.

Hello Many protocols use a "hello" protocol to discover its participants and to keep communication with them. OSPF uses a hello protocol to discover neighbors and maintain link state.

Helper address A helper address is an address configured on the gateway address of a subnet. This address acts as a DHCP proxy, so that hosts may be able to avail themselves of a DHCP server's services that is not in the same broadcast domain.

Hexadecimal This is a number system based on the number 16. It has 16 different digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. It is commonly used as a higher level representation of binary, due to the ease of conversion.

Hop This describes a transition point along a path to a destination. If a host must pass traffic through a single router in order to get to its destination, that router represents one hop. If it must pass through two routers, they represent two hops, and so on.

Hot-Standby Redundancy This is a protocol that provides full-unit redundancy for ServerIrons.

HTTP HyperText Transfer Protocol. This is a Layer 7 protocol that provides transfer of HTML data. This is the protocol a Web browser uses to call and transfer Web page data from a Web server.

Hub This is a Physical Layer device that provides a repeated signal through all of its interfaces.

IANA Internet Assigned Numbers Authority. This is an organization that regulates the assigning of IP addresses and Autonomous System numbers (in BGP).

iBGP Interior Border Gateway Protocol. This is a version of BGP which runs between multiple routers within a single AS.

ICMP Internet Control Messaging Protocol. This is a Layer 3 protocol that provides information between communicating hosts. Much of this protocol is meant to help routers share relevant information. Its most well-known service is ping ("echo" and "echo reply").

IEEE Institute of Electrical and Electronics Engineering. This is an international non-profit organization dedicated to the advancement of all technologies relating to electricity. Many networking and hardware standards are defined within this organization.

IETF Internet Engineering Task Force. This is an organization that creates Internet standards. Most well-known Internet protocols are defined here.

Internet The largest international public network that provides the open exchange of information between servers all over the world.

IP Internet Protocol. This is the most well-known Network Layer protocol. It provides the Layer 3 portion of the TCP/IP suite. It provides addressing using a 32-bit address (e.g., 192.168.123.11).

IronView This is a Brocade software suite that provides centralized management for Brocade switches.

ISP Internet Service Provider. This is a business that provides a client connectivity to the Internet.

JPEG Joint Photographic Experts Group. This is a Presentation Layer protocol for representing photographic images.

LACP Link Aggregation Control Protocol. This protocol is defined by IEEE 802.3ad, and provides for dynamic link aggregation. This protocol allows two or more network links between two devices to work together as a larger single interface.

LACPDU Link Aggregation Control Protocol Data Unit. This describes frames that are used by LACP to negotiate the link aggregation and to determine the health of the links.

LAN Local Area Network. This describes a network that encompasses a small geographic area. This would include networks within a building (or small group of buildings), small offices, homes, etc.

LDP Label Distribution Protocol. A protocol that defines how two labels in an MPLS network exchange label mapping information. For instance, how labels are distributed between LERs and LSRs.

LER Label Edge Router. A router at the edge of an MPLS network. The labeling of the packet as it enters the MPLS network is called a *push* and the removal of the label as it exists the network is called a *pop*. This is performed by the LER.

Link State Protocol A routing protocol that shares connectivity related information with other routers in order to determine the best path. TRILL will use link-state protocols to form the control plane of TRILL. Link-state protocols also continuously monitor the VLAN configuration and adjust the configuration data base in the event of changes.

LLC Logical Link Control. This is a sub-layer to the Data Link Layer. This is responsible for matching the Network Layer protocol within the Data Link Layer frame. This provides that the correct Network Layer protocol will process the data on the receiving end.

LLDP Link Layer Discovery Protocol. A mechanism whereby each switch periodically broadcasts information about itself to all of its neighbors. It's a one-way protocol, meaning that there is no acknowledgement of any of the data transmitted. Broadcasted information includes a chassis identifier, a port identifier, a time-to-live (TTL) field, and other information about the state and configuration of the device.

Loopback interface This is a virtual device. It describes the entire entity itself (e.g., a whole switch, a whole host, etc.). An address may be configured to this interface. By default, most operating systems set this address to 127.0.0.1, but additional addresses may be configured. For network devices, a loopback interface must be defined if it is to be used.

LSA Link State Advertisement. This describes the packet that OSPF routers use to communicate routing information between each other. These are various different types of LSAs for different purposes.

LSP Label Switch Paths. LSPs are established by network operators. LSPs are traffic engineered circuits/paths created to route traffic along a specific path through the network.

LSR Label Switch Routers. LSRs are the routers that perform the routing based only on the label. LSRs in MPLS networks will regularly exchange label and reachability information with each other to obtain a *virtual map* of the MPLS network.

MAC Media Access Control. This is a sub-layer to the Data Link Layer. This is responsible for the addressing at the Data Link Layer. In Ethernet, this address is a 48-bit number usually represented by six hexadecimal bytes (e.g., 12:34:56:78:9A:BC).

Mirror port This is an interface on a switch that has been defined to replicate traffic from another interface (or multiple interfaces). This permits accurate packet capturing on a switch.

MP Management Processor. This is the central management processor for a ServerIron WSM. The MP receives Layer 7 switching jobs to process and distributes them among its BPs.

MPEG Moving Pictures Experts Group. This is a Presentation Layer protocol for representing video and audio data.

MPLS Multi Protocol Label Switching. MPLS is a highly scalable, protocol agnostic, data-carrying mechanism. In an MPLS network, data packets are assigned labels, and the packet-forwarding decisions are based solely on the contents of this label, without the need to examine the packet itself.

MST Multiple Spanning Tree. This defines creating multiple instances of Spanning Tree over the same physical links and switches. Each Spanning Tree instance would have their own elections, priorities, etc.

Multicast A method of replicating packets from a single source to multiple destinations without having to replicate to all destinations. In a Network Layer setting, this is commonly used by OSPF in its communication with other routers.

Multihoming This is a method of simultaneously using more than one connection to the Internet.

NAT Network Address Translation. This protocol masks the source or destination of packets by substituting different addresses within the packet's header. This is most commonly used to translate private (RFC 1918) IP addresses to public Internet-routable addresses.

Neighbor In OSPF, this would describe another OSPF router within the same broadcast domain that is participating in the same OSPF area.

NetIron This is the high-end model of the Brocade line. These are often used by high-end ISPs or telecommunications companies. They make excellent core switches. They are chassis models and very expandable.

Network address This describes the lowest possible address of a subnet. This would be the address within a subnet in which all of the host bits were zero.

Network Layer This is the third layer of the OSI Reference Model. This layer decides how the data will reach its final destination. It is the routing layer. Data at this layer is organized into packets.

NFS Network File System. This is a protocol that provides a remote file system to be mounted locally.

NIC Network Interface Controller. This is a controller that provides physical access to your network medium.

NSSA Not So Stubby Area. In OSPF, this is an area that can receive external routes (routes from outside of its AS) and advertise them to the backbone, but it cannot receive external routes from the backbone.

Octet This is a collection of eight binary bits. This is very similar to a byte. The term "octet" is usually used when referring to an IP address. For example, using the address 192.168.123.11, the number "192" is the first octet (first eight bits of the address), "168" is the second octet, "123" is the third octet, and "11" is the fourth octet.

OSPF Open Shortest Path First. This is a link state routing protocol. It is well-suited for large networks. It makes its routing decisions based on the bandwidth cost of the path to the destination. Collections of routers are subdivided into areas.

OUI Organizationally Unique Identifier. This is the first 24-bits of a MAC address. These addresses correspond to a publicly-registered manufacturer of the NIC.

Packet This is the data unit of the Network Layer.

Packet capturing This is the process of recording duplicated frames traversing a network to troubleshoot or analyze the traffic.

PAT Port Address Translation. This is a subset of NAT. Where NAT translates one IP address into another, PAT translates according to a Layer 4 session (including the source and destination port in its tracking). This provides for multiple IP addresses to translate to the same single IP address, if desired.

Path vector This describes a routing protocol that makes its decisions based on the shortest path through larger groups of routers (Autonomous Systems).

PBR Policy-Based Routing. This protocol allows a network engineer to make exceptions to the routing table affecting only a select situation. For example, an incoming packet may be routed to a different destination to the same destination network depending on what its source address may be.

Peer In BGP, this is the router that you establish a BGP session with. Within this session, you exchange routes and advertisements.

PFC Priority-based Flow Control. The idea of PFC is to divide Ethernet traffic into different streams or priorities. That way, an Ethernet device can distinguish between the different types of traffic flowing across a link and exhibit different behaviors for different protocols.

Physical Layer This is the first layer of the OSI Reference Model. This layer is where the transition is made from logical to physical. It receives frames from the Data Link Layer, converts them to binary bits, and transmits them via electricity, laser pulses, or whatever medium the layer uses. Data at this layer is represented by binary bits.

Ping A common name for ICMP's "echo" and "echo reply" services. This provides a simple way to tell if a network host is reachable. The client sends an "echo" (or "echo request"), and the destination replies with an "echo reply."

Poison reverse This is a method of preventing routing loops in RIP. If a router receives a route advertised on the same link that it advertised the same route out of, it will set the hop count of that route to 16 (unreachable).

Policy-Based Caching This defines a method of Web caching that allows the designer to be selective about which sites are cached.

Policy Based SLB This describes a method of load balancing to certain groups of servers depending on the source IP address of the incoming client.

Predictor This is the method that a ServerIron uses to load balance to its real servers. There are several different predictors that a given virtual server can use, including round-robin, least connections, and weighted.

Presentation Layer This is the sixth layer of the OSI Reference Model. This layer formats the data it receives from the Application Layer. It prepares the data for presentation.

Private IP address This refers to RFC 1918. This standard states that there are several ranges of IP addresses that will not be publicly routed through the Internet. They are reserved for private IP networks.

Promiscuous mode When packet capturing, the client's NIC must be placed into promiscuous mode. In this mode, the NIC will receive all frames, not just the frames that are specifically addressed to it.

QoS Quality of Service. This provides the ability to shape traffic, give certain types of traffic precedence, etc. It uses priority queues defined in IEEE 802.1p.

RADIUS Remote Authentication Dial In User Service. This is service that provides centralized user authentication.

Real server This is a physical server to which a load balancer is sending data. It is the recipient of the load balancer. A load balancer answers incoming requests through a virtual server. Then, the load balancer passes the request to the real server.

Reassign threshold Between routine health checks, the reassign threshold marks the number of failed hand-offs the ServerIron will tolerate before it declares a real server inactive.

RIP Routing Information Protocol. This is a distance vector routing protocol that bases its decision by the shortest hop count. Given two paths to the same destination, it will choose the path that traverses the fewest number of routers to get to its destination.

RMON Remote Monitoring. This is a protocol that collects real-time SNMP data over time. This data is used to track a switch's performance or to help diagnose problems. Data can include CPU statistics, interface counters, and more.

Root switch In Spanning Tree Protocol, this is the switch that has won the election. The root switch's links are always active. If any other switch has a link that could cause a loop, their links are disabled.

Route redistribution This is the process in which routes from one routing protocol are translated and advertised through a different routing protocol. While this is essential in some configurations, it's generally discouraged, as much is lost in translation.

Routing Bridges These are a new type of L2 device that implement the TRILL protocol, perform L2 forwarding, and require little or no configurations. Using the configuration information distributed by the link-state protocol, R Bridges discover each other and calculate the shortest path to all other R Bridges on the VLAN.

RST Reset. This is a special TCP segment that forcibly terminates the session.

RSTP Rapid Spanning Tree Protocol. This is defined by IEEE 802.1w. It is an enhancement of the original Spanning Tree Protocol (IEEE 802.1d). It provides layer 2 loop protection with sub-second convergence.

Segment This is the data unit of the Transport Layer.

ServerIron In the Brocade line, the ServerIron is a Layer 4-7 switch. It provides load balancing, but it also provides much more. The ServerIrons come in both chassis and stackable configurations.

Session Layer This is the fifth layer of the OSI Reference Model. This layer keeps track of conversations. This may include multiple simultaneous conversations, as well as full-duplex and half-duplex conversations.

Session persistence This is the concept of load balancing a client to the same real server it started with. There are many ways to achieve this, depending on the application that is being load balanced.

sFlow This is a monitoring protocol that provides an ability to capture traffic flows for analysis. It uses a sample of all packets that traverse a given port or series of ports.

Shadow port A shadow port is a phony port defined in the real server that allows the real server to be bound to more than one virtual server on the same port.

SLB Server Load Balancing. This is the concept of using a load balancer to receive incoming requests and pass these requests through to actual servers to process. The replies are returned to the load balancer, which pass the reply back to the client. This gives the client the illusion that it has been interacting with one host.

SNMP Simple Network Management Protocol. This is a Layer 7 protocol that provides monitoring information of a network device.

Source NAT This is a method of SLB which allows the real servers to be physically detached from the ServerIron. The real server replies come back through the ServerIron because they appear to come from the ServerIron. The source address is translated to the ServerIron's IP address.

Spanning Tree Protocol This is defined in IEEE 802.1d. It is a Layer 2 protocol that prevents layer 2 loops. It uses BPDUs to discover paths through the layer 2 infrastructure. When multiple paths are found, all but one will be shut down.

SSH Secure SHell. This is a Layer 7 protocol that provides command line access to a remote host over an encrypted connection.

SSL Secure Sockets Layer. This is a protocol that provides encryption for other Layer 7 protocols. Most commonly, this is used to encrypt HTTP traffic.

SSLB Symmetric Server Load Balancing. This is a design involving multiple ServerIrons servicing the same virtual servers. The ServerIrons can work with each other in an active/standby or active/active configuration.

Stub area In OSPF, this is an area that does not receive routes from outside of the OSPF autonomous system.

Stub AS In BGP, this is an AS that has only one path in and one path out. It is not multihomed. There's no reason for it to receive all of the routes its BGP peer has to offer. It just needs a default gateway to point out the only way it can go.

Subnet This is a subdivision of a network class. A subnet extends the network portion of the subnet mask to define the needed number of networks and host addresses per network.

Subnet mask This is a 32-bit binary number that defines which portion of an IP address is the network portion, and which portion is the host portion. This is often represented in the same way as IP addresses (e.g., 255.255.255.0).

Supernet This is a method of representing a group of consecutive networks. It is usually represented in CIDR notation. This is never defined as an entity, but is used to summarize routes as they are advertised to different routers.

Switch A network device containing multiple interfaces. Traditionally, this device operates only on Layer 2. It provides an intelligent transfer from one interface to another. A switch can also be a Layer 3 device, providing routing as well as switching.

SYN Synchronize. This is the first TCP packet sent to start the initial 3-way handshake of a session. It is used to synchronize the TCP sequence numbers that will be used for the session.

SYN attack This is performed by starting the TCP 3-way handshake, but not finishing it. A SYN is sent from many clients, the server properly responds with a SYN/ACK, but the clients never respond with the final ACK. The server is left with many open sessions. It will eventually run out of resources to open new sessions.

TAC Technical Assistance Center. The Brocade TAC is the organization to call when you need assistance with your Brocade products.

TACACS Terminal Access Controller Access-Control System. This is a service that provides centralized user authentication.

TCP Transmission Control Protocol. This is a Layer 4 protocol that provides connection-oriented service. It provides windowing, flow control and acknowledgments, making certain that the data reaches its destination and in the order that was intended.

TCP/IP Transmission Control Protocol/Internet Protocol. This describes a suite of protocols on Layers 3 through 7, including IP, ICMP, TCP, UDP, HTTP, FTP, SNMP, and more. This is easily the most popular suite of protocols in use today, and it is the protocol suite of the Internet.

TCS Transparent Cache Switching. Using ServerIrons, this provides the ability to cache Web sites for end users. It is transparent because the end user does not need to change any of its configurations. The caching happens without any end-user interaction.

Telnet This is a Layer 7 protocol providing access to the command line of a remote host. This protocol has waned in popularity, due to the fact that it transmits all of its data (including username and password information) in clear text. A more secure choice, providing the same services and more, would be SSH.

Terminal emulator In the earlier days of computing, you would access the resources of a mainframe by connecting a “dumb terminal” (a device with a monitor and keyboard that you connected via serial). This would give you access to the mainframe's command line. To access the serial console of a device today, you need a serial connection, but you need software to access the serial connection and provide an interface to the command line (just like the terminal). This software is called a terminal emulator (as it emulates a terminal).

TFTP Trivial File Transfer Protocol. This is a Layer 7 protocol that provides simplistic file transfer capabilities. This is commonly used to transfer software images and configurations to and from network devices.

Three-way (3-way) handshake When initiating a TCP session, the hosts involved participate in a 3-way handshake. The client sends a TCP SYN segment to the server. The server replies to the client with a TCP SYN/ACK segment. Finally, the client replies to the server with a TCP ACK segment. The TCP session has now been initiated.

TLD Top-Level Domain. In DNS, this is the second-highest point of the name hierarchy. It is recognized by a short two to four character name. Examples of TLDs include com, net, org, and edu.

Topology Group This is used with MST. It allows an engineer to bundle several VLANs together to participate in a single instance of Spanning Tree Protocol.

Totally Stubby Area This is an OSPF area that does not receive any routes from outside its own area. This includes routes outside of the autonomous system, but also outside of its area.

Transport Layer This is the fourth layer of the OSI Reference Model. This layer makes certain that the data reaches its destination. It often incorporates windowing, flow control, and acknowledgments to achieve this. Data at this layer is organized into segments.

TRILL Transparent Interconnection of Lots of Links. TRILL is expected to be completed in the second half of 2010. TRILL will enable multi-pathing for L2 networks and remove the restrictions placed on data center environments by STP (single path) networks. Data centers with converged networks will also benefit from the multi-hop capabilities of TRILL Routing Bridges (RBridges) as they also enable multi-hop FCoE solutions.

TRILL Encapsulation Turns Ethernet frames into TRILL frames by adding a TRILL header to the frame. The new TRILL header is in exactly the same format as a legacy Ethernet header. This allows bridges (switches) that are not aware of TRILL to continue forwarding frames according to the rules they've always used.

Trunk This describes two or more links between the same devices that work together as one link.

TTL Time To Live. Many protocols have a function to age the data they provide. In DNS, for example, a host will perform a name lookup, but then it may keep the results of that lookup locally (in case it needs it again). However, the server that resolved the name stamped the data with an age (a TTL). When the TTL has expired, the host knows that it can no longer trust that information, and must do a lookup again. Tools like ping and traceroute use the TTL that is part of the IP protocol to provide their information.

UDP User Datagram Protocol. This is a Layer 4 protocol that provides connectionless service. It provides no guarantee of delivery, but, as a result, it is very fast.

URL Uniform Resource Locator. In HTTP, this provides a description of the location the file to be transferred to the client. An example of a URL would be: <http://www.brocade.com/index.html>.

URL Switching This is the concept of load balancing to real servers based on the requested URL.

VIP Virtual IP. This is an IP address that represents a function, rather than the assigned address of a NIC or host. In load balancing, this is the address that the load balancer listens for (in addition to its own configured addresses). When a connection is made to this address, it forwards the request to one of the real servers it is load balancing.

Virtual link In OSPF, all areas must be connected to the backbone (area 0). In some situations, it may be physically inconvenient (or impossible) to directly connect an area to the backbone. A virtual link provides a tunnel through another area to the backbone.

Virtual server In load balancing, this is the “fake” server that is defined on the load balancer. This defines the VIP. The load balancer answers for the virtual server requests and forwards them to the real servers.

VLAN Virtual Local Area Network. This is defined in IEEE 802.1q. A VLAN allows an engineer to subdivide a physical switch into smaller Layer 2 broadcast domains. These subdivisions may also extend to multiple switches and Layer 2 devices. It allows for the formation of a Layer 2 broadcast domain that is not restricted by the physical dimensions of the network equipment.

VLAN Group For ease of configuration, VLAN Groups allow an engineer to bundle multiple VLANs that have an identical configuration.

VLL Virtual Leased Lines. VLL is also known as Pseudo Wire Emulation (PWE). MPLS VLL is a method of providing point-to-point Ethernet /VLAN connectivity over an MPLS domain. The VLL is considered to be a Layer 2 VPN circuit across a single Layer 3 backbone.

VLSM Variable Length Subnet Mask. This is part of classless network addressing. With a variable-length mask, an engineer may more finely subdivide network classes to specific sizes and requirements.

VoIP Voice over IP. This is a protocol that provides voice services using an IP network.

VRRP Virtual Router Redundancy Protocol. This protocol provides the ability for multiple devices to share a single IP address in an active/standby fashion. VRRP-E (Extended) has provided additional features and expandability to the original protocol.

VSRRP Virtual Switch Redundancy Protocol. This is a protocol that provides Layer 2 redundancy in the event of an entire switch failing. It functions in an active/standby fashion.

WAN Wide Area Network. This describes a network extending across a larger geographic area. This often includes long distance circuits connecting geographically-diverse locations. The Internet is an example of the largest WAN.

Well-known port For TCP and UDP ports, there are several ports that have been publicly registered for use with specific Layer 7 protocols. For the most part, these are port numbers between 1 and 1024.

Wildcard mask Where a subnet mask uses a consecutive stream of 1s and 0s to define a network portion and host portion of an address, a wildcard mask uses 1s to indicate which bits of an address are changeable. In a wildcard mask, 0s indicate that this bit of the address will not change. The 0s and 1s do not have to be consecutive. This method provides a greater deal of flexibility, particularly in configuring ACLs.

Window This describes how much data can be sent before the sender must wait for an acknowledgment from the recipient.

WSM Web Switch Management Module. This is the key component of a ServerIron chassis. A WSM contains an MP and several BPs. Its function is Layer 4-7 switching. The more BPs in a WSM, the greater the capacity it provides.

BROCADE IP PRIMER

FIRST EDITION

The world of computers has changed faster than many people could even imagine. Scanning the changes over the last four decades reveals an unfathomable and exponential growth in technology. This book provides everything you need to build a solid foundation in networking technologies and design concepts. In addition to networking basics, the following topics are covered: VLANs, OSPF, MPLS, Server Load Balancing (SLB), Firewall Load Balancing (FWLB), and many others. It includes Brocade CLI code samples and self-tests so you can see what you really learned in each chapter!

JON FULLMER

\$49.95

Brocade Bookshelf
www.brocade.com/bookshelf




BROCADE