Spanning Tree Protocol Concepts

This chapter covers the following exam topics:

1.0 LAN Switching Technologies

1.3 Configure, verify, and troubleshoot STP protocols

1.3.a STP mode (PVST+ and RPVST+)

1.3.b STP root bridge selection

1.4 Configure, verify, and troubleshoot STP-related optional features

1.4.a PortFast

1.4.b BPDU guard

1.5 Configure, verify, and troubleshoot (Layer 2/Layer 3) EtherChannel

1.5.a Static

1.5.b PAGP

1.5.c LACP

Spanning Tree Protocol (STP) allows Ethernet LANs to have the added benefits of installing redundant links in a LAN, while overcoming the known problems that occur when adding those extra links. Using redundant links in a LAN design allows the LAN to keep working even when some links fail or even when some entire switches fail. Proper LAN design should add enough redundancy so that no single point of failure crashes the LAN; STP allows the design to use redundancy without causing some other problems.

STP affects many aspects of how switch forwarding logic works. Because Cisco puts the STP exam topics into the ICND2 half of the CCNA Routing and Switching exam, all the detailed examples in the ICND1 Cert Guide avoid showing redundant links in the LANs. For this ICND2 book, most of the LAN examples include redundancy. Therefore, you need to be prepared to rethink what you learned about LANs from reading the ICND1 book while thinking about LANs that have redundant links, and how STP and related features make those LANs work.

This chapter organizes the material into three sections. The first section presents core STP concepts that apply to most types of STP. STP has been improved and changed over the years, with Rapid STP (RSTP) being one major improvement. The first section looks at STP concepts without the RSTP logic added, while the second major section details RSTP concepts. The final major section discusses a small number of features that optimize and secure STP: PortFast, BPDU Guard, and EtherChannels.

As for the exam topics for this chapter, note that they all use the same three verbs: configure, verify, and troubleshoot. This chapter does not get into that level of depth on any of the specific topics, but instead lays the foundation to understand these features so that you are prepared to delve into the configuration, verification, and troubleshooting details in Chapters 3 and 4.

"Do I Know This Already?" Quiz

Take the quiz (either here, or use the PCPT software) if you want to use the score to help you decide how much time to spend on this chapter. The answers are at the bottom of the page following the quiz, and the explanations are in DVD Appendix C and in the PCPT software.

Table 2-1 "Do I Know This Already?" Foundation Topics Section-to-Question Mapping

Foundation Topics Section	Questions
Spanning Tree Protocol (IEEE 802.1D)	1–4
Rapid STP (IEEE 802.1w) Concepts	5, 6
Optional STP Features	7

1. Which of the following IEEE 802.1D port states are stable states used when STP has completed convergence? (Choose two answers.)
   - **a.** Blocking
   - **b.** Forwarding
   - **c.** Listening
   - **d.** Learning
   - **e.** Discarding
2. Which of the following are transitory IEEE 802.1D port states used only during the process of STP convergence? (Choose two answers.)
   - **a.** Blocking
   - **b.** Forwarding
   - **c.** Listening
   - **d.** Learning
   - **e.** Discarding
3. Which of the following bridge IDs wins election as root, assuming that the switches with these bridge IDs are in the same network?
   - **a.** 32769:0200.1111.1111
   - **b.** 32769:0200.2222.2222
   - **c.** 4097:0200.1111.1111
   - **d.** 4097:0200.2222.2222
   - **e.** 40961:0200.1111.1111
4. Which of the following facts determines how often a nonroot bridge or switch sends an 802.1D STP Hello BPDU message?
   - **a.** The Hello timer as configured on that switch.
   - **b.** The Hello timer as configured on the root switch.
   - **c.** It is always every 2 seconds.
   - **d.** The switch reacts to BPDUs received from the root switch by sending another BPDU 2 seconds after receiving the root BPDU.

5. Which of the following RSTP port states have the same name and purpose as a port state in traditional 802.1D STP? (Choose two answers.)

  a. Blocking

  b. Forwarding

  c. Listening

  d. Learning

  e. Discarding

6. RSTP adds some concepts to STP that enable ports to be used for a role if another port on the same switch fails. Which of the following statements correctly describe a port role that is waiting to take over for another port role? (Choose two answers.)

  a. An alternate port waits to become a root port.

  b. A backup port waits to become a root port.

  c. An alternate port waits to become a designated port.

  d. A backup port waits to become a designated port.

7. What STP feature causes an interface to be placed in the forwarding state as soon as the interface is physically active?

  a. STP

  b. EtherChannel

  c. Root Guard

  d. PortFast

Foundation Topics

Spanning Tree Protocol (IEEE 802.1D)

Without Spanning Tree Protocol (STP), a LAN with redundant links would cause Ethernet frames to loop for an indefinite period of time. With STP enabled, some switches block ports so that these ports do not forward frames. STP intelligently chooses which ports block, with two goals in mind:

- All devices in a VLAN can send frames to all other devices. In other words, STP does not block too many ports, cutting off some parts of the LAN from other parts.
- Frames have a short life and do not loop around the network indefinitely.

STP strikes a balance, allowing frames to be delivered to each device, without causing the problems that occur when frames loop through the network over and over again.

STP prevents looping frames by adding an additional check on each interface before a switch uses it to send or receive user traffic. That check: If the port is in STP forwarding state in that VLAN, use it as normal; if it is in STP blocking state, however, block all user traffic and do not send or receive user traffic on that interface in that VLAN.

Note that these STP states do not change the other information you already know about switch interfaces. The interface's state of connected/notconnect does not change. The interface's operational state as either an access or trunk port does not change. STP adds this additional STP state, with the blocking state basically disabling the interface.

In many ways, those last two paragraphs sum up what STP does. However, the details of how STP does its work can take a fair amount of study and practice. This first major section of the chapter begins by explaining the need for STP and the basic ideas of what STP does to solve the problem of looping frames. The majority of this section then looks at how STP goes about choosing which switch ports to block to accomplish STP's goals.

The Need for Spanning Tree

STP prevents three common problems in Ethernet LANs. All three problems occur as a side effect of one fact: without STP, some Ethernet frames would loop around the network for a long time (hours, days, literally forever if the LAN devices and links never failed). By default, Cisco switches run STP, but you can disable STP. Do not disable it unless you know exactly what you are doing!

Just one looping frame causes what is called a *broadcast storm*. Broadcast storms happen when any kind of Ethernet frames—broadcast frames, multicast frames, or unknown-destination unicast frames—loop around a LAN indefinitely. Broadcast storms can saturate all the links with copies of that one single frame, crowding out good frames, as well as significantly impacting end-user device performance by making the PCs process too many broadcast frames.

To help you understand how this occurs, Figure 2-1 shows a sample network in which Bob sends a broadcast frame. The dashed lines show how the switches forward the frame when STP does not exist.

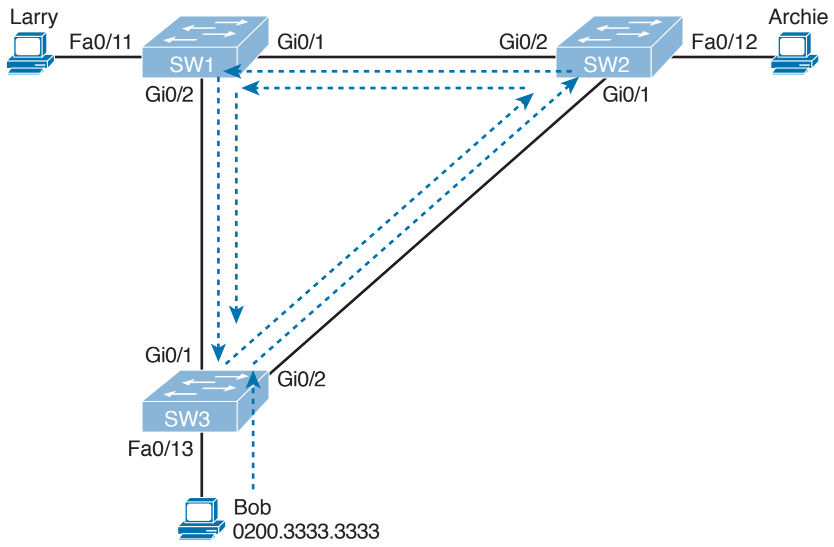


Figure 2-1 *Broadcast Storm*

Answers to the "Do I Know This Already?" quiz:

1 A, B **2** C, D **3** C **4** B **5** B, D **6** A, D **7** D

NOTE Bob's original broadcast would also be forwarded around the other direction as well, with SW3 sending a copy of the original frame out its Gi0/1 port. To reduce clutter, Figure 2-1 does not show that frame.

Remember that LAN switch? That logic tells switches to flood broadcasts out all interfaces in the same VLAN except the interface in which the frame arrived. In Figure 2-1, that means SW3 forwards Bob's frame to SW2, SW2 forwards the frame to SW1, SW1 forwards the frame back to SW3, and SW3 forwards it back to SW2 again.

When broadcast storms happen, frames like the one in Figure 2-1 keep looping until something changes—someone shuts down an interface, reloads a switch, or does something else to break the loop. Also note that the same event happens in the opposite direction. When Bob sends the original frame, SW3 also forwards a copy to SW1, SW1 forwards it to SW2, and so on.

The storm also causes a much more subtle problem called *MAC table instability*. MAC table instability means that the switches' MAC address tables keep changing, because frames with the same source MAC arrive on different ports. To see why, follow this example, in which SW3 begins Figure 2-1 with a MAC table entry for Bob, at the bottom of the figure, associated with port Fa0/13:

```
0200.3333.3333    Fa0/13    VLAN 1
```

However, now think about the switch-learning process that occurs when the looping frame goes to SW2, then SW1, and then back into SW3's Gi0/1 interface. SW3 thinks, "Hmm… the source MAC address is 0200.3333.3333, and it came in my Gi0/1 interface. Update my MAC table!" This results in the following entry on SW3, with interface Gi0/1 instead of Fa0/13:

```
0200.3333.3333    Gi0/1    VLAN 1
```

At this point, SW3 itself cannot correctly deliver frames to Bob's MAC address. At that instant, if a frame arrives at SW3 destined for Bob—a different frame than the looping frame that causes the problems—SW3 incorrectly forwards the frame out Gi0/1 to SW1, creating even more congestion.

The looping frames in a broadcast storm also cause a third problem: multiple copies of the frame arrive at the destination. Consider a case in which Bob sends a frame to Larry but none of the switches know Larry's MAC address. Switches flood frames sent to unknown destination unicast MAC addresses. When Bob sends the frame destined for Larry's MAC address, SW3 sends a copy to both SW1 and SW2. SW1 and SW2 also flood the frame, causing copies of the frame to loop. SW1 also sends a copy of each frame out Fa0/11 to Larry. As a result, Larry gets multiple copies of the frame, which may result in an application failure, if not more pervasive networking problems.

Table 2-2 summarizes the main three classes of problems that occur when STP is not used in a LAN that has redundancy.

Key Topic

Table 2-2 Three Classes of Problems Caused by Not Using STP in Redundant LANs

Problem	Description
Broadcast storms	The forwarding of a frame repeatedly on the same links, consuming significant parts of the links' capacities
MAC table instability	The continual updating of a switch's MAC address table with incorrect entries, in reaction to looping frames, resulting in frames being sent to the wrong locations
Multiple frame transmission	A side effect of looping frames in which multiple copies of one frame are delivered to the intended host, confusing the host

What IEEE 802.1D Spanning Tree Does

STP prevents loops by placing each switch port in either a forwarding state or a blocking state. Interfaces in the forwarding state act as normal, forwarding and receiving frames. However, interfaces in a blocking state do not process any frames except STP messages (and some other overhead messages). Interfaces that block do not forward user frames, do not learn MAC addresses of received frames, and do not process received user frames.

Figure 2-2 shows a simple STP tree that solves the problem shown in Figure 2-1 by placing one port on SW3 in the blocking state.

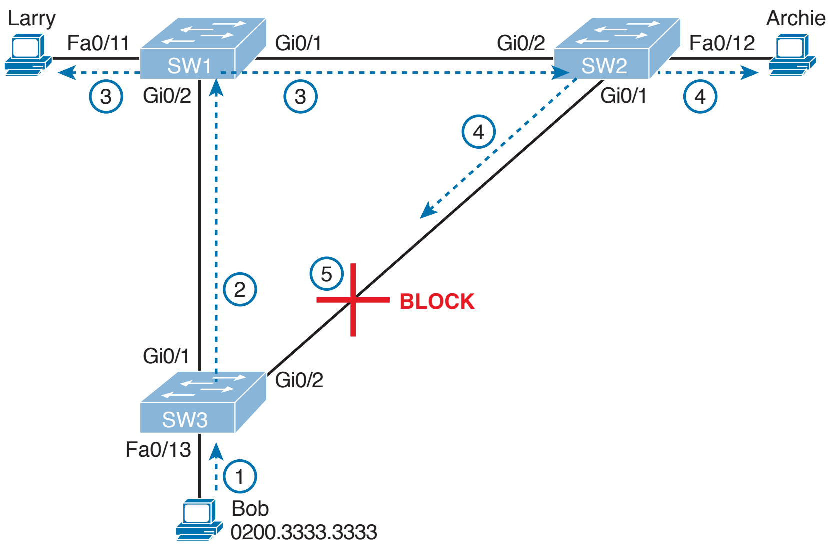


Figure 2-2 *What STP Does: Blocks a Port to Break the Loop*

Now when Bob sends a broadcast frame, the frame does not loop. As shown in the steps in the figure:

Step 1. Bob sends the frame to SW3.

Step 2. SW3 forwards the frame only to SW1, but not out Gi0/2 to SW2, because SW3's Gi0/2 interface is in a blocking state.

Step 3. SW1 floods the frame out both Fa0/11 and Gi0/1.

Step 4. SW2 floods the frame out Fa0/12 and Gi0/1.

Step 5. SW3 physically receives the frame, but it ignores the frame received from SW2 because SW3's Gi0/2 interface is in a blocking state.

With the STP topology in Figure 2-2, the switches simply do not use the link between SW2 and SW3 for traffic in this VLAN, which is the minor negative side effect of STP. However, if either of the other two links fails, STP converges so that SW3 forwards instead of blocks on its Gi0/2 interface.

NOTE The term *STP convergence* refers to the process by which the switches collectively realize that something has changed in the LAN topology and determine whether they need to change which ports block and which ports forward.

That completes the description of what STP does, placing each port into either a forwarding or blocking state. The more interesting question, and the one that takes a lot more work to understand, is the question of how and why STP makes its choices. How does STP manage to make switches block or forward on each interface? And how does it converge to change state from blocking to forwarding to take advantage of redundant links in response to network outages? The following sections answer these questions.

How Spanning Tree Works

The STP algorithm creates a spanning tree of interfaces that forward frames. The tree structure of forwarding interfaces creates a single path to and from each Ethernet link, just like you can trace a single path in a living, growing tree from the base of the tree to each leaf.

NOTE STP was created before LAN switches even existed. In those days, Ethernet bridges used STP. Today, switches play the same role as bridges, implementing STP. However, many STP terms still refer to bridge. For the purposes of STP and this chapter, consider the terms *bridge* and *switch* synonymous.

The process used by STP, sometimes called the *spanning-tree algorithm* (STA), chooses the interfaces that should be placed into a forwarding state. For any interfaces not chosen to be in a forwarding state, STP places the interfaces in blocking state. In other words, STP simply picks which interfaces should forward, and any interfaces left over go to a blocking state.

STP uses three criteria to choose whether to put an interface in forwarding state:

- STP elects a root switch. STP puts all working interfaces on the root switch in forwarding state.
- Each nonroot switch considers one of its ports to have the least administrative cost between itself and the root switch. The cost is called that switch's *root cost*. STP places its port that is part of the least root cost path, called that switch's *root port* (RP), in forwarding state.
- Many switches can attach to the same Ethernet segment, but in modern networks, normally two switches connect to each link. The switch with the lowest root cost, as compared with the other switches attached to the same link, is placed in forwarding state.

That switch is the designated switch, and that switch's interface, attached to that segment, is called the *designated port* (DP).

NOTE The real reason the root switches place all working interfaces in a forwarding state is that all its interfaces will become DPs, but it is easier to just remember that all the root switches' working interfaces will forward frames.

All other interfaces are placed in blocking state. Table 2-3 summarizes the reasons STP places a port in forwarding or blocking state.

Key Topic

Table 2-3 STP: Reasons for Forwarding or Blocking

Characterization of Port	STP State	Description
All the root switch's ports	Forwarding	The root switch is always the designated switch on all connected segments.
Each nonroot switch's root port	Forwarding	The port through which the switch has the least cost to reach the root switch (lowest root cost).
Each LAN's designated port	Forwarding	The switch forwarding the Hello on to the segment, with the lowest root cost, is the designated switch for that segment.
All other working ports	Blocking	The port is not used for forwarding user frames, nor are any frames received on these interfaces considered for forwarding.

NOTE STP only considers working interfaces (those in a connected state). Failed interfaces (for example, interfaces with no cable installed) or administratively shutdown interfaces are instead placed into an STP disabled state. So, this section uses the term *working ports* to refer to interfaces that could forward frames if STP placed the interface into a forwarding state.

The STP Bridge ID and Hello BPDU

The STA begins with an election of one switch to be the root switch. To better understand this election process, you need to understand the STP messages sent between switches as well as the concept and format of the identifier used to uniquely identify each switch.

The STP *bridge ID* (BID) is an 8-byte value unique to each switch. The bridge ID consists of a 2-byte priority field and a 6-byte system ID, with the system ID being based on a universal (burned-in) MAC address in each switch. Using a burned-in MAC address ensures that each switch's bridge ID will be unique.

STP defines messages called *bridge protocol data units* (BPDU), which switches use to exchange information with each other. The most common BPDU, called a Hello BPDU, lists many details, including the sending switch's BID. By listing its own unique BID, switches can tell which switch sent which Hello BPDU. Table 2-4 lists some of the key information in the Hello BPDU.

Key Topic

Table 2-4 Fields in the STP Hello BPDU

Field	Description
Root bridge ID	The bridge ID of the switch the sender of this Hello currently believes to be the root switch
Sender's bridge ID	The bridge ID of the switch sending this Hello BPDU
Sender's root cost	The STP cost between this switch and the current root
Timer values on the root switch	Includes the Hello timer, MaxAge timer, and forward delay timer

For the time being, just keep the first three items from Table 2-4 in mind as the following sections work through the three steps in how STP chooses the interfaces to place into a forwarding state. Next, the text examines the three main steps in the STP process.

Electing the Root Switch

Switches elect a root switch based on the BIDs in the BPDUs. The root switch is the switch with the lowest numeric value for the BID. Because the two-part BID starts with the priority value, essentially the switch with the lowest priority becomes the root. For example, if one switch has priority 4096, and another switch has priority 8192, the switch with priority 4096 wins, regardless of what MAC address was used to create the BID for each switch.

If a tie occurs based on the priority portion of the BID, the switch with the lowest MAC address portion of the BID is the root. No other tiebreaker should be needed because switches use one of their own universal (burned-in) MAC addresses as the second part of their BIDs. So if the priorities tie, and one switch uses a MAC address of 0200.0000.0000 as part of the BID and the other uses 0911.1111.1111, the first switch (MAC 0200.0000.0000) becomes the root switch.

STP elects a root switch in a manner not unlike a political election. The process begins with all switches claiming to be the root by sending Hello BPDUs listing their own BID as the root BID. If a switch hears a Hello that lists a better (lower) BID, that switch stops advertising itself as root and starts forwarding the superior Hello. The Hello sent by the better switch lists the better switch's BID as the root. It works like a political race in which a less-popular candidate gives up and leaves the race, throwing his support behind the more popular candidate. Eventually, everyone agrees which switch has the best (lowest) BID, and everyone supports the elected switch—which is where the political race analogy falls apart.

NOTE A better Hello, meaning that the listed root's BID is better (numerically lower), is called a *superior Hello*; a worse Hello, meaning that the listed root's BID is not as good (numerically higher), is called an *inferior Hello*.

Figure 2-3 shows the beginning of the root election process. In this case, SW1 has advertised itself as root, as have SW2 and SW3. However, SW2 now believes that SW1 is a better root, so SW2 is now forwarding the Hello originating at SW1. So, at this point, the figure shows SW1 is saying Hello, claiming to be root; SW2 agrees, and is forwarding SW1's Hello that lists SW1 as root; but, SW3 is still claiming to be best, sending its own Hello BPDUs, listing SW3's BID as the root.

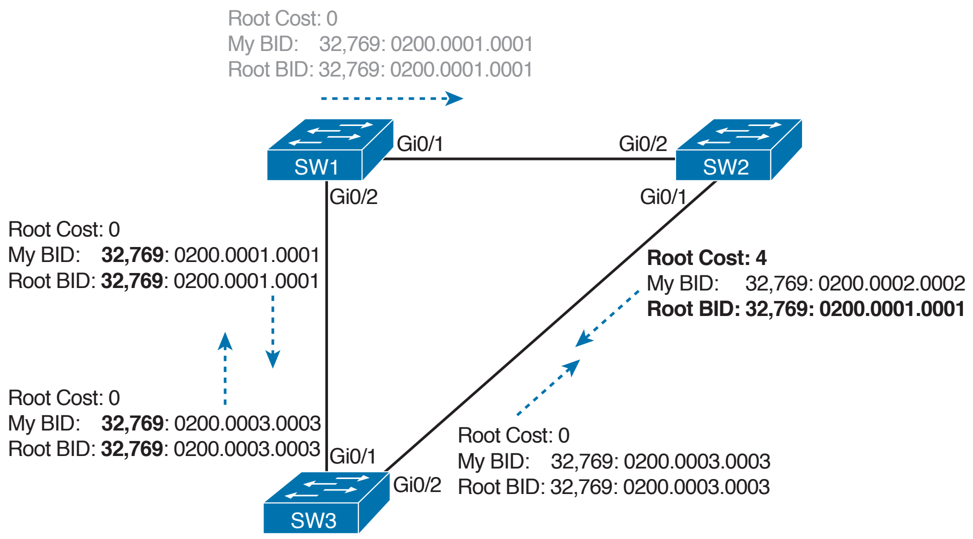

Root Cost: 0
My BID: 32,769: 0200.0001.0001
Root BID: 32,769: 0200.0001.0001
SW1
Gi0/1
Gi0/2
SW2
Gi0/2
Gi0/1
Root Cost: 0
My BID: 32,769: 0200.0001.0001
Root BID: 32,769: 0200.0001.0001
Root Cost: 4
My BID: 32,769: 0200.0002.0002
Root BID: 32,769: 0200.0001.0001
Root Cost: 0
My BID: 32,769: 0200.0003.0003
Root BID: 32,769: 0200.0003.0003
Gi0/1
Gi0/2
SW3
Root Cost: 0
My BID: 32,769: 0200.0003.0003
Root BID: 32,769: 0200.0003.0003

Figure 2-3 *Beginnings of the Root Election Process*

Two candidates still exist in Figure 2-3: SW1 and SW3. So who wins? Well, from the BID, the lower-priority switch wins; if a tie occurs, the lower MAC address wins. As shown in the figure, SW1 has a lower BID (32769:0200.0001.0001) than SW3 (32769:0200.0003.0003), so SW1 wins, and SW3 now also believes that SW1 is the better switch. Figure 2-4 shows the resulting Hello messages sent by the switches.

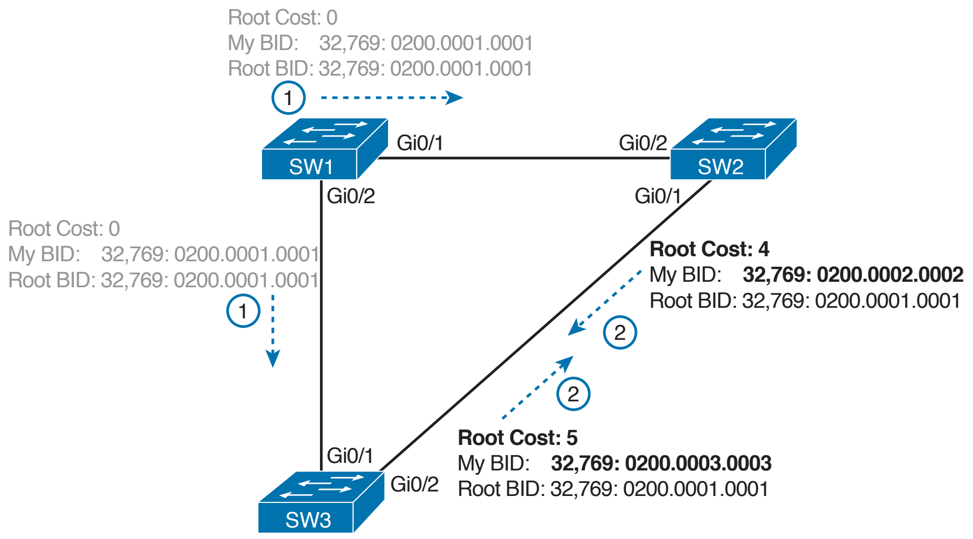

Root Cost: 0
My BID: 32,769: 0200.0001.0001
Root BID: 32,769: 0200.0001.0001
1
SW1
Gi0/1
Gi0/2
SW2
Gi0/2
Gi0/1
Root Cost: 0
My BID: 32,769: 0200.0001.0001
Root BID: 32,769: 0200.0001.0001
1
Root Cost: 4
My BID: 32,769: 0200.0002.0002
Root BID: 32,769: 0200.0001.0001
2
2
Root Cost: 5
My BID: 32,769: 0200.0003.0003
Root BID: 32,769: 0200.0001.0001
Gi0/1
Gi0/2
SW3

Figure 2-4 *SW1 Wins the Election*

After the election is complete, only the root switch continues to originate STP Hello BPDU messages. The other switches receive the Hellos, update the sender's BID field (and root cost field), and forward the Hellos out other interfaces. The figure reflects this fact, with SW1 sending Hellos at Step 1, and SW2 and SW3 independently forwarding the Hello out their other interfaces at Step 2.

Summarizing, the root election happens through each switch claiming to be root, with the best switch being elected based on the numerically lowest BID. Breaking down the BID into its components, the comparisons can be made as

Key Topic

- The lowest priority
- If that ties, the lowest switch MAC address

Choosing Each Switch's Root Port

The second part of the STP process occurs when each nonroot switch chooses its one and only *root port*. A switch's RP is its interface through which it has the least STP cost to reach the root switch (least root cost).

The idea of a switch's cost to reach the root switch can be easily seen for humans. Just look at a network diagram that shows the root switch, lists the STP cost associated with each switch port, and identifies the nonroot switch in question. Switches use a different process than looking at a network diagram, of course, but using a diagram can make it easier to learn the idea.

Figure 2-5 shows just such a figure, with the same three switches shown in the last several figures. SW1 has already won the election as root, and the figure considers the cost from SW3's perspective.

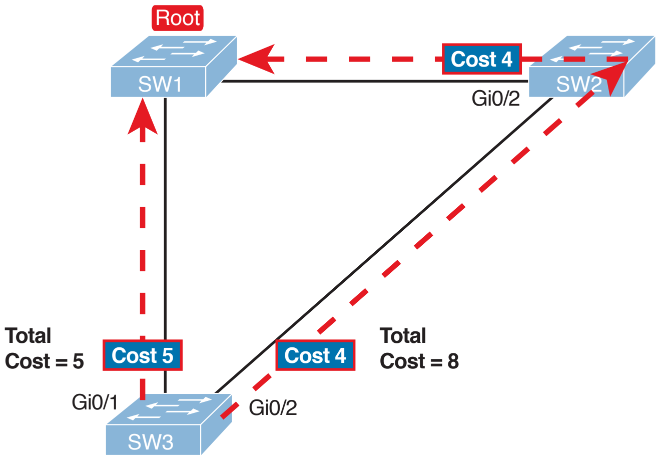


Figure 2-5 *How a Human Might Calculate STP Cost from SW3 to the Root (SW1)*

SW3 has two possible physical paths to send frames to the root switch: the direct path to the left, and the indirect path to the right through switch SW2. The cost is the sum of the costs of all the *switch ports the frame would exit* if it flowed over that path. (The calculation ignores the inbound ports.) As you can see, the cost over the direct path out SW3's G0/1 port has a total cost of 5, and the other path has a total cost of 8. SW3 picks its G0/1 port as root port because it is the port that is part of the least-cost path to send frames to the root switch.

Switches come to the same conclusion, but using a different process. Instead, they add their local interface STP cost to the root cost listed in each received Hello BPDU. The STP port cost is simply an integer value assigned to each interface, per VLAN, for the purpose of providing an objective measurement that allows STP to choose which interfaces to add to the STP topology. The switches also look at their neighbor's root cost, as announced in Hello BPDUs received from each neighbor.

Figure 2-6 shows an example of how switches calculate their best root cost and then choose their root port, using the same topology and STP costs as shown in Figure 2-5. STP on SW3 calculates its cost to reach the root over the two possible paths by adding the advertised cost (in Hello messages) to the interface costs listed in the figure.

Focus on the process for a moment. The root switch sends Hellos, with a listed root cost of 0. The idea is that the root's cost to reach itself is 0.

Next, look on the left of the figure. SW3 takes the received cost (0) from the Hello sent by SW1, and adds the interface cost (5) of the interface on which that Hello was received. SW3 calculates that the cost to reach the root switch, out that port (G0/1), is 5.

On the right side, SW2 has realized its best cost to reach the root is cost 4. So, when SW2 forwards the Hello toward SW3, SW2 lists a root cost 4. SW3's STP port cost on port G0/2 is 4, so SW3 determines a total cost to reach root out its G0/2 port of 8.

As a result of the process depicted in Figure 2-6, SW3 chooses Gi0/1 as its RP, because the cost to reach the root switch through that port (5) is lower than the other alternative (Gi0/2, cost 8). Similarly, SW2 chooses Gi0/2 as its RP, with a cost of 4 (SW1's advertised cost of 0 plus SW2's Gi0/2 interface cost of 4). Each switch places its root port into a forwarding state.

Key Topic

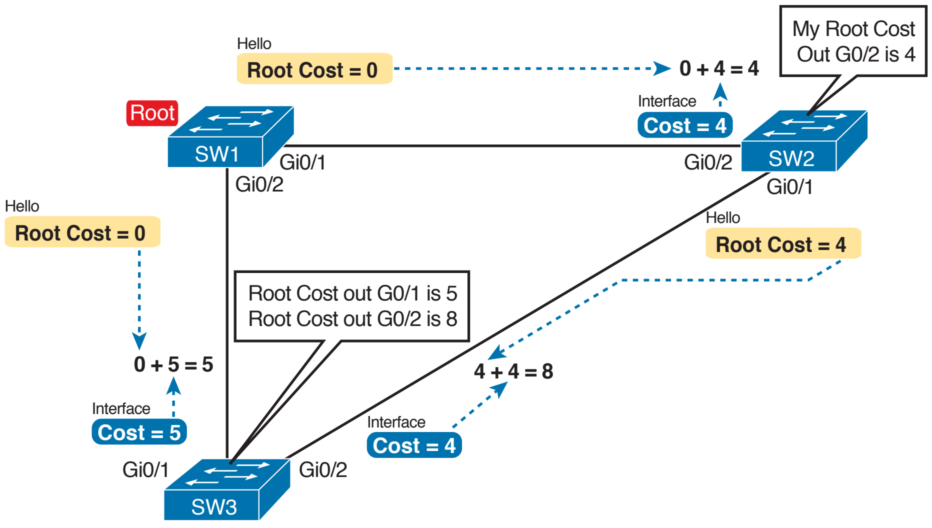


Figure 2-6 *How STP Actually Calculates the Cost from SW3 to the Root*

In more complex topologies, the choice of root port will not be so obvious. Chapter 4, "LAN Troubleshooting," discusses these more complex examples, including the tiebreakers to use if the root costs tie.

Choosing the Designated Port on Each LAN Segment

STP's final step to choose the STP topology is to choose the designated port on each LAN segment. The designated port (DP) on each LAN segment is the switch port that advertises the lowest-cost Hello onto a LAN segment. When a nonroot switch forwards a Hello, the nonroot switch sets the root cost field in the Hello to that switch's cost to reach the root. In effect, the switch with the lower cost to reach the root, among all switches connected to a segment, becomes the DP on that segment.

For example, earlier Figure 2-4 shows in bold text the parts of the Hello messages from both SW2 and SW3 that determine the choice of DP on that segment. Note that both SW2 and SW3 list their respective cost to reach the root switch (cost 4 on SW2 and cost 5 on SW3). SW2 lists the lower cost, so SW2's Gi0/1 port is the designated port on that LAN segment.

All DPs are placed into a forwarding state; so in this case, SW2's Gi0/1 interface will be in a forwarding state.

If the advertised costs tie, the switches break the tie by choosing the switch with the lower BID. In this case, SW2 would also have won, with a BID of 32769:0200.0002.0002 versus SW3's 32769:0200.0003.0003.

NOTE Two additional tiebreakers are needed in some cases, although these would be unlikely today. A single switch can connect two or more interfaces to the same collision domain by connecting to a hub. In that case, the one switch hears its own BPDUs. So, if a switch ties with itself, two additional tiebreakers are used: the lowest interface STP priority and, if that ties, the lowest internal interface number.

The only interface that does not have a reason to be in a forwarding state on the three switches in the examples shown in Figures 2-3 through 2-6 is SW3's Gi0/2 port. So, the STP process is now complete. Table 2-5 outlines the state of each port and shows why it is in that state.

Table 2-5 State of Each Interface

Switch Interface	State	Reason Why the Interface Is in Forwarding State
SW1, Gi0/1	Forwarding	The interface is on the root switch, so it becomes the DP on that link.
SW1, Gi0/2	Forwarding	The interface is on the root switch, so it becomes the DP on that link.
SW2, Gi0/2	Forwarding	The root port of SW2.
SW2, Gi0/1	Forwarding	The designated port on the LAN segment to SW3.
SW3, Gi0/1	Forwarding	The root port of SW3.
SW3, Gi0/2	Blocking	Not the root port and not the designated port.

Influencing and Changing the STP Topology

Switches do not just use STP once and never again. The switches continually watch for changes. Those changes can be because a link or switch fails or it can be a new link that can now be used. The configuration can change in a way that changes the STP topology. This section briefly discusses the kinds of things that change the STP topology, either through configuration or through changes in the status of devices and links in the LAN.

Making Configuration Changes to Influence the STP Topology

The network engineers can choose to change the STP settings to then change the choices STP makes in a given LAN. Two main tools available to the engineer are to configure the bridge ID and to change STP port costs.

Switches have a way to create a default BID, by taking a default priority value, and adding a universal MAC address that comes with the switch hardware. However, engineers typically want to choose which switch becomes the root. Chapter 3, "Spanning Tree Protocol Implementation," shows how to configure a Cisco switch to override its default BID setting to make a switch become root.

Port costs also have default values, per port, per VLAN. You can configure these port costs, or you can use the default values. Table 2-6 lists the default port costs suggested by IEEE. IOS on Cisco switches has long used the default settings as defined in the 1998 version of the 802.1D standard. The newer standard, useful when using links faster than 10 Gbps, can be used by adding a single configuration command to each switch (**spanning-tree pathcost method long**).

Key Topic

Table 2-6 Default Port Costs According to IEEE

Ethernet Speed	IEEE Cost: 1998 (and Before)	IEEE Cost: 2004 (and After)
10 Mbps	100	2,000,000
100 Mbps	19	200,000
1 Gbps	4	20,000
10 Gbps	2	2000
100 Gbps	N/A	200
1 Tbps	N/A	20

With STP enabled, all working switch interfaces will settle into an STP forwarding or blocking state, even access ports. For switch interfaces connected to hosts or routers, which do not use STP, the switch still forwards Hellos on to those interfaces. By virtue of being the only device sending a Hello onto that LAN segment, the switch is sending the least-cost Hello on to that LAN segment, making the switch become the designated port on that LAN segment. So, STP puts working access interfaces into a forwarding state as a result of the designated port part of the STP process.

Reacting to State Changes That Affect the STP Topology

Once the engineer has finished all STP configuration, the STP topology should settle into a stable state and not change, at least until the network topology changes. This section examines the ongoing operation of STP while the network is stable, and then it covers how STP converges to a new topology when something changes.

The root switch sends a new Hello BPDU every 2 seconds by default. Each nonroot switch forwards the Hello on all DPs, but only after changing items listed in the Hello. The switch sets the root cost to that local switch's calculated root cost. The switch also sets the "sender's bridge ID" field to its own bridge ID. (The root's bridge ID field is not changed.)

By forwarding the received (and changed) Hellos out all DPs, all switches continue to receive Hellos every 2 seconds. The following steps summarize the steady-state operation when nothing is currently changing in the STP topology:

Key Topic

Step 1. The root creates and sends a Hello BPDU, with a root cost of 0, out all its working interfaces (those in a forwarding state).

Step 2. The nonroot switches receive the Hello on their root ports. After changing the Hello to list their own BID as the sender's BID, and listing that switch's root cost, the switch forwards the Hello out all designated ports.

Step 3. Steps 1 and 2 repeat until something changes.

Each switch relies on these periodically received Hellos from the root as a way to know that its path to the root is still working. When a switch ceases to receive the Hellos, or receives a Hello that lists different details, something has failed, so the switch reacts and starts the process of changing the spanning-tree topology.

How Switches React to Changes with STP

For various reasons, the convergence process requires the use of three timers. Note that all switches use the timers as dictated by the root switch, which the root lists in its periodic Hello BPDU messages. Table 2-7 describes the timers.

Key Topic

Table 2-7 STP Timers

Timer	Default Value	Description
Hello	2 seconds	The time period between Hellos created by the root.
MaxAge	10 times Hello	How long any switch should wait, after ceasing to hear Hellos, before trying to change the STP topology.
Forward delay	15 seconds	Delay that affects the process that occurs when an interface changes from blocking state to forwarding state. A port stays in an interim listening state, and then an interim learning state, for the number of seconds defined by the forward delay timer.

If a switch does not get an expected Hello BPDU within the Hello time, the switch continues as normal. However, if the Hellos do not show up again within MaxAge time, the switch reacts by taking steps to change the STP topology. With default settings, MaxAge is 20 seconds (10 times the default Hello timer of 2 seconds). So, a switch would go 20 seconds without hearing a Hello before reacting.

After MaxAge expires, the switch essentially makes all its STP choices again, based on any Hellos it receives from other switches. It reevaluates which switch should be the root switch. If the local switch is not the root, it chooses its RP. And it determines whether it is DP on each of its other links. The best way to describe STP convergence is to show an example using the same familiar topology. Figure 2-7 shows the same familiar figure, with SW3's Gi0/2 in a blocking state, but SW1's Gi0/2 interface has just failed.

SW3 reacts to the change because SW3 fails to receive its expected Hellos on its Gi0/1 interface. However, SW2 does not need to react because SW2 continues to receive its periodic Hellos in its Gi0/2 interface. In this case, SW3 reacts either when MaxAge time passes without hearing the Hellos, or as soon as SW3 notices that interface Gi0/1 has failed. (If the interface fails, the switch can assume that the Hellos will not be arriving in that interface anymore.)

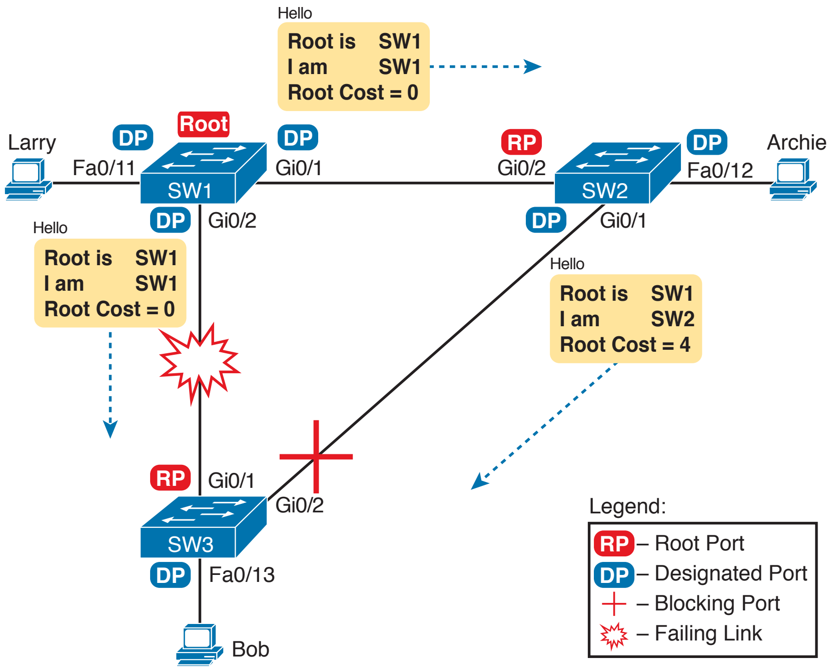


Figure 2-7 *Initial STP State Before SW1-SW3 Link Fails*

Now that SW3 can act, it begins by reevaluating the choice of root switch. SW3 still receives the Hellos from SW2, as forwarded from the root (SW1). SW1 still has a lower BID than SW3; otherwise, SW1 would not have already been the root. So, SW3 decides that SW1 is still the best switch and that SW3 is not the root.

Next, SW3 reevaluates its choice of RP. At this point, SW3 is receiving Hellos on only one interface: Gi0/2. Whatever the calculated root cost, Gi0/2 becomes SW3's new RP. (The cost would be 8, assuming the STP costs had no changes since Figures 2-5 and 2-6.)

SW3 then reevaluates its role as DP on any other interfaces. In this example, no real work needs to be done. SW3 was already DP on interface Fa0/13, and it continues to be the DP because no other switches connect to that port.

Changing Interface States with STP

STP uses the idea of roles and states. *Roles*, like root port and designated port, relate to how STP analyzes the LAN topology. *States*, like forwarding and blocking, tell a switch whether to send or receive frames. When STP converges, a switch chooses new port roles, and the port roles determine the state (forwarding or blocking).

Switches can simply move immediately from forwarding to blocking state, but they must take extra time to transition from blocking state to forwarding state. For instance, when a switch formerly used port G0/1 as its RP (a role), that port was in a forwarding state. After convergence, G0/1 might be neither an RP nor DP; the switch can immediately move that port to a blocking state.

When a port that formerly blocked needs to transition to forwarding, the switch first puts the port through two intermediate interface states. These temporary states help prevent temporary loops:

Key Topic

- **Listening:** Like the blocking state, the interface does not forward frames. The switch removes old stale (unused) MAC table entries for which no frames are received from each MAC address during this period. These stale MAC table entries could be the cause of the temporary loops.
- **Learning:** Interfaces in this state still do not forward frames, but the switch begins to learn the MAC addresses of frames received on the interface.

STP moves an interface from blocking to listening, then to learning, and then to forwarding state. STP leaves the interface in each interim state for a time equal to the forward delay timer, which defaults to 15 seconds. As a result, a convergence event that causes an interface to change from blocking to forwarding requires 30 seconds to transition from blocking to forwarding. In addition, a switch might have to wait MaxAge seconds before even choosing to move an interface from blocking to forwarding state.

For example, follow what happens with an initial STP topology as shown in Figures 2-3 through 2-6, with the SW1-to-SW3 link failing as shown in Figure 2-7. If SW1 simply quit sending Hello messages to SW3, but the link between the two did not fail, SW3 would wait MaxAge seconds before reacting (20 seconds is the default). SW3 would actually quickly choose its ports' STP roles, but then wait 15 seconds each in listening and learning states on interface Gi0/2, resulting in a 50-second convergence delay.

Table 2-8 summarizes spanning tree's various interface states for easier review.

Key Topic

Table 2-8 IEEE 802.1D Spanning-Tree States

State	Forwards Data Frames?	Learns MACs Based on Received Frames?	Transitory or Stable State?
Blocking	No	No	Stable
Listening	No	No	Transitory
Learning	No	Yes	Transitory
Forwarding	Yes	Yes	Stable
Disabled	No	No	Stable

Rapid STP (IEEE 802.1w) Concepts

The original STP worked well given the assumptions about networks and networking devices in that era. However, as with any computing or networking standard, as time passes, hardware and software capabilities improve, so new protocols emerge to take advantage of those new capabilities. For STP, one of the most significant improvements over time has been the introduction of Rapid Spanning Tree Protocol (RSTP), introduced as standard IEEE 802.1w.

Before getting into the details of RSTP, it helps to make sense of the standards numbers a bit. 802.1w was actually an amendment to the 802.1D standard. 802.1D was published anew in 1998 (and a few times before that). After the 1998 version of 802.1D, the IEEE published the 802.1w amendment in 2001. Later, when the IEEE 802.1 committee updated the 802.1D standard in 2004, the IEEE pulled the 802.1w amendment details into the 802.1D-2004 standard.

So, why do we care? Sometimes people use the term *STP* to refer to the original pre-RSTP rules for STP. Some use STP to mean anything in the 802.1D standard, which now includes RSTP. So for real life, make sure you know what people mean when they use STP: do they mean STP to include RSTP concepts, or not? For this book, throughout the book, if the distinction between STP and RSTP matters, the book will use STP for the original STP rules and RSTP for the new ones introduced by 802.1w.

NOTE The IEEE sells its standards, but through the “Get IEEE 802” program, you can get free PDFs of the current 802 standards. To read about RSTP 802.1w, you will need to download the 802.1D standard, and then look for the sections about RSTP.

Now on to the details about RSTP in this chapter. There are similarities between RSTP and STP, so this section next compares and contrasts the two. Following that, the rest of this section discusses the concepts unique to RSTP that are not found in STP: alternate root ports, different port states, backup ports, and the port roles used by RSTP.

Comparing STP and RSTP

RSTP (802.1w) works just like STP (the original 802.1D) in several ways:

Key Topic

- It elects the root switch using the same parameters and tiebreakers.
- It elects the root port on nonroot switches with the same rules.
- It elects designated ports on each LAN segment with the same rules.
- It places each port in either forwarding or blocking state, although RSTP calls the blocking state the discarding state.

In fact, RSTP works so much like STP that they can both be used in the same network. RSTP and STP switches can be deployed in the same network, with RSTP features working in switches that support it, and traditional 802.1D STP features working in the switches that support only STP.

With all these similarities, you might be wondering why the IEEE bothered to create RSTP in the first place. The overriding reason is convergence. STP takes a relatively long time to converge (50 seconds with the default settings when all the wait times must be followed). RSTP improves network convergence when topology changes occur, usually converging within a few seconds (or in slow conditions, in about 10 seconds).

IEEE 802.1w RSTP changes and adds to IEEE 802.1D STP in ways that avoid waiting on STP timers, resulting in quick transitions from forwarding to blocking state and vice versa. Specifically, RSTP, compared to STP, defines more cases in which the switch can avoid waiting for a timer to expire, such as the following:

- Adds a new mechanism to replace the root port, without any waiting to reach a forwarding state (in some conditions)
- Adds a new mechanism to replace a designated port, without any waiting to reach a forwarding state (in some conditions)
- Lowers waiting times for cases in which RSTP must wait

For instance, when a link remains up, but Hello BPDUs simply stop arriving regularly on a port, STP requires a switch to wait for MaxAge seconds. STP defines the MaxAge timers based on ten times the Hello timer, or 20 seconds, by default. RSTP shortens this timer, defining MaxAge as three times the Hello timer.

The best way to get a sense for these mechanisms is to see how the RSTP alternate port and the backup port both work. RSTP uses the term *alternate port* to refer to a switch's other ports that could be used as root port if the root port ever fails. The *backup port* concept provides a backup port on the local switch for a designated port, but only applies to some topologies that frankly do not happen often with a modern network design. However, both are instructive about how RSTP works. Table 2-9 lists these RSTP port roles.

Key Topic

Table 2-9 Port Roles in 802.1w RSTP

Function	Port Role
Nonroot switch's best path to the root	Root port
Replaces the root port when the root port fails	Alternate port
Switch port designated to forward onto a collision domain	Designated port
Replaces a designated port when a designated port fails	Backup port
Port that is administratively disabled	Disabled port

RSTP and the Alternate (Root) Port Role

With STP, each nonroot switch places one port in the STP root port (RP) role. RSTP follows that same convention, with the same exact rules for choosing the RP. RSTP then takes another step, naming other possible RPs, identifying them as *alternate ports*.

To be an alternate port, both the RP and the alternate port must receive Hellos that identify the same root switch. For instance, in Figure 2-8, SW1 is the root. SW3 will receive Hello BPDUs on two ports: G0/1 and G0/2. Both Hellos list SW1's bridge ID (BID) as the root switch, so whichever port is not the root port meets the criteria to be an alternate port. SW3 picks G0/1 as its root port in this case, and then makes G0/2 an alternate port.

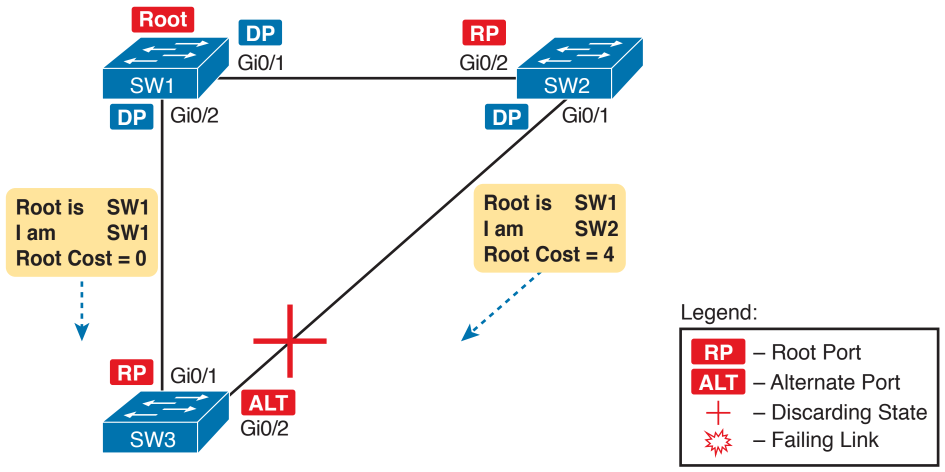


Figure 2-8 *Example of SW3 Making G0/2 Become an Alternate Port*

An alternate port basically works like the second-best option for root port. The alternate port can take over for the former root port, often very rapidly, without requiring a wait in other interim RSTP states. For instance, when the root port fails, or when Hellos stop arriving on the original root port, the switch changes the former root port's role and state: (a) the role from root port to a disabled port, and (b) the state from forwarding to discarding (the equivalent of STP's blocking state). Then, without waiting on any timers, the switch changes roles and state for the alternate port: its role changes to be the root port, with a forwarding state.

Notably, the new root port also does not need to spend time in other states, such as learning state, instead moving immediately to forwarding state.

Figure 2-9 shows an example of RSTP convergence. SW3's root port before the failure shown in this figure is SW3's G0/1, the link connected directly to SW1 (the root switch). Then SW3's link to SW1 fails as shown in Step 1 of the figure.

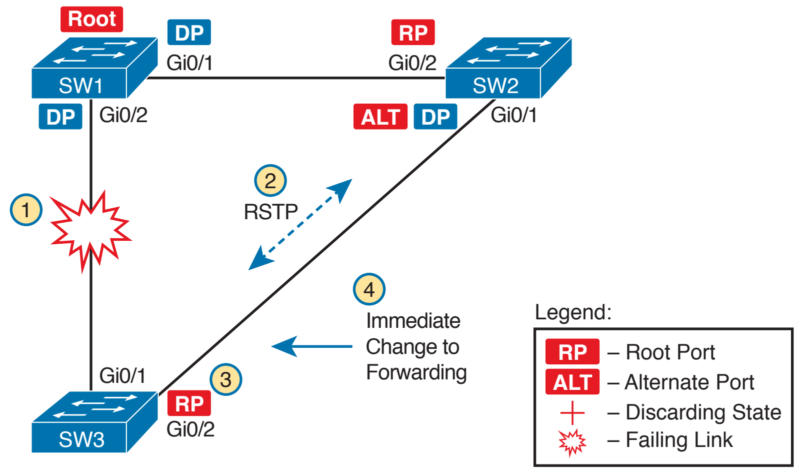


Figure 2-9 *Convergence Events with SW3 G0/1 Failure*

Following the steps in Figure 2-9:

Step 1. The link between SW1 and SW3 fails, so that SW3's current root port (Gi0/1) fails.

Step 2. SW3 and SW2 exchange RSTP messages to confirm that SW3 will now transition its former alternate port (Gi0/2) to be the root port. This action causes SW2 to flush the required MAC table entries.

Step 3. SW3 transitions G0/1 to the disabled role and G0/2 to the root port role.

Step 4. SW3 transitions G0/2 to a forwarding state immediately, without using learning state, because this is one case in which RSTP knows the transition will not create a loop.

As soon as SW3 realizes its G0/1 interface has failed, the process shown in the figure takes very little time. None of the processes rely on timers, so as soon as the work can be done, the convergence completes. (This particular convergence example takes about 1 second in a lab.)

RSTP States and Processes

The depth of the previous example does not point out all details of RSTP, of course; however, the example does show enough details to discuss RSTP states and internal processes.

Both STP and RSTP use *port states*, but with some differences. First, RSTP keeps both the learning and forwarding states as compared with STP, for the same purposes. However, RSTP does not even define a listening state, finding it unnecessary. Finally, RSTP renames the blocking state to the discarding state, and redefines its use slightly.

RSTP uses the discarding state for what 802.1D defines as two states: disabled state and blocking state. Blocking should be somewhat obvious by now: The interface can work physically, but STP/RSTP chooses to not forward traffic to avoid loops. STP's disabled state simply meant that the interface was administratively disabled. RSTP just combines those into a single discarding state. Table 2-10 shows the list of STP and RSTP states for comparison purposes.

Key Topic

Table 2-10 Port States Compared: 802.1D STP and 802.1w RSTP

Function	802.1D State	802.1w State
Port is administratively disabled	Disabled	Discarding
Stable state that ignores incoming data frames and is not used to forward data frames	Blocking	Discarding
Interim state without MAC learning and without forwarding	Listening	Not used
Interim state with MAC learning and without forwarding	Learning	Learning
Stable state that allows MAC learning and forwarding of data frames	Forwarding	Forwarding

RSTP also changes some processes and message content (compared to STP) to speed convergence. For example, STP waits for a time (forward delay) in both listening and learning states. The reason for this delay in STP is that, at the same time, the switches have all been told to time out their MAC table entries. When the topology changes, the existing MAC table entries may actually cause a loop. With STP, the switches all tell each other (with BPDU messages) that the topology has changed, and to time out any MAC table entries using the forward delay timer. This removes the entries, which is good, but it causes the need to wait in both listening and learning state for forward delay time (default 15 seconds each).

RSTP, to converge more quickly, avoids relying on timers. RSTP switches tell each other (using messages) that the topology has changed. Those messages also direct neighboring switches to flush the contents of their MAC tables in a way that removes all the potentially loop-causing entries, without a wait. As a result, RSTP creates more scenarios in which a formerly discarding port can immediately transition to a forwarding state, without waiting, and without using the learning state, as shown in the example in Figure 2-9.

RSTP and the Backup (Designated) Port Role

The RSTP backup port role acts as yet another new RSTP port role as compared to STP. As a reminder, the RSTP alternate port role creates a way for RSTP to quickly replace a switch's root port. Similarly, the RSTP backup port role creates a way for RSTP to quickly replace a switch's designated port on some LAN.

The need for a backup port can be a bit confusing at first, because the need for the backup port role only happens in designs that are a little unlikely today. The reason is that a design must use hubs, which then allows the possibility that one switch connects more than one port to the same collision domain.

Figure 2-10 shows an example. SW3 and SW4 both connect to the same hub. SW4's port F0/1 happens to win the election as designated port (DP). The other port on SW4 that connects to the same collision domain, F0/2, acts as a backup port.

With a backup port, if the current designated port fails, SW4 can start using the backup port with rapid convergence. For instance, if SW4's F0/1 interface were to fail, SW4 could transition F0/2 to the designated port role, without any delay in moving from discarding state to a forwarding state.

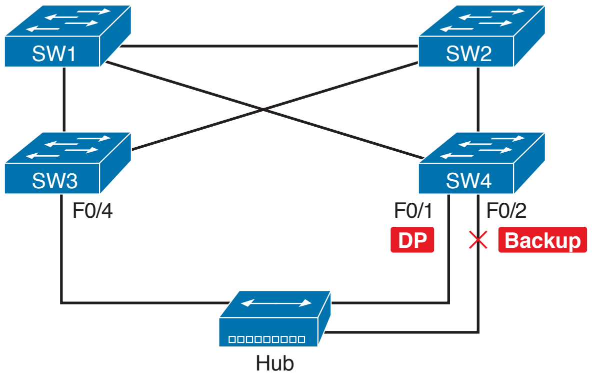


Figure 2-10 *RSTP Backup Port Example*

RSTP Port Types

The final RSTP concept included here relates to some terms RSTP uses to refer to different types of ports and the links that connect to those ports.

To begin, consider the basic figure of Figure 2-11. It shows several links between two switches. RSTP considers these links to be point-to-point links and the ports connected to them to be point-to-point ports, because the link connects exactly two devices (points).

RSTP further classifies point-to-point ports into two categories. Point-to-point ports that connect two switches are not at the edge of the network and are simply called *point-to-point ports*. Ports that instead connect to a single endpoint device at the edge of the network, like a PC or server, are called *point-to-point edge ports*, or simply *edge ports*. In Figure 2-11, SW3's switch port connected to a PC is an edge port.

Finally, RSTP defines the term *shared* to describe ports connected to a hub. The term *shared* comes from the fact that hubs create a shared Ethernet; hubs also force the attached switch port to use half-duplex logic. RSTP assumes that all half-duplex ports may be connected to hubs, treating ports that use half duplex as shared ports. RSTP converges more slowly on shared ports as compared to all point-to-point ports.

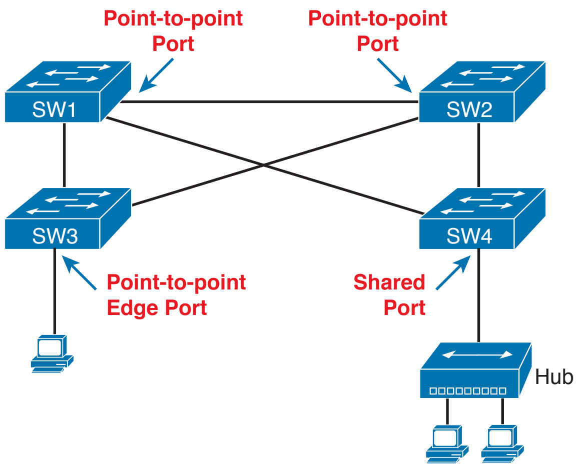


Figure 2-11 *RSTP Link Types*

Optional STP Features

At this point, you have learned plenty of details that will be useful to next configure and verify STP operations, as discussed in Chapter 3. However, before moving to that chapter, the final section of the chapter briefly introduces a few related topics that make STP work even better or be more secure: EtherChannel, PortFast, and BPDU Guard.

EtherChannel

One of the best ways to lower STP's convergence time is to avoid convergence altogether. EtherChannel provides a way to prevent STP convergence from being needed when only a single port or cable failure occurs.

EtherChannel combines multiple parallel segments of equal speed (up to eight) between the same pair of switches, bundled into an EtherChannel. The switches treat the EtherChannel as a single interface with regard to STP. As a result, if one of the links fails, but at least one of the links is up, STP convergence does not have to occur. For example, Figure 2-12 shows the familiar three-switch network, but now with two Gigabit Ethernet connections between each pair of switches.

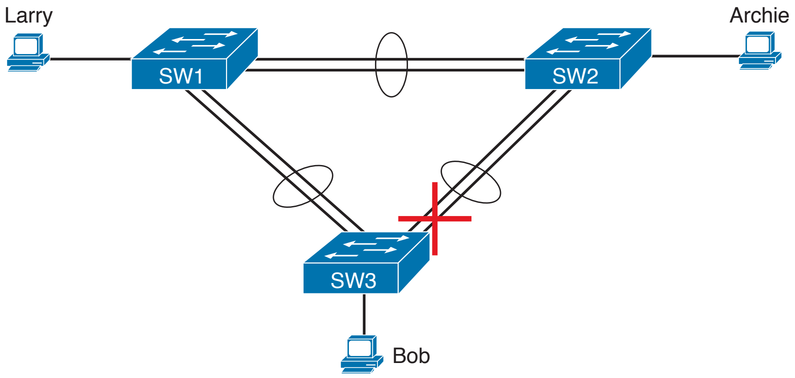


Figure 2-12 *Two-Segment EtherChannels Between Switches*

With each pair of Ethernet links configured as an EtherChannel, STP treats each EtherChannel as a single link. In other words, both links to the same switch must fail for a switch to need to cause STP convergence. Without EtherChannel, if you have multiple parallel links between two switches, STP blocks all the links except one. With EtherChannel, all the parallel links can be up and working at the same time, while reducing the number of times STP must converge, which in turn makes the network more available.

When a switch makes a forwarding decision to send a frame out an EtherChannel, the switch then has to take an extra step in logic: Out which physical interface does it send the frame? The switch has load-balancing logic that lets it pick an interface for each frame, with a goal of spreading the traffic load across all active links in the channel. As a result, a LAN design that uses EtherChannels makes much better use of the available bandwidth between switches, while also reducing the number of times that STP must converge.

Note that EtherChannels may be Layer 2 EtherChannels (as described here) or Layer 3 EtherChannels (as discussed in Chapter 19, "IPv4 Routing in the LAN"). Layer 2 EtherChannels combine links that switches use as switch ports, with the switches using Layer 2 switching logic to forward and receive Ethernet frames over the EtherChannels. Layer 3 EtherChannels also combine links, but the switches use Layer 3 routing logic to forward packets over the EtherChannels. All references to EtherChannel in Part I of this book refer to Layer 2 EtherChannels unless otherwise noted.

PortFast

PortFast allows a switch to immediately transition from blocking to forwarding, bypassing listening and learning states. However, the only ports on which you can safely enable PortFast are ports on which you know that no bridges, switches, or other STP-speaking devices are connected. Otherwise, using PortFast risks creating loops, the very thing that the listening and learning states are intended to avoid.

PortFast is most appropriate for connections to end-user devices. If you turn on PortFast on ports connected to end-user devices, when an end-user PC boots, the switch port can move to an STP forwarding state and forward traffic as soon as the PC NIC is active. Without PortFast, each port must wait while the switch confirms that the port is a DP, and then wait while the interface sits in the temporary listening and learning states before settling into the forwarding state.

PortFast is a popular feature for edge ports; in fact, RSTP incorporates PortFast concepts. You may recall the mention of RSTP port types, particularly point-to-point edge port types, around Figure 2-11. RSTP, by design of the protocol, converges quickly on these point-to-point edge type ports by bypassing the learning state, which is the same idea Cisco originally introduced with PortFast. In practice, Cisco switches enable RSTP point-to-point edge ports by enabling PortFast on the port.

BPDU Guard

STP opens up the LAN to several different types of possible security exposures. For example:

- An attacker could connect a switch to one of these ports, one with a low STP priority value, and become the root switch. The new STP topology could have worse performance than the desired topology.

- The attacker could plug into multiple ports, into multiple switches, become root, and actually forward much of the traffic in the LAN. Without the networking staff realizing it, the attacker could use a LAN analyzer to copy large numbers of data frames sent through the LAN.
- Users could innocently harm the LAN when they buy and connect an inexpensive consumer LAN switch (one that does not use STP). Such a switch, without any STP function, would not choose to block any ports and could cause a loop.

The *Cisco BPDU Guard* feature helps defeat these kinds of problems by disabling a port if any BPDUs are received on the port. So, this feature is particularly useful on ports that should be used only as an access port and never connected to another switch.

In addition, the BPDU Guard feature helps prevent problems with PortFast. PortFast should be enabled only on access ports that connect to user devices, not to other LAN switches. Using BPDU Guard on these same ports makes sense because if another switch connects to such a port, the local switch can disable the port before a loop is created.

Chapter Review

One key to doing well on the exams is to perform repetitive spaced review sessions. Review this chapter's material using either the tools in the book, DVD, or interactive tools for the same material found on the book's companion website. Refer to the "Your Study Plan" element for more details. Table 2-11 outlines the key review elements and where you can find them. To better track your study progress, record when you completed these activities in the second column.

Table 2-11 Chapter Review Tracking

Review Element	Review Date(s)	Resource Used
Review key topics		Book, DVD/website
Review key terms		Book, DVD/website
Answer DIKTA questions		Book, PCPT
Review memory tables		Book, App

Review All the Key Topics

Key Topic

Table 2-12 Key Topics for Chapter 2

Key Topic Element	Description	Page Number
Table 2-2	Lists the three main problems that occur when not using STP in a LAN with redundant links	47
Table 2-3	Lists the reasons why a switch chooses to place an interface into forwarding or blocking state	49
Table 2-4	Lists the most important fields in Hello BPDU messages	50
List	Logic for the root switch election	52
Figure 2-6	Shows how switches calculate their root cost	53

Foundation Topics

Implementing STP

Cisco IOS switches usually use STP (IEEE 802.1D) by default rather than RSTP, and with effective default settings. You can buy some Cisco switches and connect them with Ethernet cables in a redundant topology, and STP will ensure that frames do not loop. And you never even have to think about changing any settings!

Although STP works without any configuration, most medium-size to large-size campus LANs benefit from some STP configuration. With all defaults, the switches choose the root based on the lowest burned-in MAC address on the switches because they all default to use the same STP priority. As a better option, configure the switches so that the root is predictable.

For instance, Figure 3-1 shows a typical LAN design model, with two distribution layer switches (D1 and D2). The design may have dozens of access layer switches that connect to end users; the figure shows just three access switches (A1, A2, and A3). For a variety of reasons, most network engineers make the distribution layer switches be the root. For instance, the configuration could make D1 be the root by having a lower priority, with D2 configured with the next lower priority, so it becomes root if D1 fails.

Key Topic

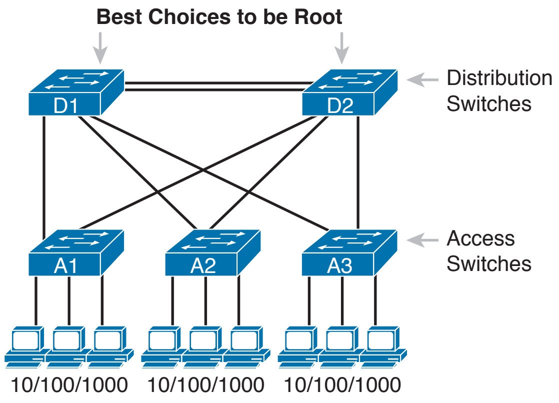


Figure 3-1 *Typical Configuration Choice: Making Distribution Switch Be Root*

This first section of the chapter examines a variety of topics that somehow relate to STP configuration. It begins with a look at STP configuration options, as a way to link the concepts of Chapter 2 to the configuration choices in this chapter. Following that, this section introduces some **show** commands for the purpose of verifying the default STP settings before changing any configuration.

Answers to the "Do I Know This Already?" quiz:

1 B, C **2** B **3** A, D **4** D **5** A, D **6** B, D

Setting the STP Mode

Chapter 2 described how 802.1D STP works in one VLAN. Now that this chapter turns our attention to STP configuration in Cisco switches, one of the first questions is this: Which kind of STP do you intend to use in a LAN? And to answer that question, you need to know a little more background.

The IEEE first standardized STP as the IEEE 802.1D standard, first published back in 1990. To put some perspective on that date, Cisco sold no LAN switches at the time, and virtual LANs did not exist yet. Instead of multiple VLANs in a LAN, there was just one broadcast domain, and one instance of STP. However, the addition of VLANs and the introduction of LAN switches into the market have created a need to add to and extend STP.

Today, Cisco IOS–based LAN switches allow you to use one of three STP configuration modes that reflect that history. The first two sections of this chapter use the mode called Per-VLAN Spanning Tree Plus (PVST+, or sometimes PVSTP), a Cisco-proprietary improvement of 802.1D STP. The *per-VLAN* part of the name gives away the main feature: PVST+ creates a different STP topology per VLAN, whereas 802.1D actually did not. PVST+ also introduced PortFast. Cisco switches often use PVST+ as the default STP mode per a default global command of **spanning-tree mode pvst**.

Over time, Cisco added RSTP support as well, with two STP modes that happen to use RSTP. One mode basically takes PVST+ and upgrades it to use RSTP logic as well, with a mode called *Rapid PVST+*, enabled with the global command **spanning-tree mode rapid-pvst**. Cisco IOS–based switches support a third mode, called Multiple Spanning Tree (MST) (or Multiple Instance of Spanning Tree), enabled with the **spanning-tree mode mst** command. (This book does not discuss MST beyond this brief mention; the CCNP Switch exam typically includes MST details.)

Connecting STP Concepts to STP Configuration Options

If you think back to the details of STP operation in Chapter 2, STP uses two types of numbers for most of its decisions: the BID and STP port costs. Focusing on those two types of numbers, consider this summary of what STP does behind the scenes:

- Uses the BID to elect the root switch, electing the switch with the numerically lowest BID
- Uses the total STP cost in each path to the root, when each nonroot switch chooses its own root port (RP)
- Uses each switch's root cost, which is in turn based on STP port costs, when switches decide which switch port becomes the designated port (DP) on each LAN segment

Unsurprisingly, Cisco switches let you configure part of a switch's BID and the STP port cost, which in turn influences the choices each switch makes with STP.

Per-VLAN Configuration Settings

Beyond supporting the configuration of the BID and STP port costs, Cisco switches support configuring both settings per VLAN. By default, Cisco switches use IEEE 802.1D, not RSTP (802.1w), with a Cisco-proprietary feature called Per-VLAN Spanning Tree Plus (PVST+). PVST+ (often abbreviated as simply PVST today) creates a different instance of STP for each VLAN. So, before looking at the tunable STP parameters, you need to have a basic understanding of PVST+, because the configuration settings can differ for each instance of STP.

PVST+ gives engineers a load-balancing tool with STP. By changing some STP configuration parameters differently for different VLANs, the engineer could cause switches to pick different RPs and DPs in different VLANs. As a result, some traffic in some VLANs can be forwarded over one trunk, and traffic for other VLANs can be forwarded over a different trunk.

Figure 3-2 shows the basic idea, with SW3 forwarding odd-numbered VLAN traffic over the left trunk (Gi0/1) and even-numbered VLANs over the right trunk (Gi0/2).

Key Topic

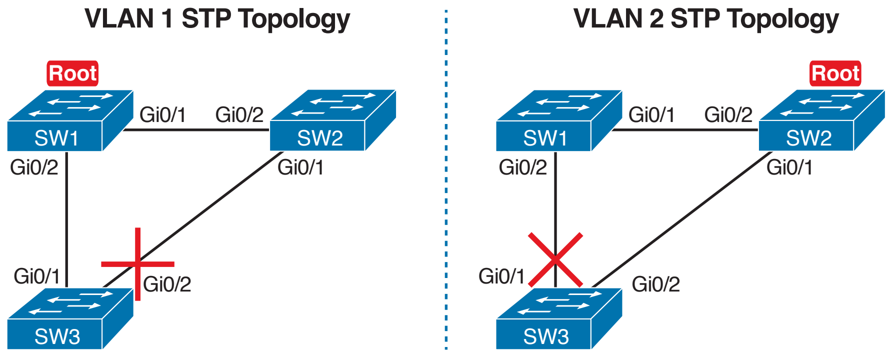


Figure 3-2 *Load Balancing with PVST+*

The next few pages look specifically at how to change the BID and STP port cost settings, per VLAN, when using the default PVST+ mode.

The Bridge ID and System ID Extension

Originally, a switch's BID was formed by combining the switch's 2-byte priority and its 6-byte MAC address. Later, the IEEE changed the rules, splitting the original priority field into two separate fields, as shown in Figure 3-3: a 4-bit priority field and a 12-bit subfield called the *system ID extension* (which represents the VLAN ID).

Key Topic

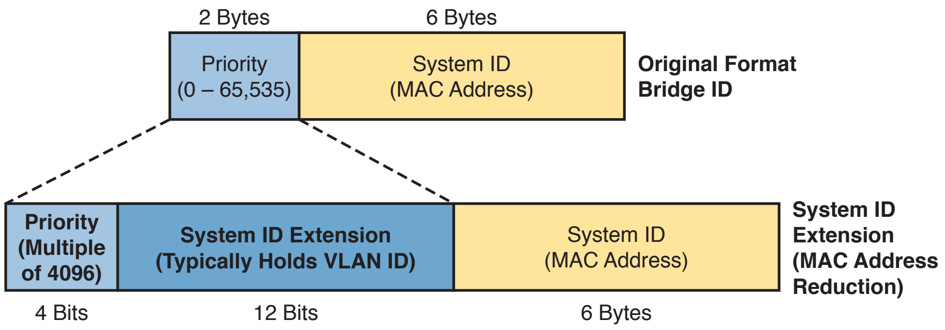

2 Bytes
6 Bytes
Priority (0 – 65,535)
System ID (MAC Address)
Original Format Bridge ID
Priority (Multiple of 4096)
System ID Extension (Typically Holds VLAN ID)
System ID (MAC Address)
System ID Extension (MAC Address Reduction)
4 Bits
12 Bits
6 Bytes

Figure 3-3 *STP System ID Extension*

Cisco switches let you configure the BID, but only the priority part. The switch fills in its universal (burned-in) MAC address as the system ID. It also plugs in the VLAN ID of a

VLAN in the 12-bit system ID extension field. The only part configurable by the network engineer is the 4-bit priority field.

Configuring the number to put in the priority field, however, is one of the strangest things to configure on a Cisco router or switch. As shown at the top of Figure 3-3, the priority field was originally a 16-bit number, which represented a decimal number from 0 to 65,535. Because of that history, the current configuration command (**spanning-tree vlan** *vlan-id* **priority** *x*) requires a decimal number between 0 and 65,535. But not just any number in that range will suffice—it must be a multiple of 4096: 0, 4096, 8192, 12288, and so on, up through 61,440.

The switch still sets the first 4 bits of the BID based on the configured value. As it turns out, of the 16 allowed multiples of 4096, from 0 through 61,440, each has a different binary value in their first 4 bits: 0000, 0001, 0010, and so on, up through 1111. The switch sets the true 4-bit priority based on the first 4 bits of the configured value.

Although the history and configuration might make the BID priority idea seem a bit convoluted, having an extra 12-bit field in the BID works well in practice because it can be used to identify the VLAN ID. VLAN IDs range from 1 to 4094, requiring 12 bits. Cisco switches place the VLAN ID into the system ID extension field, so each switch has a unique BID per VLAN.

For example, a switch configured with VLANs 1 through 4, with a default base priority of 32,768, has a default STP priority of 32,769 in VLAN 1, 32,770 in VLAN 2, 32,771 in VLAN 3, and so on. So, you can view the 16-bit priority as a base priority (as configured in the **spanning-tree vlan** *vlan-id* **priority** *x* command) plus the VLAN ID.

NOTE Cisco switches must use the system ID extension version of the bridge ID; it cannot be disabled.

Per-VLAN Port Costs

Each switch interface defaults its per-VLAN STP cost based on the IEEE recommendations listed in Table 2-6 in Chapter 2. On interfaces that support multiple speeds, Cisco switches base the cost on the current actual speed. So, if an interface negotiates to use a lower speed, the default STP cost reflects that lower speed. If the interface negotiates to use a different speed, the switch dynamically changes the STP port cost as well.

Alternatively, you can configure a switch's STP port cost with the **spanning-tree** [**vlan** *vlan-id*] **cost** *cost* interface subcommand. You see this command most often on trunks because setting the cost on trunks has an impact on the switch's root cost, whereas setting STP costs on access ports does not.

For the command itself, it can include the VLAN ID, or not. The command only needs a **vlan** parameter on trunk ports to set the cost per VLAN. On a trunk, if the command omits the VLAN parameter, it sets the STP cost for all VLANs whose cost is not set by a **spanning-tree vlan** *x* **cost** command for that VLAN.

STP Configuration Option Summary

Table 3-2 summarizes the default settings for both the BID and the port costs and lists the optional configuration commands covered in this chapter.

Key Topic

Table 3-2 STP Defaults and Configuration Options

Setting	Default	Command(s) to Change Default
BID priority	Base: 32,768	**spanning-tree vlan** *vlan-id* **root** {**primary** | **secondary**} **spanning-tree vlan** *vlan-id* **priority** *priority*
Interface cost	100 for 10 Mbps 19 for 100 Mbps 4 for 1 Gbps 2 for 10 Gbps	**spanning-tree vlan** *vlan-id* **cost** *cost*
PortFast	Not enabled	**spanning-tree portfast**
BPDU Guard	Not enabled	**spanning-tree bpduguard enable**

Next, the configuration section shows how to examine the operation of STP in a simple network, along with how to change these optional settings.

Verifying STP Operation

Before taking a look at how to change the configuration, first consider a few STP verification commands. Looking at these commands first will help reinforce the default STP settings. In particular, the examples in this section use the network shown in Figure 3-4.

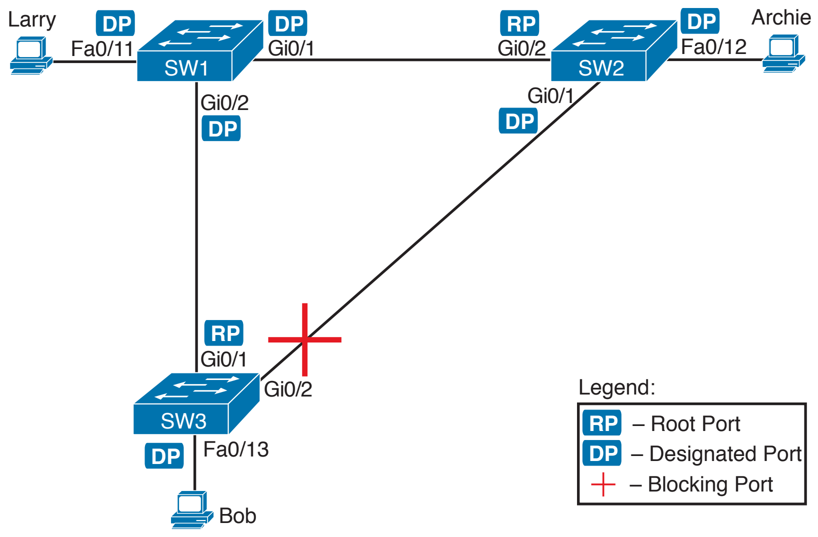


Figure 3-4 *Sample LAN for STP Configuration and Verification Examples*

Example 3-1 begins the discussion with a useful command for STP: the **show spanning-tree vlan 10** command. This command identifies the root switch and lists settings on the local switch. Example 3-1 lists the output of this command on both SW1 and SW2, as explained following the example.

Example 3-1 *STP Status with Default STP Parameters on SW1 and SW2*

```
SW1# show spanning-tree vlan 10

VLAN0010
  Spanning tree enabled protocol ieee
  Root ID    Priority    32778
             Address     1833.9d7b.0e80
             This bridge is the root
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    32778  (priority 32768 sys-id-ext 10)
             Address     1833.9d7b.0e80
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  300 sec

Interface           Role Sts Cost      Prio.Nbr Type
------------------- ---- --- --------- -------- --------------------------------
Fa0/11              Desg FWD 19        128.11   P2p Edge
Gi0/1               Desg FWD 4         128.25   P2p
Gi0/2               Desg FWD 4         128.26   P2p
```

```
SW2# show spanning-tree vlan 10

VLAN0010
  Spanning tree enabled protocol ieee
  Root ID    Priority    32778
             Address     1833.9d7b.0e80
             Cost        4
             Port        26 (GigabitEthernet0/2)
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    32778  (priority 32768 sys-id-ext 10)
             Address     1833.9d7b.1380
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  300 sec

Interface           Role Sts Cost      Prio.Nbr Type
------------------- ---- --- --------- -------- --------------------------------
Fa0/12              Desg FWD 19        128.12   P2p
Gi0/1               Desg FWD 4         128.25   P2p
Gi0/2               Root FWD 4         128.26   P2p
```

Example 3-1 begins with the output of the **show spanning-tree vlan 10** command on SW1. This command first lists three major groups of messages: one group of messages about the root switch, followed by another group about the local switch, and ending with interface role and status information. In this case, SW1 lists its own BID as the root, with even a specific statement that "This bridge is the root," confirming that SW1 is now the root of the VLAN 10 STP topology.

Next, compare the highlighted lines of the same command on SW2 in the lower half of the example. SW2 lists SW1's BID details as the root; in other words, SW2 agrees that SW1 has won the root election. SW2 does not list the phrase "This bridge is the root." SW2 then lists its own (different) BID details in the lines after the details about the root's BID.

The output also confirms a few default values. First, each switch lists the priority part of the BID as a separate number: 32778. This value comes from the default priority of 32768, plus VLAN 10, for a total of 32778. The output also shows the interface cost for some Fast Ethernet and Gigabit Ethernet interfaces, defaulting to 19 and 4, respectively.

Finally, the bottom of the output from the **show spanning-tree** command lists each interface in the VLAN, including trunks, with the STP port role and port state listed. For instance, on switch SW1, the output lists three interfaces, with a role of Desg for designated port (DP) and a state of FWD for forwarding. SW2 lists three interfaces, two DPs, and one root port, so all three are in an FWD or forwarding state.

Example 3-1 shows a lot of good STP information, but two other commands, shown in Example 3-2, work better for listing BID information in a shorter form. The first, **show spanning-tree root**, lists the root's BID for each VLAN. This command also lists other details, like the local switch's root cost and root port. The other command, **show spanning-tree vlan 10 bridge**, breaks out the BID into its component parts. In this example, it shows SW2's priority as the default of 32768, the VLAN ID of 10, and the MAC address.

Example 3-2 *Listing Root Switch and Local Switch BIDs on Switch SW2*

```
SW2# show spanning-tree root

                                        Root     Hello Max Fwd
Vlan                     Root ID        Cost     Time  Age Dly  Root Port
---------------- -------------------- --------- ----- --- ---  ------------
VLAN0001         32769 1833.9d5d.c900        23     2  20  15  Gi0/1
VLAN0010         32778 1833.9d7b.0e80         4     2  20  15  Gi0/2
VLAN0020         32788 1833.9d7b.0e80         4     2  20  15  Gi0/2
VLAN0030         32798 1833.9d7b.0e80         4     2  20  15  Gi0/2
VLAN0040         32808 1833.9d7b.0e80         4     2  20  15  Gi0/2

SW2# show spanning-tree vlan 10 bridge

                                                   Hello  Max  Fwd
Vlan                         Bridge ID              Time  Age  Dly  Protocol
---------------- --------------------------------- -----  ---  ---  --------
VLAN0010         32778 (32768,  10) 1833.9d7b.1380    2    20   15  ieee
```

Note that both the commands in Example 3-2 have a VLAN option: **show spanning-tree** [**vlan** *x*] **root** and **show spanning-tree** [**vlan** *x*] **bridge**. Without the VLAN listed, each command lists one line per VLAN; with the VLAN, the output lists the same information, but just for that one VLAN.

Configuring STP Port Costs

Changing the STP port costs requires a simple interface subcommand: **spanning-tree** [**vlan** *x*] **cost** *x*. To show how it works, consider the following example, which changes what happens in the network shown in Figure 3-4.

Back in Figure 3-4, with default settings, SW1 became root, and SW3 blocked on its G0/2 interface. A brief scan of the figure, based on the default STP cost of 4 for Gigabit interfaces, shows that SW3 should have found a cost 4 path and a cost 8 path to reach the root, as shown in Figure 3-5.

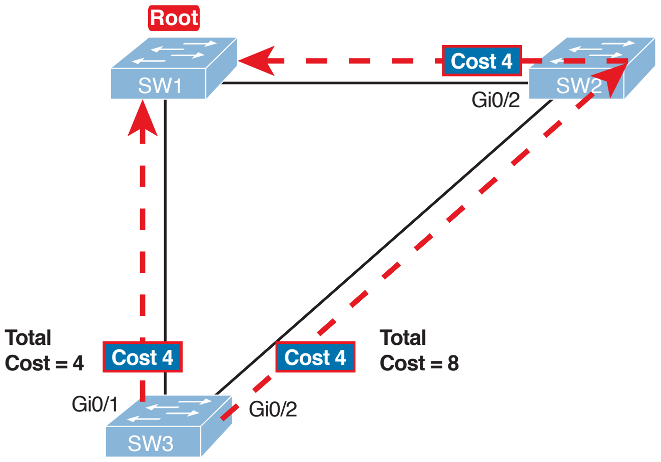


Figure 3-5 *Analysis of SW3's Current Root Cost of 4 with Defaults*

To show the effects of changing the port cost, the next example shows a change to SW3's configuration, setting its G0/1 port cost higher so that the better path to the root goes out SW3's G0/2 port instead. Example 3-3 also shows several other interesting effects.

Example 3-3 *Manipulating STP Port Cost and Watching the Transition to Forwarding State*

```
SW3# debug spanning-tree events
Spanning Tree event debugging is on
SW3# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
SW3(config)# interface gigabitethernet0/1
SW3(config-if)# spanning-tree vlan 10 cost 30
SW3(config-if)# ^Z
SW3#
*Mar 11 06:28:00.860: STP: VLAN0010 new root port Gi0/2, cost 8
*Mar 11 06:28:00.860: STP: VLAN0010 Gi0/2 -> listening
*Mar 11 06:28:00.860: STP: VLAN0010 sent Topology Change Notice on Gi0/2
*Mar 11 06:28:00.860: STP[10]: Generating TC trap for port GigabitEthernet0/1
*Mar 11 06:28:00.860: STP: VLAN0010 Gi0/1 -> blocking
*Mar 11 06:28:15.867: STP: VLAN0010 Gi0/2 -> learning
*Mar 11 06:28:30.874: STP[10]: Generating TC trap for port GigabitEthernet0/2
*Mar 11 06:28:30.874: STP: VLAN0010 sent Topology Change Notice on Gi0/2
*Mar 11 06:28:30.874: STP: VLAN0010 Gi0/2 -> forwarding
```

This example starts with the **debug spanning-tree events** command on SW3. This command tells the switch to issue debug log messages whenever STP performs changes to an interface's role or state. These messages show up in the example as a result of the configuration.

Next, the example shows the configuration to change SW3's port cost, in VLAN 10, to 30, with the **spanning-tree vlan 10 cost 30** interface subcommand. Based on the figure, the root cost through SW3's G0/1 will now be 30 instead of 4. As a result, SW3's best cost to reach the root is cost 8, with SW3's G0/2 as its root port.

The debug messages tell us what STP on SW3 is thinking behind the scenes, with timestamps. Note that the first five debug messages, displayed immediately after the user exited configuration mode in this case, all happen at the same time (down to the same millisecond). Notably, G0/1, which had been forwarding, immediately moves to a blocking state. Interface G0/2, which had been blocking, does not go to a forwarding state, instead moving to a listening state (at least, according to this message).

Now look for the debug message that lists G0/2 transitioning to learning state, and then the next one that shows it finally reaching forwarding state. How long between the messages? In each case, the message's timestamps show that 15 seconds passed. In this experiment, the switches used a default setting of forward delay (15 seconds). So, these debug messages confirm the steps that STP takes to transition an interface from blocking to forwarding state.

If you did not happen to enable a debug when configuring the cost, using **show** commands later can confirm the same choice by SW3, to now use its G0/2 port as its RP. Example 3-4 shows the new STP port cost setting on SW3, along with the new root port and root cost, using the **show spanning-tree vlan 10** command. Note that G0/2 is now listed as the root port. The top of the output lists SW3's root cost as 8, matching the analysis shown in Figure 3-5.

Example 3-4 *New STP Status and Settings on SW3*

```
SW3# show spanning-tree vlan 10

VLAN0010
  Spanning tree enabled protocol ieee
  Root ID    Priority    32778
             Address     1833.9d7b.0e80
             Cost        8
             Port        26 (GigabitEthernet0/2)
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    32778  (priority 32768 sys-id-ext 10)
             Address     f47f.35cb.d780
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  300 sec

Interface           Role Sts Cost      Prio.Nbr Type
------------------- ---- --- --------- -------- --------------------------------
Fa0/23              Desg FWD 19        128.23   P2p
Gi0/1               Altn BLK 30        128.25   P2p
Gi0/2               Root FWD 4         128.26   P2p
```

Configuring Priority to Influence the Root Election

The other big STP configuration option is to influence the root election by changing the priority of a switch. The priority can be set explicitly with the **spanning-tree vlan** *vlan-id* **priority** *value* global configuration command, which sets the base priority of the switch. (This is the command that requires a parameter of a multiple of 4096.)

However, Cisco gives us a better configuration option than configuring a specific priority value. In most designs, the network engineers pick two switches to be root: one to be root if all switches are up, and another to take over if the first switch fails. Switch IOS supports this idea with the **spanning-tree vlan** *vlan-id* **root primary** and **spanning-tree vlan** *vlan-id* **root secondary** commands.

The **spanning-tree vlan** *vlan-id* **root primary** command tells the switch to set its priority low enough to become root right now. The switch looks at the current root in that VLAN, and at the root's priority. Then the local switch chooses a priority value that causes the local switch to take over as root.

Remembering that Cisco switches use a default base priority of 32,768, this command chooses the base priority as follows:

Key Topic

- If the current root has a base priority higher than 24,576, the local switch uses a base priority of 24,576.
- If the current root's base priority is 24,576 or lower, the local switch sets its base priority to the highest multiple of 4096 that still results in the local switch becoming root.

For the switch intended to take over as the root if the first switch fails, use the **spanning-tree vlan** *vlan-id* **root secondary** command. This command is much like the **spanning-tree vlan** *vlan-id* **root primary** command, but with a priority value worse than the primary switch but better than all the other switches. This command sets the switch's base priority to 28,672 regardless of the current root's current priority value.

For example, in Figures 3-4 and 3-5, SW1 was the root switch, and as shown in various commands, all three switches defaulted to use a base priority of 32,768. Example 3-5 shows a configuration that makes SW2 the primary root, and SW1 the secondary, just to show the role move from one to the other. These commands result in SW2 having a base priority of 24,576, and SW1 having a base priority of 28,672.

Example 3-5 *Making SW2 Become Root Primary, and SW1 Root Secondary*

```
! First, on SW2:
SW2# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
SW2(config)# spanning-tree vlan 10 root primary
SW2(config)# ^Z
```

```
! Next, SW1 is configured to back-up SW1
SW1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)# spanning-tree vlan 10 root secondary
SW1(config)# ^Z
SW1#
```

```
! The next command shows the local switch's BID (SW1)
SW1# show spanning-tree vlan 10 bridge

                                                   Hello  Max  Fwd
Vlan                         Bridge ID              Time  Age  Dly  Protocol
---------------- --------------------------------- -----  ---  ---  --------
VLAN0010         28682 (28672,  10) 1833.9d7b.0e80    2    20   15  ieee

! The next command shows the root's BID (SW2)
SW1# show spanning-tree vlan 10 root

                                        Root    Hello Max Fwd
Vlan                   Root ID          Cost    Time  Age Dly  Root Port
---------------- -------------------- --------- ----- --- ---  ------------
VLAN0010         24586 1833.9d7b.1380         4     2   20  15  Gi0/1
```

The output of the two **show** commands clearly points out the resulting priority values on each switch. First, the **show spanning-tree bridge** command lists the local switch's BID information, while the **show spanning-tree root** command lists the root's BID, plus the local switch's root cost and root port (assuming it is not the root switch). So, SW1 lists its own BID, with priority 28,682 (base 28,672, with VLAN 10) with the **show spanning-tree bridge** command. Still on SW1, the output lists the root's priority as 24,586 in VLAN 10, implied as base 24,576 plus 10 for VLAN 10, with the **show spanning-tree root** command.

Note that alternatively you could have configured the priority settings specifically. SW1 could have used the **spanning-tree vlan 10 priority 28672** command, with SW2 using the **spanning-tree vlan 10 priority 24576** command. In this particular case, both options would result in the same STP operation.

Implementing Optional STP Features

This just-completed first major section of the chapter showed examples that used PVST+ only, assuming a default global command of **spanning-tree mode pvst**. At the same time, all the configuration commands shown in that first section, commands that influence STP operation, would influence both traditional STP and RSTP operation.

This section, the second of three major sections in this chapter, now moves on to discuss some useful but optional features that make both STP and RSTP work even better.

Configuring PortFast and BPDU Guard

You can easily configure the PortFast and BPDU Guard features on any interface, but with two different configuration options. One option works best when you want to enable these features only on a few ports, and the other works best when you want to enable these features on most every access port.

First, to enable the features on just one port at a time, use the **spanning-tree portfast** and the **spanning-tree bpduguard enable** interface subcommands. Example 3-6 shows an

```
Configured Pathcost method used is short

Name                   Blocking Listening Learning Forwarding STP Active
---------------------- -------- --------- -------- ---------- ----------
VLAN0001                      3         0        0          2          5
---------------------- -------- --------- -------- ---------- ----------
1 vlan                        3         0        0          2          5
```

Configuring EtherChannel

As introduced back in Chapter 2, two neighboring switches can treat multiple parallel links between each other as a single logical link called an *EtherChannel*. STP operates on the EtherChannel, instead of the individual physical links, so that STP either forwards or blocks on the entire logical EtherChannel for a given VLAN. As a result, a switch in a forwarding state can then load balance traffic over all the physical links in the EtherChannel. Without EtherChannel, only one of the parallel links between two switches would be allowed to forward traffic, with the rest of the links blocked by STP.

NOTE All references to EtherChannel in this Chapter refer to Layer 2 EtherChannels, and not to Layer 3 EtherChannels (as discussed in Chapter 19, "IPv4 Routing in the LAN").

EtherChannel may be one of the most challenging switch features to make work. First, the configuration has several options, so you have to remember the details of which options work together. Second, the switches also require a variety of other interface settings to match among all the links in the channel, so you have to know those settings as well.

This section focuses on the correct EtherChannel configuration. Chapter 4's section "Troubleshooting Layer 2 EtherChannel" looks at many of the potential problems with EtherChannel, including all those other configuration settings that a switch checks before allowing the EtherChannel to work.

Configuring a Manual EtherChannel

The simplest way to configure an EtherChannel is to add the correct **channel-group** configuration command to each physical interface, on each switch, all with the **on** keyword. The **on** keyword tells the switches to place a physical interface into an EtherChannel.

Before getting into the configuration and verification, however, you need to start using three terms as synonyms: *EtherChannel*, *PortChannel*, and *Channel-group*. Oddly, IOS uses the **channel-group** configuration command, but then to display its status, IOS uses the **show etherchannel** command. Then, the output of this **show** command refers to neither an "EtherChannel" nor a "Channel-group," instead using the term "PortChannel." So, pay close attention to these three terms in the example.

To configure an EtherChannel manually, follow these steps:

Key Topic

Step 1. Add the **channel-group** *number* **mode on** command in interface configuration mode under each physical interface that should be in the channel to add it to the channel.

Step 2. Use the same number for all commands on the same switch, but the channel-group number on the neighboring switch can differ.

Example 3-9 shows a simple example, with two links between switches SW1 and SW2, as shown in Figure 3-6. The configuration shows SW1's two interfaces placed into channel-group 1, with two **show** commands to follow.

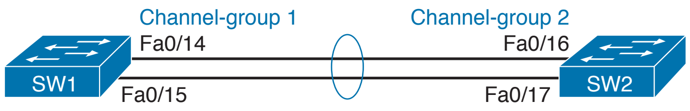


Figure 3-6 *Sample LAN Used in EtherChannel Example*

Example 3-9 *Configuring and Monitoring EtherChannel*

```
SW1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)# interface fa 0/14
SW1(config-if)# channel-group 1 mode on
SW1(config)# interface fa 0/15
SW1(config-if)# channel-group 1 mode on
SW1(config-if)# ^Z

SW1# show spanning-tree vlan 3

VLAN0003
  Spanning tree enabled protocol ieee
  Root ID     Priority     28675
              Address      0019.e859.5380
              Cost         12
              Port         72  (Port-channel1)
              Hello Time    2 sec   Max Age 20 sec   Forward Delay 15 sec

  Bridge ID   Priority     28675   (priority 28672 sys-id-ext 3)
              Address      0019.e86a.6f80
              Hello Time    2 sec   Max Age 20 sec   Forward Delay 15 sec
              Aging Time 300

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------
Po1              Root FWD 12        128.64   P2p Peer(STP)

SW1# show etherchannel 1 summary
Flags:  D - down        P - bundled in port-channel
        I - stand-alone s - suspended
        H - Hot-standby (LACP only)
        R - Layer3      S - Layer2
        U - in use      f - failed to allocate aggregator
        M - not in use, minimum links not met
        u - unsuitable for bundling
        w - waiting to be aggregated
        d - default port


Number of channel-groups in use: 1
Number of aggregators:           1

Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------------------------------
1      Po1(SU)          -        Fa0/14(P)   Fa0/15(P)
```

Take a few moments to look at the output in the two **show** commands in the example, as well. First, the **show spanning-tree** command lists Po1, short for PortChannel1, as an interface. This interface exists because of the **channel-group** commands using the **1** parameter. STP no longer operates on physical interfaces F0/14 and F0/15, instead operating on the PortChannel1 interface, so only that interface is listed in the output.

Next, note the output of the **show etherchannel 1 summary** command. It lists as a heading "Port-channel," with Po1 below it. It also lists both F0/14 and F0/15 in the list of ports, with a (P) beside each. Per the legend, the *P* means that the ports are bundled in the port channel, which is a code that means these ports have passed all the configuration checks and are valid to be included in the channel.

NOTE Cisco uses the term *EtherChannel* to refer to the concepts discussed in this section. To refer to the item configured in the switch, Cisco instead uses the term *port channel*, with the command keyword **port-channel**. For the purposes of understanding the technology, you may treat these terms as synonyms. However, it helps to pay close attention to the use of the terms *port channel* and *EtherChannel* as you work through the examples in this section, because IOS uses both.

Configuring Dynamic EtherChannels

Cisco switches support two different protocols that allow the switches to negotiate whether a particular link becomes part of an EtherChannel or not. Basically, the configuration enables the protocol for a particular channel-group number. At that point, the switch can use the protocol to send messages to/from the neighboring switch and discover whether their configuration settings pass all checks. If a given physical link passes, the link is added to the EtherChannel and used; if not, it is placed in a down state, and not used, until the configuration inconsistency can be resolved.

Cisco switches support the Cisco-proprietary Port Aggregation Protocol (PAgP) and the IEEE standard Link Aggregation Control Protocol (LACP), based on IEEE standard 802.3ad. Although differences exist between the two, to the depth discussed here, they both accomplish the same task: negotiate so that only links that pass the configuration checks are actually used in an EtherChannel.

To configure either protocol, a switch uses the **channel-group** configuration commands on each switch, but with a keyword that either means "use this protocol and begin negotiations" or "use this protocol and wait for the other switch to begin negotiations." As shown in Figure 3-7, the **desirable** and **auto** keywords enable PAgP, and the **active** and **passive** keywords enable LACP. With these options, at least one side has to begin the negotiations. In other words, with PAgP, at least one of the two sides must use **desirable**, and with LACP, at least one of the two sides must use **active**.

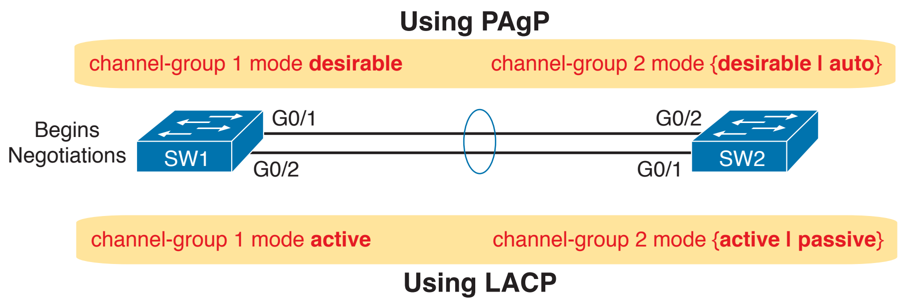

Using PAgP
channel-group 1 mode desirable
channel-group 2 mode {desirable | auto}
Begins Negotiations
SW1
G0/1
G0/2
G0/2
G0/1
SW2
channel-group 1 mode active
channel-group 2 mode {active | passive}
Using LACP

Figure 3-7 *Correct EtherChannel Configuration Combinations*

NOTE Do not use the **on** parameter on one end, and either **auto** or **desirable** (or for LACP, **active** or **passive**) on the neighboring switch. The **on** option uses neither PAgP nor LACP, so a configuration that uses **on**, with PAgP or LACP options on the other end, would prevent the EtherChannel from working.

For example, in the design shown in Figure 3-7, imagine both physical interfaces on both switches were configured with the **channel-group 2 mode desirable** interface subcommand. As a result, the two switches would negotiate and create an EtherChannel. Example 3-10 shows the verification of that configuration, with the command **show etherchannel 2 port-channel**. This command confirms the protocol in use (PAgP, because the **desirable** keyword was configured), and the list of interfaces in the channel.

Example 3-10 *EtherChannel Verification: PAgP Desirable Mode*

```
SW1# show etherchannel 2 port-channel
                Port-channels in the group:
                ---------------------------


Port-channel: Po2
------------


Age of the Port-channel   = 0d:00h:04m:04s
Logical slot/port   = 16/1          Number of ports = 2
GC                  = 0x00020001      HotStandBy port = null
Port state          = Port-channel Ag-Inuse
Protocol            =   PAgP
Port security       = Disabled
```

```
Ports in the Port-channel:

Index   Load   Port     EC state        No of bits
------+------+------+------------------+-----------
  0     00     Gi0/1    Desirable-Sl       0
  0     00     Gi0/2    Desirable-Sl       0

Time since last port bundled:    0d:00h:03m:57s    Gi0/2
```

Implementing RSTP

All you have to do to migrate from STP to RSTP is to configure the **spanning-tree mode rapid-pvst** global command on all the switches. However, for exam preparation, it helps to work through the various **show** commands, particularly to prepare for Simlet questions. Those questions can ask you to interpret **show** command output without allowing you to look at the configuration, and the output of **show** commands when using STP versus RSTP is very similar.

This third and final major section of this chapter focuses on pointing out the similarities and differences between STP and RSTP as seen in Catalyst switch configuration and verification commands. This section explains the configuration and verification of RSTP, with emphasis on how to identify RSTP features.

Identifying the STP Mode on a Catalyst Switch

Cisco Catalyst switches operate in some STP mode as defined by the **spanning-tree mode** global configuration command. Based on this command's setting, the switch is using either 802.1D STP or 802.1w RSTP, as noted in Table 3-4.

Key Topic

Table 3-4 Cisco Catalyst STP Configuration Modes

Parameter on spanning-tree mode Command	Uses STP or RSTP?	Protocol Listed in Command Output	Description
pvst	STP	ieee	Default; Per-VLAN Spanning Tree instance
rapid-pvst	RSTP	rstp	Like PVST, but uses RSTP rules instead of STP for each STP instance
mst	RSTP	mst	Creates multiple RSTP instances but does not require one instance per each VLAN

To determine whether a Cisco Catalyst switch uses RSTP, you can look for two types of information. First, you can look at the configuration, as noted in the left column of Table 3-4. Also, some **show** commands list the STP protocol as a reference to the configuration of the **spanning-tree mode** global configuration command. A protocol of **rstp** or **mst** refers to one of the modes that uses RSTP, and a protocol of **ieee** refers to the mode that happens to use STP.

Before looking at an example of the output, review the topology in Figure 3-8. The remaining RSTP examples in this chapter use this topology. In the RSTP examples in this chapter, SW1 will become root, and SW3 will block on one port (G0/2), as shown.

When STP converges based on some change, not all the ports have to change their state. For instance, a port that was forwarding, if it still needs to forward, just keeps on forwarding. Ports that were blocking that still need to block keep on blocking. But when a port needs to change state, something has to happen, based on the following rules:

Key Topic

- For interfaces that stay in the same STP state, nothing needs to change.
- For interfaces that need to move from a forwarding state to a blocking state, the switch immediately changes the state to blocking.
- For interfaces that need to move from a blocking state to a forwarding state, the switch first moves the interface to listening state, then learning state, each for the time specified by the forward delay timer (default 15 seconds). Only then is the interface placed into forwarding state.

Because the transition from blocking to forwarding does require some extra steps, you should be ready to respond to conceptual questions about the transition. To be ready, review the section "Reacting to State Changes That Affect the STP Topology" in Chapter 2.

Troubleshooting Layer 2 EtherChannel

EtherChannels can prove particularly challenging to troubleshoot for a couple of reasons. First, you have to be careful to match the correct configuration, and there are many more incorrect configuration combinations than there are correct combinations. Second, many interface settings must match on the physical links, both on the local switch and on the neighboring switch, before a switch will add the physical link to the channel. This second major section in the chapter works through both sets of issues.

Incorrect Options on the channel-group Command

In Chapter 3, the section titled "Configuring EtherChannel" listed the small set of working configuration options on the **channel-group** command. Those rules can be summarized as follows, for a single EtherChannel:

Key Topic

1. On the local switch, all the **channel-group** commands for all the physical interfaces must use the same channel-group number.
2. The channel-group number can be different on the neighboring switches.
3. If using the **on** keyword, you must use it on the corresponding interfaces of both switches.
4. If you use the **desirable** keyword on one switch, the switch uses PAgP; the other switch must use either **desirable** or **auto**.
5. If you use the **active** keyword on one switch, the switch uses LACP; the other switch must use either **active** or **passive**.

These rules summarize the correct configuration options, but the options actually leave many more incorrect choices. The following list shows some incorrect configurations that the switches allow, even though they would result in the EtherChannel not working. The list compares the configuration on one switch to another based on the physical interface configuration. Each lists the reasons why the configuration is incorrect.

- Configuring the **on** keyword on one switch, and **desirable**, **auto**, **active**, or **passive** on the other switch. The **on** keyword does not enable PAgP, and does not enable LACP, and the other options rely on PAgP or LACP.

- Configuring the **auto** keyword on both switches. Both use PAgP, but both wait on the other switch to begin negotiations.
- Configuring the **passive** keyword on both switches. Both use LACP, but both wait on the other switch to begin negotiations.
- Configuring the **active** keyword on one switch and either **desirable** or **auto** on the other switch. The **active** keyword uses LACP, whereas the other keywords use PAgP.
- Configuring the **desirable** keyword on one switch and either **active** or **passive** on the other switch. The **desirable** keyword uses PAgP, whereas the other keywords use LACP.

Example 4-2 shows an example that matches the last item in the list. In this case, SW1's two ports (F0/14 and F0/15) have been configured with the **desirable** keyword, and SW2's matching F0/16 and F0/17 have been configured with the **active** keyword. The example lists some telling status information about the failure, with notes following the example.

Example 4-2 *Incorrect Configuration Using Mismatched PortChannel Protocols*

```
SW1# show etherchannel summary
Flags:  D - down        P - bundled in port-channel
        I - stand-alone s - suspended
        H - Hot-standby (LACP only)
        R - Layer3      S - Layer2
        U - in use      f - failed to allocate aggregator

        M - not in use, minimum links not met
        u - unsuitable for bundling
        w - waiting to be aggregated
        d - default port


Number of channel-groups in use: 1
Number of aggregators:           1

Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------------------------------
1      Po1(SD)         PAgP      Fa0/14(I)   Fa0/15(I)

SW1# show interfaces status | include Po|14|15
Port      Name               Status       Vlan       Duplex  Speed Type
Fa0/14                       connected    301        a-full  a-100 10/100BaseTX
Fa0/15                       connected    301        a-full  a-100 10/100BaseTX
Po1                          notconnect   unassigned   auto    auto
```

Start at the top, in the legend of the **show etherchannel summary** command. The *D* code letter means that the channel itself is down, with *S* meaning that the channel is a Layer 2 EtherChannel. Code *I* means that the physical interface is working independently from the PortChannel (described as "stand-alone"). Then, the bottom of that command's output highlights PortChannel 1 (Po1) as Layer 2 EtherChannel in a down state (SD), with F0/14 and F0/15 as stand-alone interfaces (I).

Interestingly, because the problem is a configuration mistake, the two physical interfaces still operate independently, as if the PortChannel did not exist. The last command in the example shows that while the PortChannel 1 interface is down, the two physical interfaces are in a connected state.

NOTE As a suggestion for attacking EtherChannel problems on the exam, rather than memorizing all the incorrect configuration options, concentrate on the list of correct configuration options. Then look for any differences between a given question's configuration as compared to the known correct configurations and work from there.

Configuration Checks Before Adding Interfaces to EtherChannels

Even when the **channel-group** commands have all been configured correctly, other configuration settings can cause problems as well. This last topic examines those configuration settings and their impact.

First, a local switch checks each new physical interface that is configured to be part of an EtherChannel, comparing each new link to the existing links. That new physical interface's settings must be the same as the existing links' settings; otherwise, the switch does not add the new link to the list of approved and working interfaces in the channel. That is, the physical interface remains configured as part of the PortChannel, but it is not used as part of the channel, often being placed into some nonworking state.

The list of items the switch checks includes the following:

Key Topic

- Speed
- Duplex
- Operational access or trunking state (all must be access, or all must be trunks)
- If an access port, the access VLAN
- If a trunk port, the allowed VLAN list (per the **switchport trunk allowed** command)
- If a trunk port, the native VLAN
- STP interface settings

In addition, switches check the settings on the neighboring switch. To do so, the switches either use PAgP or LACP (if already in use), or use Cisco Discovery Protocol (CDP) if using manual configuration. The neighbor must match on all parameters in this list except the STP settings.

As an example, SW1 and SW2 again use two links in one EtherChannel. Before configuring the EtherChannel, SW1's F0/15 was given a different STP port cost than F0/14. Example 4-3 picks up the story just after configuring the correct **channel-group** commands, when the switch is deciding whether to use F0/14 and F0/15 in this EtherChannel.

Example 4-3 *Local Interfaces Fail in EtherChannel Because of Mismatched STP Cost*

```
*Mar  1 23:18:56.132: %PM-4-ERR_DISABLE: channel-misconfig (STP) error detected on
  Po1, putting Fa0/14 in err-disable state
*Mar  1 23:18:56.132: %PM-4-ERR_DISABLE: channel-misconfig (STP) error detected on
  Po1, putting Fa0/15 in err-disable state
```

```
*Mar  1 23:18:56.132: %PM-4-ERR_DISABLE: channel-misconfig (STP) error detected on
  Po1, putting Po1 in err-disable state
*Mar  1 23:18:58.120: %LINK-3-UPDOWN: Interface FastEthernet0/14, changed state to
  down
*Mar  1 23:18:58.137: %LINK-3-UPDOWN: Interface Port-channel1, changed state to down
*Mar  1 23:18:58.137: %LINK-3-UPDOWN: Interface FastEthernet0/15, changed state to
  down

SW1# show etherchannel summary
Flags:  D - down        P - bundled in port-channel
        I - stand-alone s - suspended
        H - Hot-standby (LACP only)
        R - Layer3      S - Layer2
        U - in use      f - failed to allocate aggregator

        M - not in use, minimum links not met
        u - unsuitable for bundling
        w - waiting to be aggregated
        d - default port


Number of channel-groups in use: 1
Number of aggregators:           1

Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------------------------------
1      Po1(SD)          -        Fa0/14(D)   Fa0/15(D)
```

The messages at the top of the example specifically state what the switch does when determining whether the interface settings match. In this case, SW1 detects the different STP costs. SW1 does not use F0/14, does not use F0/15, and even places them into an err-disabled state. The switch also puts the PortChannel into err-disabled state. As a result, the PortChannel is not operational, and the physical interfaces are also not operational.

To solve this problem, you must reconfigure the physical interfaces to use the same STP settings. In addition, the PortChannel and physical interfaces must be **shutdown**, and then **no shutdown**, to recover from the err-disabled state. (Note that when a switch applies the **shutdown** and **no shutdown** commands to a PortChannel, it applies those same commands to the physical interfaces, as well; so, just do the **shutdown/no shutdown** on the PortChannel interface.)

Analyzing the Switch Data Plane Forwarding

STP and EtherChannel both have an impact on what a switch's forwarding logic can use. STP limits which interfaces the data plane even considers using by placing some ports in a blocking state (STP) or discarding state (RSTP), which in turn tells the data plane to simply not use that port. EtherChannel gives the data plane new ports to use in the switch's MAC address table—EtherChannels—while telling the data plane to not use the underlying physical interfaces in an EtherChannel in the MAC table.

6. An engineer migrates from a more traditional OSPFv2 configuration that uses **network** commands in OSPF configuration mode to instead use OSPFv2 interface configuration. Which of the following commands configures the area number assigned to an interface in this new configuration?
   a. The **area** command in interface configuration mode
   b. The **ip ospf** command in interface configuration mode
   c. The **router ospf** command in interface configuration mode
   d. The **network** command in interface configuration mode
7. Which of the following configuration settings on a router does not influence which IPv4 route a router chooses to add to its IPv4 routing table when using OSPFv2?
   a. **auto-cost reference-bandwidth**
   b. **delay**
   c. **bandwidth**
   d. **ip ospf cost**

Foundation Topics

Implementing Single-Area OSPFv2

OSPF configuration includes only a few required steps, but it has many optional steps. After an OSPF design has been chosen—a task that can be complex in larger IP internetworks—the configuration can be as simple as enabling OSPF on each router interface and placing that interface in the correct OSPF area.

This section shows several configuration examples, all with a single-area OSPF internetwork. Following those examples, the text goes on to cover several of the additional optional configuration settings. For reference, the following list outlines the configuration steps covered in this first major section of the chapter, as well as a brief reference to the required commands:

Config Checklist

Step 1. Use the **router ospf** *process-id* global command to enter OSPF configuration mode for a particular OSPF process.

Step 2. (Optional) Configure the OSPF router ID by doing the following:

- **A.** Use the **router-id** *id-value* router subcommand to define the router ID
- **B.** Use the **interface loopback** *number* global command, along with an **ip address** *address mask* command, to configure an IP address on a loopback interface (chooses the highest IP address of all working loopbacks)
- **C.** Rely on an interface IP address (chooses the highest IP address of all working nonloopbacks)

Step 3. Use one or more **network** *ip-address wildcard-mask* **area** *area-id* router subcommands to enable OSPFv2 on any interfaces matched by the configured address and mask, enabling OSPF on the interface for the listed area.

Step 4. (Optional) Use the **passive-interface** *type number* router subcommand to configure any OSPF interfaces as passive if no neighbors can or should be discovered on the interface.

For a more visual perspective on OSPFv2 configuration, Figure 8-1 shows the relationship between the key OSPF configuration commands. Note that the configuration creates a routing process in one part of the configuration, and then indirectly enables OSPF on each interface. The configuration does not name the interfaces on which OSPF is enabled, instead requiring IOS to apply some logic by comparing the OSPF **network** command to the interface **ip address** commands. The upcoming example discusses more about this logic.

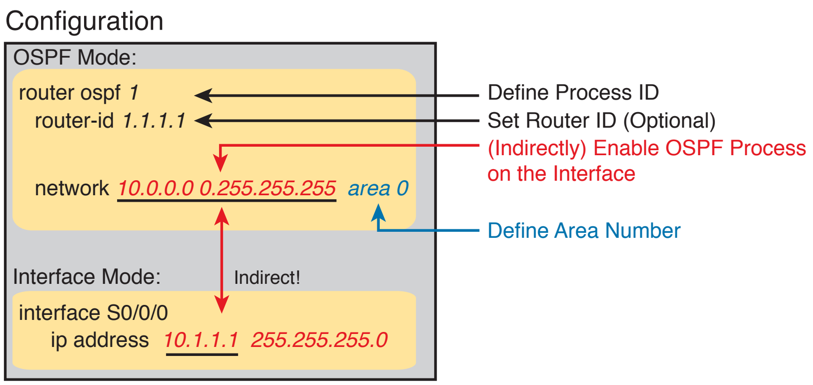


Figure 8-1 *Organization of OSPFv2 Configuration*

OSPF Single-Area Configuration

Figure 8-2 shows a sample network that will be used for the single-area OSPF configuration examples. All links sit in area 0. The design has four routers, each connected to one or two LANs. However, note that Routers R3 and R4, at the top of the figure, connect to the same two VLANs/subnets, so they will form neighbor relationships with each other over each of those VLANs as well. (The two switches at the top of the design are acting as Layer 2 switches.)

Example 8-1 shows the IPv4 addressing configuration on Router R3, before getting into the OSPF detail. The configuration enables 802.1Q trunking on R3's G0/0 interface, and assigns an IP address to each subinterface. (Not shown, switch S3 has configured trunking on the other side of that Ethernet link.)

Example 8-1 *IPv4 Address Configuration on R3 (Including VLAN Trunking)*

```
interface GigabitEthernet 0/0.341
 encapsulation dot1q 341
 ip address 10.1.3.1 255.255.255.128
!
interface GigabitEthernet 0/0.342
 encapsulation dot1q 342
 ip address 10.1.3.129 255.255.255.128
!
interface serial 0/0/0
 ip address 10.1.13.3 255.255.255.128
```

Answers to the "Do I Know This Already?" quiz:

1 B **2** A **3** A, E **4** C **5** D **6** B, **7** B

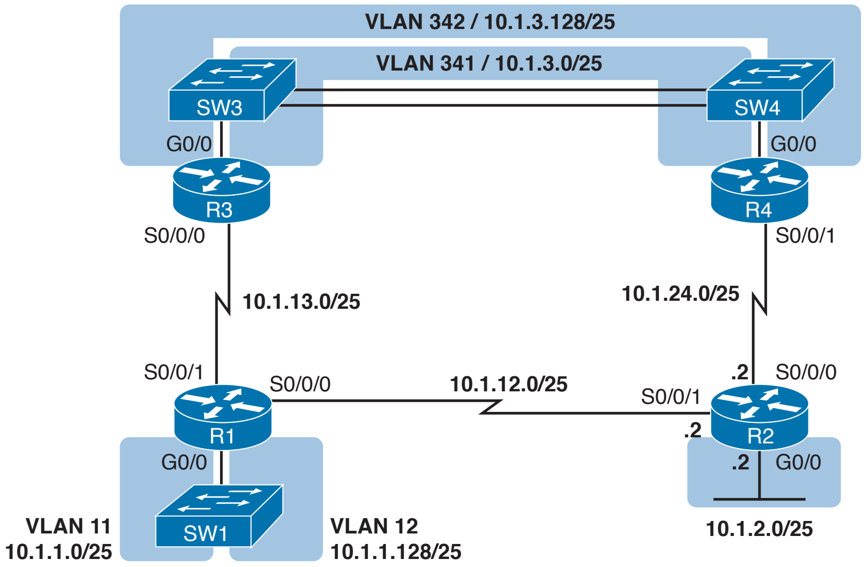


Figure 8-2 *Sample Network for OSPF Single-Area Configuration*

The beginning single-area configuration on R3, as shown in Example 8-2, enables OSPF on all the interfaces shown in Figure 8-2. First, the **router ospf 1** global command puts the user in OSPF configuration mode, and sets the OSPF *process-id*. This number just needs to be unique on the local router, allowing the router to support multiple OSPF processes in a single router by using different process IDs. (The **router** command uses the *process-id* to distinguish between the processes.) The *process-id* does not have to match on each router, and it can be any integer between 1 and 65,535.

Example 8-2 *OSPF Single-Area Configuration on R3 Using One* **network** *Command*

```
router ospf 1
 network 10.0.0.0 0.255.255.255 area 0
```

Speaking generally rather than about this example, the OSPF **network** command tells a router to find its local interfaces that match the first two parameters on the **network** command. Then, for each matched interface, the router enables OSPF on those interfaces, discovers neighbors, creates neighbor relationships, and assigns the interface to the area listed in the **network** command. (Note that the area can be configured as either an integer or a dotted-decimal number, but this book makes a habit of configuring the area number as an integer. The integer area numbers range from 0 through 4,294,967,295.)

For the specific command in Example 8-2, any matched interfaces are assigned to area 0. However, the first two parameters—the *ip_address* and *wildcard_mask* parameter values of 10.0.0.0 and 0.255.255.255—need some explaining. In this case, the command matches all three interfaces shown for Router R3; the next topic explains why.

Matching with the OSPF network Command

The key to understanding the traditional OSPFv2 configuration shown in this first example is to understand the OSPF **network** command. The OSPF **network** command compares the first parameter in the command to each interface IP address on the local router, trying to find a match. However, rather than comparing the entire number in the **network** command to the entire IPv4 address on the interface, the router can compare a subset of the octets, based on the wildcard mask, as follows:

Key Topic

Wildcard 0.0.0.0: Compare all 4 octets. In other words, the numbers must exactly match.

Wildcard 0.0.0.255: Compare the first 3 octets only. Ignore the last octet when comparing the numbers.

Wildcard 0.0.255.255: Compare the first 2 octets only. Ignore the last 2 octets when comparing the numbers.

Wildcard 0.255.255.255: Compare the first octet only. Ignore the last 3 octets when comparing the numbers.

Wildcard 255.255.255.255: Compare nothing—this wildcard mask means that all addresses will match the **network** command.

Basically, a wildcard mask value of 0 in an octet tells IOS to compare to see if the numbers match, and a value of 255 tells IOS to ignore that octet when comparing the numbers.

The **network** command provides many flexible options because of the wildcard mask. For example, in Router R3, many **network** commands could be used, with some matching all interfaces, and some matching a subset of interfaces. Table 8-2 shows a sampling of options, with notes.

Table 8-2 Example OSPF **network** Commands on R3, with Expected Results

Command	Logic in Command	Matched Interfaces
network 10.1.0.0 0.0.255.255	Match interface IP addresses that begin with 10.1	G0/0.341 G0/0.342 S0/0/0
network 10.0.0.0 0.255.255.255	Match interface IP addresses that begin with 10	G0/0.341 G0/0.342 S0/0/0
network 0.0.0.0 255.255.255.255	Match all interface IP addresses	G0/0.341 G0/0.342 S0/0/0
network 10.1.13.0 0.0.0.255	Match interface IP addresses that begin with 10.1.13	S0/0/0
network 10.1.3.1 0.0.0.0	Match one IP address: 10.1.3.1	G0/0.341

The wildcard mask gives the local router its rules for matching its own interfaces. For example, Example 8-2 shows R3 using the **network 10.0.0.0 0.255.255.255 area 0** command. However, the wildcard mask allows for many different valid OSPF configurations. For instance, in that same internetwork, Routers R1 and R2 could use the configuration shown in Example 8-3, with two other wildcard masks. In both routers, OSPF is enabled on all the interfaces shown in Figure 8-2.

Example 8-3 *OSPF Configuration on Routers R1 and R2*

```
! R1 configuration next - one network command enables OSPF
! on all three interfaces
router ospf 1
 network 10.1.0.0 0.0.255.255 area 0
```

```
! R2 configuration next - One network command per interface
router ospf 1
 network 10.1.12.2 0.0.0.0 area 0
 network 10.1.24.2 0.0.0.0 area 0
 network 10.1.2.2 0.0.0.0 area 0
```

Finally, note that other wildcard mask values can be used as well, as long as the wildcard mask in binary is one unbroken string of 0s and another single string of binary 1s. Basically, that includes all wildcard masks that could be used to match all IP addresses in a subnet, as discussed in the “Finding the Right Wildcard Mask to Match a Subnet” section of Chapter 16, “Basic IPv4 Access Control Lists” (which is Chapter 25 of the ICND1 Cert Guide). For example, a mask of 0.255.255.0 would not be allowed.

NOTE The first two parameters of the **network** command are the address and the wildcard mask. By convention, if the wildcard mask octet is 255, the matching address octet should be configured as a 0. Interestingly, IOS will actually accept a **network** command that breaks this rule, but then IOS will change that octet of the address to a 0 before putting it into the running configuration file. For example, IOS will change a typed command that begins with **network 1.2.3.4 0.0.255.255** to **network 1.2.0.0 0.0.255.255**.

Verifying OSPFv2 Single Area

As mentioned in Chapter 7, OSPF routers use a three-step process to eventually add OSPF-learned routes to the IP routing table. First, they create neighbor relationships. Then they build and flood LSAs, so each router in the same area has a copy of the same LSDB. Finally, each router independently computes its own IP routes using the SPF algorithm and adds them to its routing table.

The **show ip ospf neighbor**, **show ip ospf database**, and **show ip route** commands display information for each of these three steps, respectively. To verify OSPF, you can use the same sequence. Or, you can just go look at the IP routing table, and if the routes look correct, OSPF probably worked.

For example, first, examine the list of neighbors known on Router R3 from the configuration in Examples 8-1, 8-2, and 8-3. R3 should have one neighbor relationship with R1, over the serial link. It also has two neighbor relationships with R4, over the two different VLANs to which both routers connect. Example 8-4 shows all three.

Key Topic

Example 8-4 *OSPF Neighbors on Router R3 from Figure 8-2*

```
R3# show ip ospf neighbor

Neighbor ID     Pri   State        Dead Time   Address         Interface
```

```
1.1.1.1           0   FULL/  -    00:00:33    10.1.13.1      Serial0/0/0
10.1.24.4         1   FULL/DR     00:00:35    10.1.3.130     GigabitEthernet0/0.342
10.1.24.4         1   FULL/DR     00:00:36    10.1.3.4       GigabitEthernet0/0.341
```

The detail in the output mentions several important facts, and for most people, working right to left works best in this case. For example, looking at the headings:

Interface: This is the local router's interface connected to the neighbor. For example, the first neighbor in the list is reachable through R3's S0/0/0 interface.

Address: This is the neighbor's IP address on that link. Again, for this first neighbor, the neighbor, which is R1, uses IP address 10.1.13.1.

State: While many possible states exist, for the details discussed in this chapter, FULL is the correct and fully working state in this case.

Neighbor ID: This is the router ID of the neighbor.

Next, Example 8-5 shows the contents of the LSDB on Router R3. Interestingly, when OSPF is working correctly in an internetwork with a single-area design, all the routers will have the same LSDB contents. So, the **show ip ospf database** command in Example 8-5 should list the same exact information, no matter on which of the four routers it is issued.

Example 8-5 *OSPF Database on Router R3 from Figure 8-2*

```
R3# show ip ospf database

            OSPF Router with ID (10.1.13.3) (Process ID 1)

                Router Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum Link count
1.1.1.1         1.1.1.1         498         0x80000006 0x002294 6
2.2.2.2         2.2.2.2         497         0x80000004 0x00E8C6 5
10.1.13.3       10.1.13.3       450         0x80000003 0x001043 4
10.1.24.4       10.1.24.4       451         0x80000003 0x009D7E 4

                Net Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum
10.1.3.4        10.1.24.4       451         0x80000001 0x0045F8
10.1.3.130      10.1.24.4       451         0x80000001 0x00546B
```

For the purposes of this book, do not be concerned about the specifics in the output of this command. However, for perspective, note that the LSDB should list one "Router Link State" (Type 1 Router LSA) for each of the routers in the same area. In this design, all four routers are in the same area, so there are four highlighted Type 1 LSAs listed.

Next, Example 8-6 shows R3's IPv4 routing table with the **show ip route** command. Note that it lists connected routes as well as OSPF routes. Take a moment to look back at Figure 8-2, and look for the subnets that are not locally connected to R3. Then look for those routes in the output in Example 8-5.

Example 8-6 *IPv4 Routes Added by OSPF on Router R3 from Figure 8-2*

```
R3# show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
! Legend lines omitted for brevity


      10.0.0.0/8 is variably subnetted, 11 subnets, 2 masks
O        10.1.1.0/25 [110/65] via 10.1.13.1, 00:13:28, Serial0/0/0
O        10.1.1.128/25 [110/65] via 10.1.13.1, 00:13:28, Serial0/0/0
O        10.1.2.0/25 [110/66] via 10.1.3.130, 00:12:41, GigabitEthernet0/0.342
                     [110/66] via 10.1.3.4, 00:12:41, GigabitEthernet0/0.341
C        10.1.3.0/25 is directly connected, GigabitEthernet0/0.341
L        10.1.3.1/32 is directly connected, GigabitEthernet0/0.341
C        10.1.3.128/25 is directly connected, GigabitEthernet0/0.342
L        10.1.3.129/32 is directly connected, GigabitEthernet0/0.342
O        10.1.12.0/25 [110/128] via 10.1.13.1, 00:13:28, Serial0/0/0
C        10.1.13.0/25 is directly connected, Serial0/0/0
L        10.1.13.3/32 is directly connected, Serial0/0/0
O        10.1.24.0/25
           [110/65] via 10.1.3.130, 00:12:41, GigabitEthernet0/0.342
           [110/65] via 10.1.3.4, 00:12:41, GigabitEthernet0/0.341
```

First, take a look at the bigger ideas confirmed by this output. The code of "O" on the left identifies a route as being learned by OSPF. The output lists five such IP routes. From Figure 8-2, five subnets exist that are not connected subnets off Router R3. Looking for a quick count of OSPF routes, versus nonconnected routes in the diagram, gives a quick check of whether OSPF learned all routes.

Next, take a look at the first route (to subnet 10.1.1.0/25). It lists the subnet ID and mask, identifying the subnet. It also lists two numbers in brackets. The first, 110, is the administrative distance of the route. All the OSPF routes in this example use the default of 110. The second number, 65, is the OSPF metric for this route.

Additionally, the **show ip protocols** command is also popular as a quick look at how any routing protocol works. This command lists a group of messages for each IPv4 routing protocol running on a router. Example 8-7 shows a sample, this time taken from Router R3.

Example 8-7 *The* **show ip protocols** *Command on R3*

```
R3# show ip protocols
*** IP Routing is NSF aware ***

Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 10.1.13.3
```

```
 Number of areas in this router is 1. 1 normal 0 stub 0 nssa
 Maximum path: 4
 Routing for Networks:
   10.0.0.0 0.255.255.255 area 0
 Routing Information Sources:
   Gateway         Distance      Last Update
   1.1.1.1              110      06:26:17
   2.2.2.2              110      06:25:30
   10.1.24.4            110      06:25:30
 Distance: (default is 110)
```

The output shows several interesting facts. The first highlighted line repeats the parameters on the **router ospf 1** global configuration command. The second highlighted item points out R3's router ID, as discussed further in the next section. The third highlighted line repeats more configuration, listing the parameters of the **network 10.0.0.0 0.255.255.255 area 0** OSPF subcommand. Finally, the last highlighted item in the example acts as a heading before a list of known OSPF routers, by router ID.

Configuring the OSPF Router ID

While OSPF has many other optional features, most enterprise networks that use OSPF choose to configure each router's OSPF router ID. OSPF-speaking routers must have a router ID (RID) for proper operation. By default, routers will choose an interface IP address to use as the RID. However, many network engineers prefer to choose each router's router ID, so command output from commands like **show ip ospf neighbor** lists more recognizable router IDs.

8

To choose its RID, a Cisco router uses the following process when the router reloads and brings up the OSPF process. Note that when one of these steps identifies the RID, the process stops.

Key Topic

1. If the **router-id** *rid* OSPF subcommand is configured, this value is used as the RID.
2. If any loopback interfaces have an IP address configured, and the interface has an interface status of up, the router picks the highest numeric IP address among these loopback interfaces.
3. The router picks the highest numeric IP address from all other interfaces whose interface status code (first status code) is up. (In other words, an interface in up/down state will be included by OSPF when choosing its router ID.)

The first and third criteria should make some sense right away: the RID is either configured or is taken from a working interface's IP address. However, this book has not yet explained the concept of a *loopback interface*, as mentioned in Step 2.

A loopback interface is a virtual interface that can be configured with the **interface loopback** *interface-number* command, where *interface-number* is an integer. Loopback interfaces are always in an "up and up" state unless administratively placed in a shutdown state. For example, a simple configuration of the command **interface loopback 0**, followed by **ip address 2.2.2.2 255.255.255.0**, would create a loopback interface and assign it an IP address. Because loopback interfaces do not rely on any hardware, these interfaces can be up/up whenever IOS is running, making them good interfaces on which to base an OSPF RID.

Example 8-8 shows the configuration that existed in Routers R1 and R2 before the creation of the **show** command output in Examples 8-4, 8-5, and 8-6. R1 set its router ID using the direct method, while R2 used a loopback IP address.

Example 8-8 *OSPF Router ID Configuration Examples*

```
! R1 Configuration first
router ospf 1
 router-id 1.1.1.1
 network 10.1.0.0 0.0.255.255 area 0
```

```
! R2 Configuration next
!
interface Loopback2
 ip address 2.2.2.2 255.255.255.255
```

Each router chooses its OSPF RID when OSPF is initialized, which happens when the router boots or when a CLI user stops and restarts the OSPF process (with the **clear ip ospf process** command). So, if OSPF comes up, and later the configuration changes in a way that would impact the OSPF RID, OSPF does not change the RID immediately. Instead, IOS waits until the next time the OSPF process is restarted.

Example 8-9 shows the output of the **show ip ospf** command on R1, after the configuration of Example 8-8 was made, and after the router was reloaded, which made the OSPF router ID change.

Example 8-9 *Confirming the Current OSPF Router ID*

```
R1# show ip ospf
 Routing Process "ospf 1" with ID 1.1.1.1
! lines omitted for brevity
```

OSPF Passive Interfaces

Once OSPF has been enabled on an interface, the router tries to discover neighboring OSPF routers and form a neighbor relationship. To do so, the router sends OSPF Hello messages on a regular time interval (called the Hello Interval). The router also listens for incoming Hello messages from potential neighbors.

Sometimes, a router does not need to form neighbor relationships with neighbors on an interface. Often, no other routers exist on a particular link, so the router has no need to keep sending those repetitive OSPF Hello messages.

When a router does not need to discover neighbors off some interface, the engineer has a couple of configuration options. First, by doing nothing, the router keeps sending the messages, wasting some small bit of CPU cycles and effort. Alternately, the engineer can configure the interface as an OSPF passive interface, telling the router to do the following:

Key Topic

- Quit sending OSPF Hellos on the interface.
- Ignore received Hellos on the interface.
- Do not form neighbor relationships over the interface.

CHAPTER 16

Basic IPv4 Access Control Lists

This chapter covers the following exam topics:

4.0 Infrastructure Services

4.4 Configure, verify, and troubleshoot IPv4 and IPv6 access list for traffic filtering

4.1.a Standard

IPv4 access control lists (ACL) give network engineers the ability to program a filter into a router. Each router, on each interface, for both the inbound and outbound direction, can enable a different ACL with different rules. Each ACL's rules tell the router which packets to discard, and which to allow through.

This chapter discusses the basics of IPv4 ACLs, and in particular, one type of IP ACL: standard numbered IP ACLs. The next chapter, titled, "Advanced IPv4 Access Control Lists," completes the discussion by describing other types of IP ACLs.

Those of you who have already read the ICND1 100-105 Cert Guide will probably begin to recognize a lot of similar figures and examples. In fact, this chapter's content is identical to the ICND1 book's Chapter 25. Likewise, this book's Chapter 17 is identical to the ICND1 book's Chapter 26. I kept the IPv4 ACL content identical in both books in an attempt to make you a little more efficient in your use of time. Here's why:

- Cisco's ICND1 and ICND2 ACL exam topics overlap quite a bit, so both books need to cover those topics.
- If this book had some ACL content that was not identical to the ICND1 book, you would probably want to read it all, in case there was something new.
- With these two chapters, the version in the ICND1 and ICND2 books have the exact same content, so you can make better use of your time here with ICND2. If you have read these chapters in the ICND1 book, refer to your notes, skim this chapter as appropriate, and use the chapter review tools to make sure you have mastered the content.

I believe this plan will work better for those of you using both books, and will be of no difference to those of you using only the ICND1 book or only the ICND2 book. Regardless, jump in and read/review about IPv4 ACLs.

"Do I Know This Already?" Quiz

Take the quiz (either here, or use the PCPT software) if you want to use the score to help you decide how much time to spend on this chapter. The answers are at the bottom of the page following the quiz, and the explanations are in DVD Appendix C and in the PCPT software.

Table 16-1 "Do I Know This Already?" Foundation Topics Section-to-Question Mapping

Foundation Topics Section	Questions
IP Access Control List Basics	1
Standard Numbered IPv4 ACLs	2–5
Practice Applying Standard IP ACLs	6

1. Barney is a host with IP address 10.1.1.1 in subnet 10.1.1.0/24. Which of the following are things that a standard IP ACL could be configured to do? (Choose two answers.)
   a. Match the exact source IP address.
   b. Match IP addresses 10.1.1.1 through 10.1.1.4 with one **access-list** command without matching other IP addresses.
   c. Match all IP addresses in Barney's subnet with one **access-list** command without matching other IP addresses.
   d. Match only the packet's destination IP address.

2. Which of the following answers list a valid number that can be used with standard numbered IP ACLs? (Choose two answers.)
   a. 1987
   b. 2187
   c. 187
   d. 87

3. Which of the following wildcard masks is most useful for matching all IP packets in subnet 10.1.128.0, mask 255.255.255.0?
   a. 0.0.0.0
   b. 0.0.0.31
   c. 0.0.0.240
   d. 0.0.0.255
   e. 0.0.15.0
   f. 0.0.248.255

4. Which of the following wildcard masks is most useful for matching all IP packets in subnet 10.1.128.0, mask 255.255.240.0?
   a. 0.0.0.0
   b. 0.0.0.31
   c. 0.0.0.240
   d. 0.0.0.255
   e. 0.0.15.255
   f. 0.0.248.255

5. ACL 1 has three statements, in the following order, with address and wildcard mask values as follows: 1.0.0.0 0.255.255.255, 1.1.0.0 0.0.255.255, and 1.1.1.0 0.0.0.255. If a router tried to match a packet sourced from IP address 1.1.1.1 using this ACL, which ACL statement does a router consider the packet to have matched?

- **a.** First
- **b.** Second
- **c.** Third
- **d.** Implied deny at the end of the ACL

6. Which of the following **access-list** commands matches all packets sent from hosts in subnet 172.16.4.0/23?

- **a.** **access-list 1 permit 172.16.0.5 0.0.255.0**
- **b.** **access-list 1 permit 172.16.4.0 0.0.1.255**
- **c.** **access-list 1 permit 172.16.5.0**
- **d.** **access-list 1 permit 172.16.5.0 0.0.0.127**

Foundation Topics

IPv4 Access Control List Basics

IPv4 access control lists (IP ACL) give network engineers a way to identify different types of packets. To do so, the ACL configuration lists values that the router can see in the IP, TCP, UDP, and other headers. For example, an ACL can match packets whose source IP address is 1.1.1.1, or packets whose destination IP address is some address in subnet 10.1.1.0/24, or packets with a destination port of TCP port 23 (Telnet).

IPv4 ACLs perform many functions in Cisco routers, with the most common use as a packet filter. Engineers can enable ACLs on a router so that the ACL sits in the forwarding path of packets as they pass through the router. After it is enabled, the router considers whether each IP packet will either be discarded or allowed to continue as if the ACL did not exist.

However, ACLs can be used for many other IOS features as well. As an example, ACLs can be used to match packets for applying Quality of Service (QoS) features. QoS allows a router to give some packets better service, and other packets worse service. For example, packets that hold digitized voice need to have very low delay, so ACLs can match voice packets, with QoS logic in turn forwarding voice packets more quickly than data packets.

This first section introduces IP ACLs as used for packet filtering, focusing on these aspects of ACLs: the locations and direction in which to enable ACLs, matching packets by examining headers, and taking action after a packet has been matched.

ACL Location and Direction

Cisco routers can apply ACL logic to packets at the point at which the IP packets enter an interface, or the point at which they exit an interface. In other words, the ACL becomes associated with an interface and for a direction of packet flow (either in or out). That is, the ACL can be applied inbound to the router, before the router makes its forwarding (routing) decision, or outbound, after the router makes its forwarding decision and has determined the exit interface to use.

The arrows in Figure 16-1 show the locations at which you could filter packets flowing left to right in the topology. For example, imagine that you wanted to allow packets sent by host A to server S1, but to discard packets sent by host B to server S1. Each arrowed line represents a location and direction at which a router could apply an ACL, filtering the packets sent by host B.

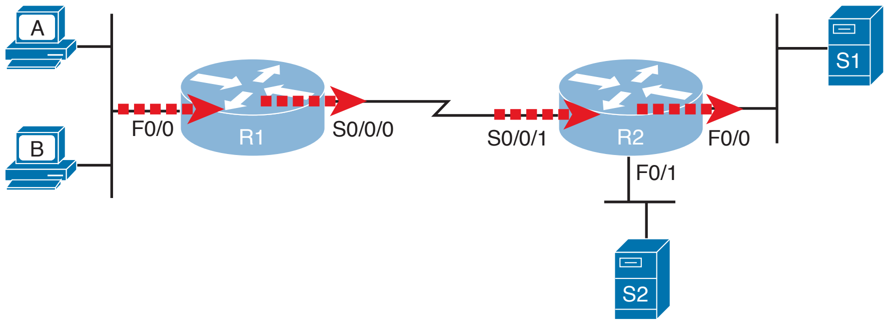


Figure 16-1 *Locations to Filter Packets from Hosts A and B Going Toward Server S1*

The four arrowed lines in the figure point out the location and direction for the router interfaces used to forward the packet from host B to server S1. In this particular example, those interfaces and direction are inbound on R1's F0/0 interface, outbound on R1's S0/0/0 interface, inbound on R2's S0/0/1 interface, and outbound on R2's F0/0 interface. If, for example, you enabled an ACL on R2's F0/1 interface, in either direction, that ACL could not possibly filter the packet sent from host B to server S1, because R2's F0/1 interface is not part of the route from B to S1.

Key Topic

In short, to filter a packet, you must enable an ACL on an interface that processes the packet, in the same direction the packet flows through that interface.

When enabled, the router then processes every inbound or outbound IP packet using that ACL. For example, if enabled on R1 for packets inbound on interface F0/0, R1 would compare every inbound IP packet on F0/0 to the ACL to decide that packet's fate: to continue unchanged, or to be discarded.

Matching Packets

When you think about the location and direction for an ACL, you must already be thinking about what packets you plan to filter (discard), and which ones you want to allow through. To tell the router those same ideas, you must configure the router with an IP ACL that matches packets. *Matching packets* refers to how to configure the ACL commands to look at each packet, listing how to identify which packets should be discarded, and which should be allowed through.

Answers to the "Do I Know This Already?" quiz:

1 A, C **2** A, D **3** D **4** E **5** A **6** B

Each IP ACL consists of one or more configuration commands, with each command listing details about values to look for inside a packet's headers. Generally, an ACL command uses logic like "look for these values in the packet header, and if found, discard the packet." (The action could instead be to allow the packet, rather than discard.) Specifically, the ACL looks for header fields you should already know well, including the source and destination IP addresses, plus TCP and UDP port numbers.

For example, consider an example with Figure 16-2, in which you want to allow packets from host A to server S1, but to discard packets from host B going to that same server. The hosts all now have IP addresses, and the figure shows pseudocode for an ACL on R2. Figure 16-2 also shows the chosen location to enable the ACL: inbound on R2's S0/0/1 interface.

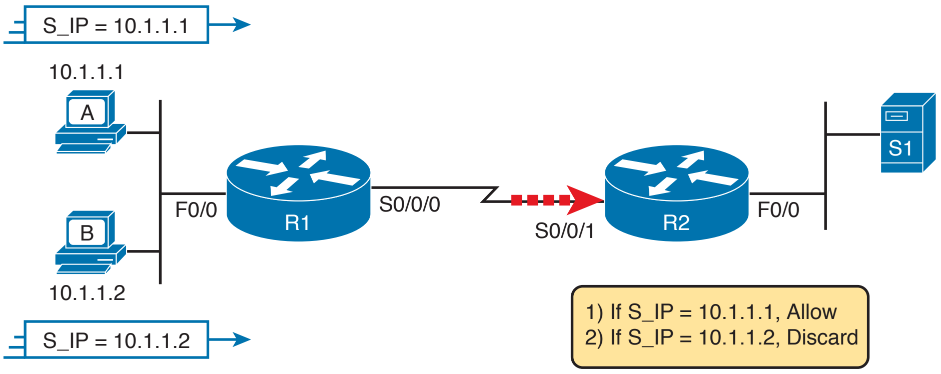


Figure 16-2 *Pseudocode to Demonstrate ACL Command-Matching Logic*

Figure 16-2 shows a two-line ACL in a rectangle at the bottom, with simple matching logic: both statements just look to match the source IP address in the packet. When enabled, R2 looks at every inbound IP packet on that interface and compares each packet to those two ACL commands. Packets sent by host A (source IP address 10.1.1.1) are allowed through, and those sourced by host B (source IP address 10.1.1.2) are discarded.

Taking Action When a Match Occurs

When using IP ACLs to filter packets, only one of two actions can be chosen. The configuration commands use the keywords **deny** and **permit**, and they mean (respectively) to discard the packet or to allow it to keep going as if the ACL did not exist.

This book focuses on using ACLs to filter packets, but IOS uses ACLs for many more features. Those features typically use the same matching logic. However, in other cases, the **deny** or **permit** keywords imply some other action.

Types of IP ACLs

Cisco IOS has supported IP ACLs since the early days of Cisco routers. Beginning with the original standard numbered IP ACLs in the early days of IOS, which could enable the logic shown earlier around Figure 16-2, Cisco has added many ACL features, including the following:

- Standard numbered ACLs (1–99)
- Extended numbered ACLs (100–199)
- Additional ACL numbers (1300–1999 standard, 2000–2699 extended)
- Named ACLs
- Improved editing with sequence numbers

This chapter focuses solely on standard numbered IP ACLs, while the next chapter discusses the other three primary categories of IP ACLs. Briefly, IP ACLs will be either numbered or named in that the configuration identifies the ACL either using a number or a name. ACLs will also be either standard or extended, with extended ACLs having much more robust abilities in matching packets. Figure 16-3 summarizes the big ideas related to categories of IP ACLs.

Key Topic

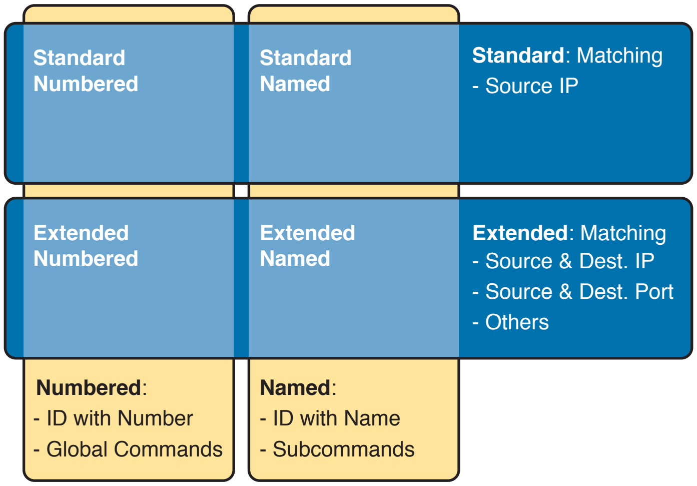


Figure 16-3 *Comparisons of IP ACL Types*

Standard Numbered IPv4 ACLs

The title of this section serves as a great introduction, if you can decode what Cisco means by each specific word. This section is about a type of Cisco filter (*ACL*) that matches only the source IP address of the packet (*standard*), is configured to identify the ACL using numbers rather than names (*numbered*), and looks at IPv4 packets.

This section examines the particulars of standard numbered IP ACLs. First, it examines the idea that one ACL is a list, and what logic that list uses. Following that, the text closely looks at how to match the source IP address field in the packet header, including the syntax of the commands. This section ends with a complete look at the configuration and verification commands to implement standard ACLs.

List Logic with IP ACLs

A single ACL is both a single entity and, at the same time, a list of one or more configuration commands. As a single entity, the configuration enables the entire ACL on an interface, in a specific direction, as shown earlier in Figure 16-1. As a list of commands, each command has different matching logic that the router must apply to each packet when filtering using that ACL.

When doing ACL processing, the router processes the packet, compared to the ACL, as follows:

Key Topic

ACLs use first-match logic. Once a packet matches one line in the ACL, the router takes the action listed in that line of the ACL, and stops looking further in the ACL.

To see exactly what that means, consider the example built around Figure 16-4. The figure shows an example ACL 1 with three lines of pseudocode. This example applies ACL 1 on R2's S0/0/1 interface, inbound (the same location as in earlier Figure 16-2).

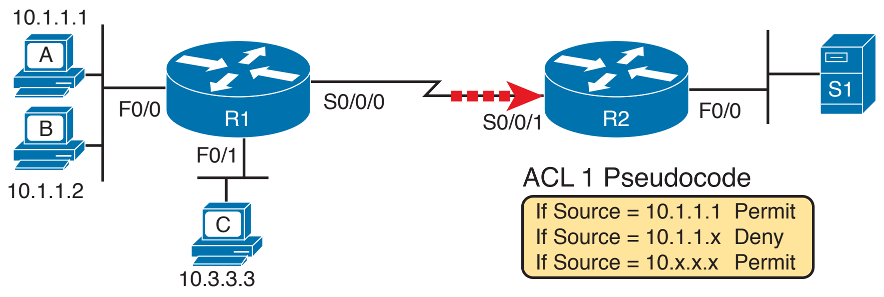

10.1.1.1
A
B
10.1.1.2
F0/0
R1
F0/1
C
10.3.3.3
S0/0/0
S0/0/1
R2
F0/0
S1
ACL 1 Pseudocode
If Source = 10.1.1.1 Permit
If Source = 10.1.1.x Deny
If Source = 10.x.x.x Permit

Figure 16-4 *Backdrop for Discussion of List Process with IP ACLs*

Consider the first-match ACL logic for a packet sent by host A to server S1. The source IP address will be 10.1.1.1, and it will be routed so that it enters R2's S0/0/1 interface, driving R2's ACL 1 logic. R2 compares this packet to the ACL, matching the first item in the list with a permit action. So this packet should be allowed through, as shown in Figure 16-5, on the left.

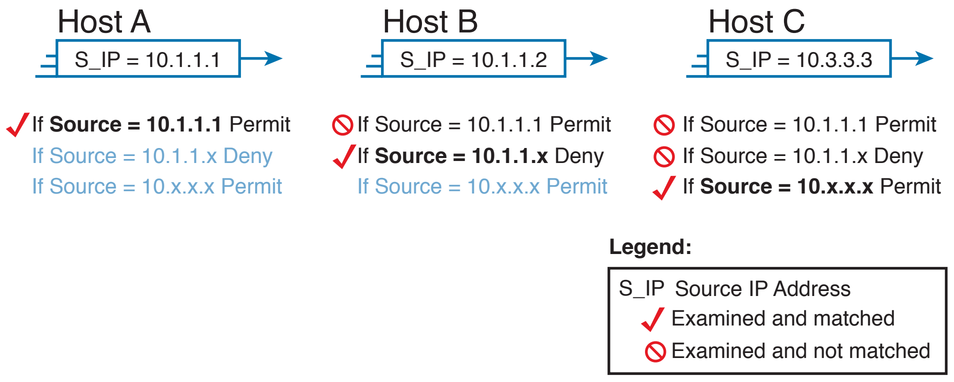

Host A
S_IP = 10.1.1.1
If Source = 10.1.1.1 Permit
If Source = 10.1.1.x Deny
If Source = 10.x.x.x Permit
Host B
S_IP = 10.1.1.2
If Source = 10.1.1.1 Permit
If Source = 10.1.1.x Deny
If Source = 10.x.x.x Permit
Host C
S_IP = 10.3.3.3
If Source = 10.1.1.1 Permit
If Source = 10.1.1.x Deny
If Source = 10.x.x.x Permit
Legend:
S_IP Source IP Address
Examined and matched
Examined and not matched

Figure 16-5 *ACL Items Compared for Packets from Hosts A, B, and C in Figure 16-4*

Next, consider a packet sent by host B, source IP address 10.1.1.2. When the packet enters R2's S0/0/1 interface, R2 compares the packet to ACL 1's first statement, and does not make a match (10.1.1.1 is not equal to 10.1.1.2). R2 then moves to the second statement, which requires some clarification. The ACL pseudocode, back in Figure 16-4, shows 10.1.1.x, which is meant to be shorthand that any value can exist in the last octet. Comparing only the first three octets, R2 decides that this latest packet does have a source IP address that begins with the first three octets 10.1.1, so R2 considers that to be a match on the second statement. R2 takes the listed action (deny), discarding the packet. R2 also stops ACL processing on the packet, ignoring the third line in the ACL.

Finally, consider a packet sent by host C, again to server S1. The packet has source IP address 10.3.3.3, so when it enters R2's S0/0/1 interface, and drives ACL processing on R2, R2 looks at the first command in ACL 1. R2 does not match the first ACL command (10.1.1.1 in the command is not equal to the packet's 10.3.3.3). R2 looks at the second command, compares the first three octets (10.1.1) to the packet source IP address (10.3.3), and still finds no match. R2 then looks at the third command. In this case, the wildcard means ignore the last three octets, and just compare the first octet (10), so the packet matches. R2 then takes the listed action (permit), allowing the packet to keep going.

This sequence of processing an ACL as a list happens for any type of IOS ACL: IP, other protocols, standard or extended, named or numbered.

Finally, if a packet does not match any of the items in the ACL, the packet is discarded. The reason is that every IP ACL has a *deny all* statement implied at the end of the ACL. It does not exist in the configuration, but if a router keeps searching the list, and no match is made by the end of the list, IOS considers the packet to have matched an entry that has a **deny** action.

Matching Logic and Command Syntax

Standard numbered IP ACLs use the following global command:

```
access-list {1-99 | 1300-1999} {permit | deny} matching-parameters
```

Each standard numbered ACL has one or more **access-list** commands with the same number, any number from the ranges shown in the preceding line of syntax. (One number is no better than the other.)

Besides the ACL number, each **access-list** command also lists the action (**permit** or **deny**), plus the matching logic. The rest of this section examines how to configure the matching parameters, which for standard ACLs, means that you can only match the source IP address, or portions of the source IP address using something called an ACL wildcard mask.

Matching the Exact IP Address

To match a specific source IP address, the entire IP address, all you have to do is type that IP address at the end of the command. For example, the previous example uses pseudocode for "permit if source = 10.1.1.1." The following command configures that logic with correct syntax using ACL number 1:

```
access-list 1 permit 10.1.1.1
```

Matching the exact full IP address is that simple.

In earlier IOS versions, the syntax included a **host** keyword. Instead of simply typing the full IP address, you first typed the **host** keyword, and then the IP address. Note that in later

IOS versions, if you use the **host** keyword, IOS accepts the command, but then removes the keyword.

```
access-list 1 permit host 10.1.1.1
```

Matching a Subset of the Address with Wildcards

Often, the business goals you want to implement with an ACL do not match a single particular IP address, but rather a range of IP addresses. Maybe you want to match all IP addresses in a subnet. Maybe you want to match all IP addresses in a range of subnets. Regardless, you want to check for more than one IP address in a range of addresses.

IOS allows standard ACLs to match a range of addresses using a tool called a *wildcard mask*. Note that this is not a subnet mask. The wildcard mask (which this book abbreviates as *WC mask*) gives the engineer a way to tell IOS to ignore parts of the address when making comparisons, essentially treating those parts as wildcards, as if they already matched.

You can think about WC masks in decimal and in binary, and both have their uses. To begin, think about WC masks in decimal, using these rules:

Key Topic

Decimal 0: The router must compare this octet as normal.

Decimal 255: The router ignores this octet, considering it to already match.

Keeping these two rules in mind, consider Figure 16-6, which demonstrates this logic using three different but popular WC masks: one that tells the router to ignore the last octet, one that tells the router to ignore the last two octets, and one that tells the router to ignore the last three octets.

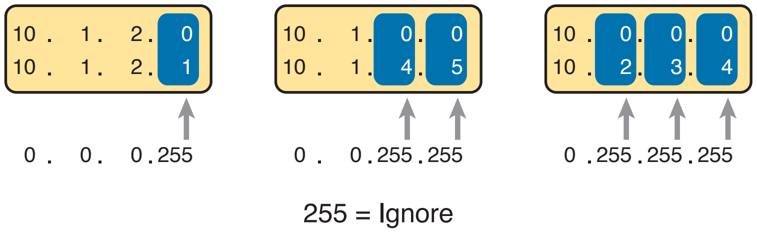


Figure 16-6 *Logic for WC Masks 0.0.0.255, 0.0.255.255, and 0.255.255.255*

All three examples in the boxes of Figure 16-6 show two numbers that are clearly different. The WC mask causes IOS to compare only some of the octets, while ignoring other octets. All three examples result in a match, because each wildcard mask tells IOS to ignore some octets. The example on the left shows WC mask 0.0.0.255, which tells the router to treat the last octet as a wildcard, essentially ignoring that octet for the comparison. Similarly, the middle example shows WC mask 0.0.255.255, which tells the router to ignore the two octets on the right. The rightmost case shows WC mask 0.255.255.255, telling the router to ignore the last three octets when comparing values.

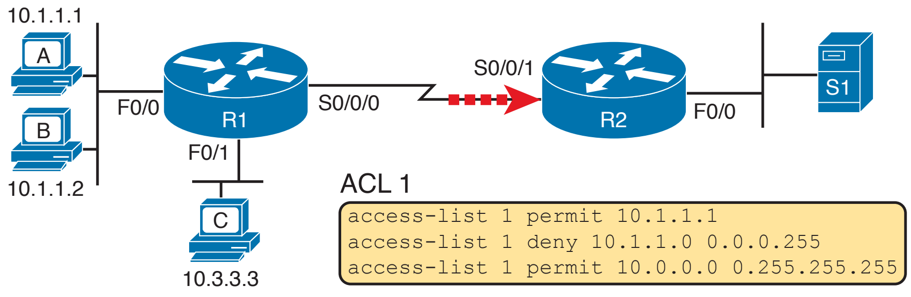
To see the WC mask in action, think back to the earlier example related to Figure 16-4 and Figure 16-5. The pseudocode ACL in those two figures used logic that can be created using a WC mask. As a reminder, the logic in the pseudocode ACL in those two figures included the following:

Line 1: Match and permit all packets with a source address of exactly 10.1.1.1.

Line 2: Match and deny all packets with source addresses with first three octets 10.1.1.

Line 3: Match and permit all addresses with first single octet 10.

Figure 16-7 shows the updated version of Figure 16-4, but with the completed, correct syntax, including the WC masks. In particular, note the use of WC mask 0.0.0.255 in the second command, telling R2 to ignore the last octet of the number 10.1.1.0, and the WC mask 0.255.255.255 in the third command, telling R2 to ignore the last three octets in the value 10.0.0.0.



```
access-list 1 permit 10.1.1.1
access-list 1 deny 10.1.1.0 0.0.0.255
access-list 1 permit 10.0.0.0 0.255.255.255
```

Figure 16-7 *Syntactically Correct ACL Replaces Pseudocode from Figure 16-4*

Finally, note that when using a WC mask, the **access-list** command's loosely defined *source* parameter should be a 0 in any octets where the WC mask is a 255. IOS will specify a source address to be 0 for the parts that will be ignored, even if nonzero values were configured.

Binary Wildcard Masks

Wildcard masks, as dotted-decimal number (DDN) values, actually represent a 32-bit binary number. As a 32-bit number, the WC mask actually directs the router's logic bit by bit. In short, a WC mask bit of 0 means the comparison should be done as normal, but a binary 1 means that the bit is a wildcard, and can be ignored when comparing the numbers.

Thankfully, for the purposes of CCENT and CCNA R&S study, and for most real-world applications, you can ignore the binary WC mask. Why? Well, we generally want to match a range of addresses that can be easily identified by a subnet number and mask, whether it be a real subnet, or a summary route that groups subnets together. If you can describe the range of addresses with a subnet number and mask, you can find the numbers to use in your ACL with some simple decimal math, as discussed next.

NOTE If you really want to know the binary mask logic, take the two DDN numbers the ACL will compare (one from the **access-list** command, and the other from the packet header) and convert both to binary. Then, also convert the WC mask to binary. Compare the first two binary numbers bit by bit, but also ignore any bits for which the WC mask happens to list a binary 1, because that tells you to ignore the bit. If all the bits you checked are equal, it's a match!

Finding the Right Wildcard Mask to Match a Subnet

In many cases, an ACL needs to match all hosts in a particular subnet. To match a subnet with an ACL, you can use the following shortcut:

Key Topic

- Use the subnet number as the source value in the **access-list** command.
- Use a wildcard mask found by subtracting the subnet mask from 255.255.255.255.

For example, for subnet 172.16.8.0 255.255.252.0, use the subnet number (172.16.8.0) as the address parameter, and then do the following math to find the wildcard mask:

$$\begin{array}{r} 255.255.255.255 \\ -255.255.252.0 \\ \hline 0.\quad 0.\quad 3.255 \end{array}$$

Continuing this example, a completed command for this same subnet would be as follows:

```
access-list 1 permit 172.16.8.0 0.0.3.255
```

The upcoming section, "Practice Applying Standard IP ACLs," gives you a chance to practice matching subnets when configuring ACLs.

Matching Any/All Addresses

In some cases, you will want one ACL command to match any and all packets that reach that point in the ACL. First, you have to know the (simple) way to match all packets using the **any** keyword. More importantly, you need to think about when to match any and all packets.

First, to match any and all packets with an ACL command, just use the **any** keyword for the address. For example, to permit all packets:

```
access-list 1 permit any
```

So, when and where should you use such a command? Remember, all Cisco IP ACLs end with an implicit deny any concept at the end of each ACL. That is, if a router compares a packet to the ACL, and the packet matches none of the configured statements, the router discards the packet. Want to override that default behavior? Configure a **permit any** at the end of the ACL.

You might also want to explicitly configure a command to deny all traffic (for example, **access-list 1 deny any**) at the end of an ACL. Why, when the same logic already sits at the end of the ACL anyway? Well, the ACL **show** commands list counters for the number of packets matched by each command in the ACL, but there is no counter for that implicit deny any concept at the end of the ACL. So, if you want to see counters for how many packets are matched by the deny any logic at the end of the ACL, configure an explicit **deny any**.

Implementing Standard IP ACLs

This chapter has already introduced all the configuration steps in bits and pieces. This section summarizes those pieces as a configuration process. The process also refers to the **access-list** command, whose generic syntax is repeated here for reference:

```
access-list access-list-number {deny | permit} source [source-wildcard]
```

Key Topic

Step 1. Plan the location (router and interface) and direction (in or out) on that interface:

A. Standard ACLs should be placed near to the destination of the packets so that they do not unintentionally discard packets that should not be discarded.

B. Because standard ACLs can only match a packet's source IP address, identify the source IP addresses of packets as they go in the direction that the ACL is examining.

Step 2. Configure one or more **access-list** global configuration commands to create the ACL, keeping the following in mind:

A. The list is searched sequentially, using first-match logic.

B. The default action, if a packet does not match any of the **access-list** commands, is to **deny** (discard) the packet.

Step 3. Enable the ACL on the chosen router interface, in the correct direction, using the **ip access-group** *number* {**in** | **out**} interface subcommand.

The rest of this section shows a couple of examples.

Standard Numbered ACL Example 1

The first example shows the configuration for the same requirements demonstrated with Figure 16-4 and Figure 16-5. Restated, the requirements for this ACL are as follows:

1. Enable the ACL inbound on R2's S0/0/1 interface.
2. Permit packets coming from host A.
3. Deny packets coming from other hosts in host A's subnet.
4. Permit packets coming from any other address in Class A network 10.0.0.0.
5. The original example made no comment about what to do by default, so simply deny all other traffic.

Example 16-1 shows a completed correct configuration, starting with the configuration process, followed by output from the **show running-config** command.

Example 16-1 *Standard Numbered ACL Example 1 Configuration*

```
R2# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)# access-list 1 permit 10.1.1.1
R2(config)# access-list 1 deny 10.1.1.0 0.0.0.255
R2(config)# access-list 1 permit 10.0.0.0 0.255.255.255
R2(config)# interface S0/0/1
R2(config-if)# ip access-group 1 in
R2(config-if)# ^Z
R2# show running-config
! Lines omitted for brevity

access-list 1 permit 10.1.1.1
access-list 1 deny 10.1.1.0 0.0.0.255
access-list 1 permit 10.0.0.0 0.255.255.255
```

First, pay close attention to the configuration process at the top of the example. Note that the **access-list** command does not change the command prompt from the global configuration mode prompt, because the **access-list** command is a global configuration command. Then, compare that to the output of the **show running-config** command: the details are identical compared to the commands that were added in configuration mode. Finally, make sure to note the **ip access-group 1 in** command, under R2's S0/0/1 interface, which enables the ACL logic (both location and direction).

Example 16-2 lists some output from Router R2 that shows information about this ACL. The **show ip access-lists** command lists details about IPv4 ACLs only, while the **show access-lists** command lists details about IPv4 ACLs plus any other types of ACLs that are currently configured, for example, IPv6 ACLs.

Example 16-2 *ACL* **show** *Commands on R2*

```
R2# show ip access-lists
Standard IP access list 1
    10 permit 10.1.1.1 (107 matches)
    20 deny   10.1.1.0, wildcard bits 0.0.0.255 (4 matches)
    30 permit 10.0.0.0, wildcard bits 0.255.255.255 (10 matches)
R2# show access-lists
Standard IP access list 1
    10 permit 10.1.1.1 (107 matches)
    20 deny   10.1.1.0, wildcard bits 0.0.0.255 (4 matches)
    30 permit 10.0.0.0, wildcard bits 0.255.255.255 (10 matches)
R2# show ip interface s0/0/1
Serial0/0/1 is up, line protocol is up
  Internet address is 10.1.2.2/24
  Broadcast address is 255.255.255.255
  Address determined by setup command
  MTU is 1500 bytes
  Helper address is not set
  Directed broadcast forwarding is disabled
  Multicast reserved groups joined: 224.0.0.9
  Outgoing access list is not set
  Inbound  access list is 1
! Lines omitted for brevity
```

The output of these commands shows two items of note. The first line of output in this case notes the type (standard), and the number. If more than one ACL existed, you would see multiple stanzas of output, one per ACL, each with a heading line like this one. Next, these commands list packet counts for the number of packets that the router has matched with each command. For example, 107 packets so far have matched the first line in the ACL.

Finally, the end of the example lists the **show ip interface** command output. This command lists, among many other items, the number or name of any IP ACL enabled on the interface per the **ip access-group** interface subcommand.

Standard Numbered ACL Example 2

For the second example, use Figure 16-8, and imagine your boss gave you some requirements hurriedly in the hall. At first, he tells you he wants to filter packets going from the servers on the right toward the clients on the left. Then, he says he wants you to allow access for hosts A, B, and other hosts in their same subnet to server S1, but deny access to that server to the hosts in host C's subnet. Then, he tells you that, additionally, hosts in host A's subnet should be denied access to server S2, but hosts in host C's subnet should be allowed access to server S2—all by filtering packets going right to left only. He then tells you to put the ACL inbound on R2's F0/0 interface.

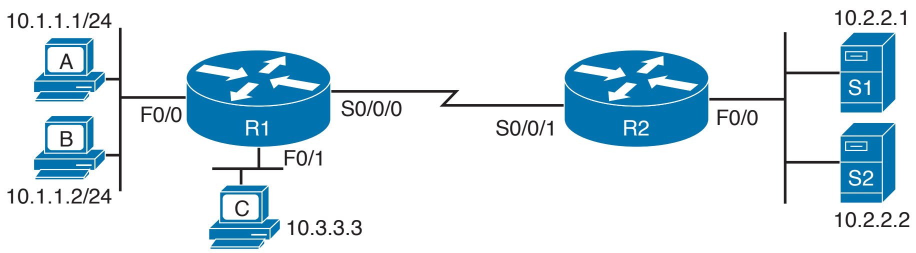


Figure 16-8 *Standard Numbered ACL Example 2*

If you cull through all the boss's comments, the requirements might reduce to the following:

1. Enable the ACL inbound on R2's F0/0 interface.
2. Permit packets from server S1 going to hosts in A's subnet.
3. Deny packets from server S1 going to hosts in C's subnet.
4. Permit packets from server S2 going to hosts in C's subnet.
5. Deny packets from server S2 going to hosts in A's subnet.
6. (There was no comment about what to do by default; use the implied **deny all** default.)

As it turns out, you cannot do everything your boss asked with a standard ACL. For example, consider the obvious command for requirement number 2: **access-list 2 permit 10.2.2.1**. That permits all traffic whose source IP is 10.2.2.1 (server S1). The very next requirement asks you to filter (deny) packets sourced from that same IP address! Even if you added another command that checked for source IP address 10.2.2.1, the router would never get to it, because routers use first-match logic when searching the ACL. You cannot check both the destination and source IP address, because standard ACLs cannot check the destination IP address.

To solve this problem, you should get a new boss! No, seriously, you have to rethink the problem and change the rules. In real life, you would probably use an extended ACL instead, which lets you check both the source and destination IP address.

For the sake of practicing another standard ACL, imagine your boss lets you change the requirements. First, you will use two outbound ACLs, both on Router R1. Each ACL will permit traffic from a single server to be forwarded onto that connected LAN, with the following modified requirements:

1. Using an outbound ACL on R1's F0/0 interface, permit packets from server S1, and deny all other packets.
2. Using an outbound ACL on R1's F0/1 interface, permit packets from server S2, and deny all other packets.

Example 16-3 shows the configuration that completes these requirements.

Example 16-3 *Alternative Configuration in Router R1*

```
access-list 2 remark This ACL permits server S1 traffic to host A's subnet
access-list 2 permit 10.2.2.1
!
access-list 3 remark This ACL permits server S2 traffic to host C's subnet
access-list 3 permit 10.2.2.2
!
interface F0/0
 ip access-group 2 out
!
interface F0/1
 ip access-group 3 out
```

As highlighted in the example, the solution with ACL number 2 permits all traffic from server S1, with that logic enabled for packets exiting R1's F0/0 interface. All other traffic will be discarded because of the implied deny all at the end of the ACL. In addition, ACL 3 permits traffic from server S2, which is then permitted to exit R1's F0/1 interface. Also, note that the solution shows the use of the **access-list remark** parameter, which allows you to leave text documentation that stays with the ACL.

NOTE When routers apply an ACL to filter packets in the outbound direction, as shown in Example 16-3, the router checks packets that it routes against the ACL. However, a router does not filter packets that the router itself creates with an outbound ACL. Examples of those packets include routing protocol messages, and packets sent by the **ping** and **traceroute** commands on that router.

Troubleshooting and Verification Tips

Troubleshooting IPv4 ACLs requires some attention to detail. In particular, you have to be ready to look at the address and wildcard mask and confidently predict the addresses matched by those two combined parameters. The upcoming practice problems a little later in this chapter can help prepare you for that part of the work. But a few other tips can help you verify and troubleshoot ACL problems on the exams as well.

First, you can tell if the router is matching packets or not with a couple of tools. Example 16-2 already showed that IOS keeps statistics about the packets matched by each line of an ACL. In addition, if you add the **log** keyword to the end of an **access-list** command, IOS then issues log messages with occasional statistics about matches of that particular line of the ACL. Both the statistics and the log messages can be helpful in deciding which line in the ACL is being matched by a packet.

For example, Example 16-4 shows an updated version of ACL 2 from Example 16-3, this time with the **log** keyword added. The bottom of the example then shows a typical log message, this one showing the resulting match based on a packet with source IP address 10.2.2.1 (as matched with the ACL), to destination address 10.1.1.1.

Example 16-4 *Creating Log Messages for ACL Statistics*

```
R1# show running-config
! lines removed for brevity
access-list 2 remark This ACL permits server S1 traffic to host A's subnet
access-list 2 permit 10.2.2.1 log
!
interface F0/0
 ip access-group 2 out

R1#
Feb  4 18:30:24.082: %SEC-6-IPACCESSLOGNP: list 2 permitted 0 10.2.2.1 -> 10.1.1.1, 1
  packet
```

Anytime you troubleshoot an ACL for the first time, before getting into the details of the matching logic, take the time to think about both the interface on which the ACL is enabled, and the direction of packet flow. Sometimes, the matching logic is perfect—but the ACL has been enabled on the wrong interface, or for the wrong direction, to match the packets as configured for the ACL.

For example, Figure 16-9 repeats the same ACL shown earlier in Figure 16-7. The first line of that ACL matches the specific host address 10.1.1.1. If that ACL exists on Router R2, placing that ACL as an inbound ACL on R2's S0/0/1 interface can work, because packets sent by host 10.1.1.1—on the left side of the figure—can enter R2's S0/0/1 interface. However, if R2 enables ACL 1 on its F0/0 interface, for inbound packets, the ACL will never match a packet with source IP address 10.1.1.1, because packets sent by host 10.1.1.1 will never enter that interface. Packets sent by 10.1.1.1 will exit R2's F0/0 interface, but never enter it, just because of the network topology.

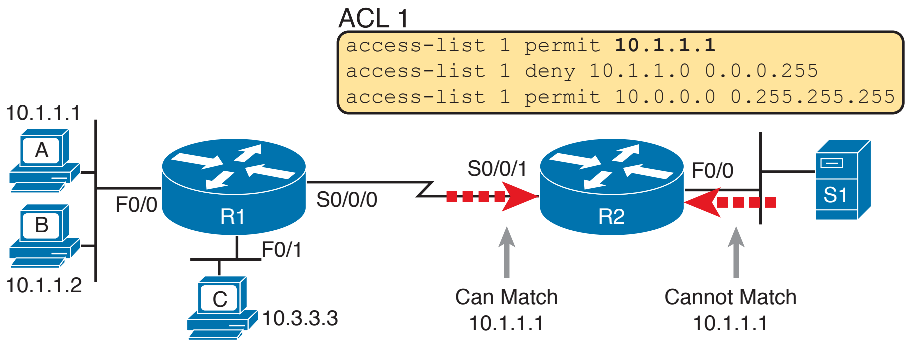

ACL 1
access-list 1 permit 10.1.1.1
access-list 1 deny 10.1.1.0 0.0.0.255
access-list 1 permit 10.0.0.0 0.255.255.255
10.1.1.1
A
B
10.1.1.2
F0/0
R1
S0/0/0
F0/1
C
10.3.3.3
S0/0/1
R2
F0/0
S1
Can Match 10.1.1.1
Cannot Match 10.1.1.1

Figure 16-9 *Example of Checking the Interface and Direction for an ACL*

Practice Applying Standard IP ACLs

Some CCENT and CCNA R&S topics, like ACLs, simply require more drills and practice than others. ACLs require you to think of parameters to match ranges of numbers, and that of course requires some use of math, and some use of processes.

This section provides some practice problems and tips, from two perspectives. First, this section asks you to build one-line standard ACLs to match some packets. Second, this section asks you to interpret existing ACL commands to describe what packets the ACL will match. Both skills are useful for the exams.

Practice Building access-list Commands

In this section, practice getting comfortable with the syntax of the **access-list** command, particularly with choosing the correct matching logic. These skills will be helpful when reading about extended and named ACLs in the next chapter.

First, the following list summarizes some important tips to consider when choosing matching parameters to any **access-list** command:

Key Topic

- To match a specific address, just list the address.
- To match any and all addresses, use the **any** keyword.
- To match based only on the first one, two, or three octets of an address, use the 0.255.255.255, 0.0.255.255, and 0.0.0.255 WC masks, respectively. Also, make the source (address) parameter have 0s in the wildcard octets (those octets with 255 in the wildcard mask).
- To match a subnet, use the subnet ID as the source, and find the WC mask by subtracting the DDN subnet mask from 255.255.255.255.

Table 16-2 lists the criteria for several practice problems. Your job: Create a one-line standard ACL that matches the packets. The answers are listed in the section "Answers to Earlier Practice Problems," later in this chapter.

Table 16-2 Building One-Line Standard ACLs: Practice

Problem	Criteria
1	Packets from 172.16.5.4
2	Packets from hosts with 192.168.6 as the first three octets
3	Packets from hosts with 192.168 as the first two octets
4	Packets from any host
5	Packets from subnet 10.1.200.0/21
6	Packets from subnet 10.1.200.0/27
7	Packets from subnet 172.20.112.0/23
8	Packets from subnet 172.20.112.0/26
9	Packets from subnet 192.168.9.64/28
10	Packets from subnet 192.168.9.64/30

Reverse Engineering from ACL to Address Range

In some cases, you may not be creating your own ACL. Instead, you may need to interpret some existing **access-list** commands. To answer these types of questions on the exams, you need to determine the range of IP addresses matched by a particular address/wildcard mask combination in each ACL statement.

Under certain assumptions that are reasonable for CCENT and CCNA R&S certifications, calculating the range of addresses matched by an ACL can be relatively simple. Basically, the range of addresses begins with the address configured in the ACL command. The range of addresses ends with the sum of the address field and the wildcard mask. That's it.

For example, with the command **access-list 1 permit 172.16.200.0 0.0.7.255**, the low end of the range is simply 172.16.200.0, taken directly from the command itself. Then, to find the high end of the range, just add this number to the WC mask, as follows:

$$\begin{array}{r} 172.16.200.0 \\ +\ 0.\ \ 0.\ \ \ \ 7.255 \\ \hline 172.16.207.255 \end{array}$$

For this last bit of practice, look at the existing **access-list** commands in Table 16-3. In each case, make a notation about the exact IP address, or range of IP addresses, matched by the command.

Table 16-3 Finding IP Addresses/Ranges Matching by Existing ACLs

Problem	Commands for Which to Predict the Source Address Range
1	**access-list 1 permit 10.7.6.5**
2	**access-list 2 permit 192.168.4.0 0.0.0.127**
3	**access-list 3 permit 192.168.6.0 0.0.0.31**
4	**access-list 4 permit 172.30.96.0 0.0.3.255**
5	**access-list 5 permit 172.30.96.0 0.0.0.63**
6	**access-list 6 permit 10.1.192.0 0.0.0.31**
7	**access-list 7 permit 10.1.192.0 0.0.1.255**
8	**access-list 8 permit 10.1.192.0 0.0.63.255**

Interestingly, IOS lets the CLI user type an **access-list** command in configuration mode, and IOS will potentially change the address parameter before placing the command into the running-config file. This process of just finding the range of addresses matched by the **access-list** command expects that the **access-list** command came from the router, so that any such changes were complete.

The change IOS can make with an **access-list** command is to convert to 0 any octet of an address for which the wildcard mask's octet is 255. For example, with a wildcard mask of 0.0.255.255, IOS ignores the last two octets. IOS expects the address field to end with two 0s. If not, IOS still accepts the **access-list** command, but IOS changes the last two octets of the address to 0s. Example 16-5 shows an example, where the configuration shows address 10.1.1.1, but wildcard mask 0.0.255.255.

Example 16-5 *IOS Changing the Address Field in an* **access-list** *Command*

```
R2# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)# access-list 21 permit 10.1.1.1 0.0.255.255
R2(config)# ^Z
R2#
R2# show ip access-lists
Standard IP access list 21
    10 permit 10.1.0.0, wildcard bits 0.0.255.255
```

The math to find the range of addresses relies on the fact that either the command is fully correct, or that IOS has already set these address octets to 0, as shown in the example.

NOTE The most useful WC masks, in binary, do not interleave 0s and 1s. This book assumes the use of only these types of WC masks. However, Cisco IOS allows WC masks that interleave 0s and 1s, but using these WC masks breaks the simple method of calculating the range of addresses. As you progress through to CCIE studies, be ready to dig deeper to learn how to determine what an ACL matches.

Chapter Review

One key to doing well on the exams is to perform repetitive spaced review sessions. Review this chapter's material using either the tools in the book, DVD, or interactive tools for the same material found on the book's companion website. Refer to the "Your Study Plan" element for more details. Table 16-4 outlines the key review elements and where you can find them. To better track your study progress, record when you completed these activities in the second column.

Table 16-4 Chapter Review Tracking

Review Element	Review Date(s)	Resource Used
Review key topics		Book, DVD/website
Review key terms		Book, DVD/website
Repeat DIKTA questions		Book, PCPT
Review command tables		Book

Review All the Key Topics

Key Topic

Table 16-5 Key Topics for Chapter 16

Key Topic Element	Description	Page Number
Paragraph	Summary of the general rule of the location and direction for an ACL	441
Figure 16-3	Summary of four main categories of IPv4 ACLs in Cisco IOS	443
Paragraph	Summary of first-match logic used by all ACLs	444
List	Wildcard mask logic for decimal 0 and 255	446
List	Wildcard mask logic to match a subnet	448
List	Steps to plan and implement a standard IP ACL	449
List	Tips for creating matching logic for the source address field in the **access-list** command	454

Key Terms You Should Know

standard access list, wildcard mask

Additional Practice for This Chapter's Processes

For additional practice with analyzing subnets, you may do the same set of practice problems using your choice of tools:

Application: Use the Basic IPv4 Access Control Lists application on the DVD or companion website.

PDF: Alternatively, practice the same problems found in these apps using DVD Appendix D, "Practice for Chapter 16: Basic IPv4 Access Control Lists."

Command References

Tables 16-6 and 16-7 list configuration and verification commands used in this chapter. As an easy review exercise, cover the left column in a table, read the right column, and try to recall the command without looking. Then repeat the exercise, covering the right column, and try to recall what the command does.

Table 16-6 Chapter 16 Configuration Command Reference

Command	Description
access-list *access-list-number* {**deny** | **permit**} *source* [*source-wildcard*] [**log**]	Global command for standard numbered access lists. Use a number between 1 and 99 or 1300 and 1999, inclusive.
access-list *access-list-number* **remark** *text*	Defines a remark that helps you remember what the ACL is supposed to do
ip access-group *number* {**in** | **out**}	Interface subcommand to enable access lists

Table 16-7 Chapter 16 EXEC Command Reference

Command	Description
show ip interface [*type number*]	Includes a reference to the access lists enabled on the interface
show access-lists [*access-list-number* | *access-list-name*]	Shows details of configured access lists for all protocols
show ip access-lists [*access-list-number* | *access-list-name*]	Shows IP access lists

Answers to Earlier Practice Problems

Table 16-8 lists the answers to the problems listed earlier in Table 16-2.

Table 16-8 Building One-Line Standard ACLs: Answers

Problem	Answers
1	**access-list 1 permit 172.16.5.4**
2	**access-list 2 permit 192.168.6.0 0.0.0.255**
3	**access-list 3 permit 192.168.0.0 0.0.255.255**
4	**access-list 4 permit any**
5	**access-list 5 permit 10.1.200.0 0.0.7.255**
6	**access-list 6 permit 10.1.200.0 0.0.0.31**
7	**access-list 7 permit 172.20.112.0 0.0.1.255**
8	**access-list 8 permit 172.20.112.0 0.0.0.63**
9	**access-list 9 permit 192.168.9.64 0.0.0.15**
10	**access-list 10 permit 192.168.9.64 0.0.0.3**

Table 16-9 lists the answers to the problems listed earlier in Table 16-3.

Table 16-9 Address Ranges for Problems in Table 16-3: Answers

Problem	Address Range
1	One address: 10.7.6.5
2	192.168.4.0 – 192.168.4.127
3	192.168.6.0 – 192.168.6.31
4	172.30.96.0 – 172.30.99.255
5	172.30.96.0 – 172.30.96.63
6	10.1.192.0 – 10.1.192.31
7	10.1.192.0 – 10.1.193.255
8	10.1.192.0 – 10.1.255.255

This page intentionally left blank

CHAPTER 17

Advanced IPv4 Access Control Lists

This chapter covers the following exam topics:

4.0 Infrastructure Services

4.4 Configure, verify, and troubleshoot IPv4 and IPv6 access list for traffic filtering

4.1.a Standard

4.1.b Extended

4.1.c Named

IPv4 ACLs are either standard or extended ACLs, with standard ACLs matching only the source IP address, and extended matching a variety of packet header fields. At the same time, IP ACLs are either numbered or named. Figure 17-1 shows the categories, and the main features of each, as introduced in the previous chapter.

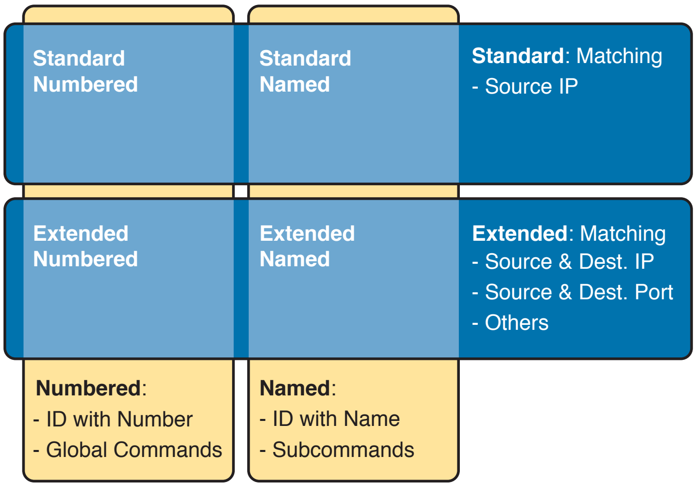


Figure 17-1 *Comparisons of IP ACL Types*

This chapter discusses the other three categories of ACLs beyond standard numbered IP ACLs, and ends with a few miscellaneous features to secure Cisco routers and switches. Note that this chapter's technical content is identical to the ICND1 100-105 Cert Guide's Chapter 26; use that fact to speed your reading now if you have already read that chapter.

“Do I Know This Already?” Quiz

Take the quiz (either here, or use the PCPT software) if you want to use the score to help you decide how much time to spend on this chapter. The answers are at the bottom of the page following the quiz, and the explanations are in DVD Appendix C and in the PCPT software.

Table 17-1 “Do I Know This Already?” Foundation Topics Section-to-Question Mapping

Foundation Topics Section	Questions
Extended IP Access Control Lists	1–3
Named ACLs and ACL Editing	4
Troubleshooting with IPv4 ACLs	5–6

1. Which of the following fields cannot be compared based on an extended IP ACL? (Choose two answers.)
   - **a.** Protocol
   - **b.** Source IP address
   - **c.** Destination IP address
   - **d.** TOS byte
   - **e.** URL
   - **f.** Filename for FTP transfers
2. Which of the following **access-list** commands permit packets going from host 10.1.1.1 to all web servers whose IP addresses begin with 172.16.5? (Choose two answers.)
   - **a.** **access-list 101 permit tcp host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**
   - **b.** **access-list 1951 permit ip host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**
   - **c.** **access-list 2523 permit ip host 10.1.1.1 eq www 172.16.5.0 0.0.0.255**
   - **d.** **access-list 2523 permit tcp host 10.1.1.1 eq www 172.16.5.0 0.0.0.255**
   - **e.** **access-list 2523 permit tcp host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**
3. Which of the following **access-list** commands permits packets going to any web client from all web servers whose IP addresses begin with 172.16.5?
   - **a.** **access-list 101 permit tcp host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**
   - **b.** **access-list 1951 permit ip host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**
   - **c.** **access-list 2523 permit tcp any eq www 172.16.5.0 0.0.0.255**
   - **d.** **access-list 2523 permit tcp 172.16.5.0 0.0.0.255 eq www 172.16.5.0 0.0.0.255**
   - **e.** **access-list 2523 permit tcp 172.16.5.0 0.0.0.255 eq www any**

4. In a router running a recent IOS version (at least version 15.0), an engineer needs to delete the second line in ACL 101, which currently has four commands configured. Which of the following options could be used? (Choose two answers.)

  a. Delete the entire ACL and reconfigure the three ACL statements that should remain in the ACL.

  b. Delete one line from the ACL using the **no access-list...** global command.

  c. Delete one line from the ACL by entering ACL configuration mode for the ACL and then deleting only the second line based on its sequence number.

  d. Delete the last three lines from the ACL from global configuration mode, and then add the last two statements back into the ACL.

5. An engineer is considering configuring an ACL on Router R1. The engineer could use ACL A, which would be enabled with the **ip access-group A out** command on interface G0/1, or ACL B, which would be enabled with the **ip access-group B in** command on that same interface. R1's G0/1 interface uses IPv4 address 1.1.1.1. Which of the answers are true when comparing these options? (Choose two answers.)

  a. ACL A creates more risk of filtering important overhead traffic than ACL B.

  b. ACL B creates more risk of filtering important overhead traffic than ACL A.

  c. A **ping 1.1.1.1** command on R1 would bypass ACL A even if enabled.

  d. A **ping 1.1.1.1** command on R1 would bypass ACL B even if enabled.

6. An engineer configures an ACL but forgets to save the configuration. At that point, which of the following commands display the configuration of an IPv4 ACL, including line numbers? (Choose two answers.)

  a. **show running-config**

  b. **show startup-config**

  c. **show ip access-lists**

  d. **show access-lists**

Foundation Topics

Extended Numbered IP Access Control Lists

Extended IP access lists have many similarities compared to the standard numbered IP ACLs discussed in the previous chapter. Just like standard IP ACLs, you enable extended access lists on interfaces for packets either entering or exiting the interface. IOS searches the list sequentially. Extended ACLs also use first-match logic, because the router stops the search through the list as soon as the first statement is matched, taking the action defined in the first-matched statement. All these features are also true of standard numbered access lists (and named ACLs).

Extended ACLs differ from standard ACLs mostly because of the larger variety of packet header fields that can be used to match a packet. One extended ACL statement can examine multiple parts of the packet headers, requiring that all the parameters be matched correctly to match that one ACL statement. That powerful matching logic makes extended access lists both more useful and more complex than standard IP ACLs.

Matching the Protocol, Source IP, and Destination IP

Like standard numbered IP ACLs, extended numbered IP ACLs also use the **access-list** global command. The syntax is identical, at least up through the **permit** or **deny** keyword. At that point, the command lists matching parameters, and those differ, of course. In particular, the extended ACL **access-list** command requires three matching parameters: the IP protocol type, the source IP address, and the destination IP address.

The IP header's Protocol field identifies the header that follows the IP header. Figure 17-2 shows the location of the IP Protocol field, the concept of it pointing to the type of header that follows, along with some details of the IP header for reference.

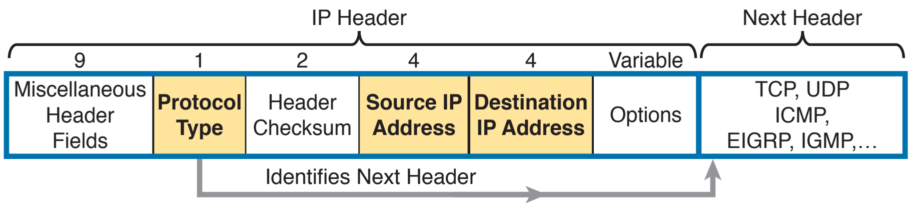


Figure 17-2 *IP Header, with Focus on Required Fields in Extended IP ACLs*

IOS requires that you configure parameters for the three highlighted parts of Figure 17-2. For the protocol type, you simply use a keyword, such as **tcp**, **udp**, or **icmp**, matching IP packets that happen to have a TCP, UDP, or ICMP header, respectively, following the IP header. Or you can use the keyword **ip**, which means "all IPv4 packets." You also must configure some values for the source and destination IP address fields that follow; these fields use the same syntax and options for matching the IP addresses as discussed in Chapter 16, "Basic IPv4 Access Control Lists." Figure 17-3 shows the syntax.

Key Topic

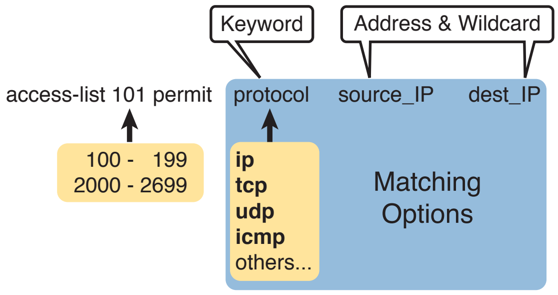


Figure 17-3 *Extended ACL Syntax, with Required Fields*

NOTE When matching IP addresses in the source and destination fields, there is one difference with standard ACLs: When matching a specific IP address, the extended ACL requires the use of the **host** keyword. You cannot simply list the IP address alone.

Answers to the "Do I Know This Already?" quiz:

1 E, F **2** A, E **3** E **4** A, C **5** B, C **6** C, D

Table 17-2 lists several sample **access-list** commands that use only the required matching parameters. Feel free to cover the right side and use the table for an exercise, or just review the explanations to get an idea for the logic in some sample commands.

Table 17-2 Extended **access-list** Commands and Logic Explanations

access-list Statement	What It Matches
access-list 101 deny tcp any any	Any IP packet that has a TCP header
access-list 101 deny udp any any	Any IP packet that has a UDP header
access-list 101 deny icmp any any	Any IP packet that has an ICMP header
access-list 101 deny ip host 1.1.1.1 host 2.2.2.2	All IP packets from host 1.1.1.1 going to host 2.2.2.2, regardless of the header after the IP header
access-list 101 deny udp 1.1.1.0 0.0.0.255 any	All IP packets that have a UDP header following the IP header, from subnet 1.1.1.0/24, and going to any destination

The last entry in Table 17-2 helps make an important point about how IOS processes extended ACLs:

Key Topic

In an extended ACL **access-list** command, all the matching parameters must match the packet for the packet to match the command.

For example, in that last example from Table 17-2, the command checks for UDP, a source IP address from subnet 1.1.1.0/24, and any destination IP address. If a packet with source IP address 1.1.1.1 were examined, it would match the source IP address check, but if it had a TCP header instead of UDP, it would not match this **access-list** command. All parameters must match.

Matching TCP and UDP Port Numbers

Extended ACLs can also examine parts of the TCP and UDP headers, particularly the source and destination port number fields. The port numbers identify the application that sends or receives the data.

The most useful ports to check are the well-known ports used by servers. For example, web servers use well-known port 80 by default. Figure 17-4 shows the location of the port numbers in the TCP header, following the IP header.

Key Topic

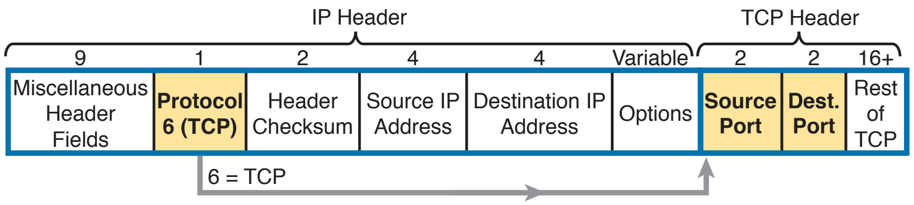


Figure 17-4 *IP Header, Followed by a TCP Header and Port Number Fields*

When an extended ACL command includes either the **tcp** or **udp** keyword, that command can optionally reference the source and/or destination port. To make these comparisons, the syntax uses keywords for equal, not equal, less than, greater than, and for a range of port numbers. In addition, the command can use either the literal decimal port numbers, or more convenient keywords for some well-known application ports. Figure 17-5 shows the positions of the source and destination port fields in the **access-list** command and these port number keywords.

Key Topic

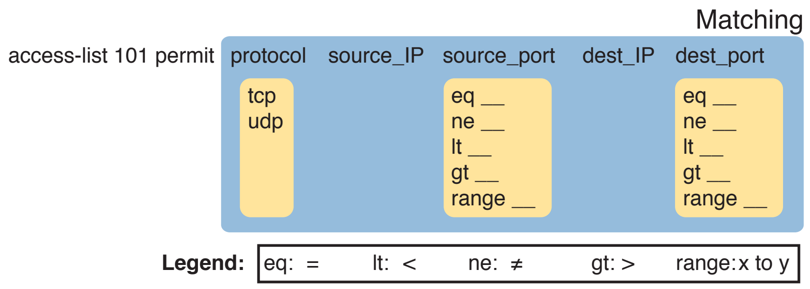


Figure 17-5 *Extended ACL Syntax with TCP and UDP Port Numbers Enabled*

For example, consider the simple network shown in Figure 17-6. The FTP server sits on the right, with the client on the left. The figure shows the syntax of an ACL that matches the following:

- Packets that include a TCP header
- Packets sent from the client subnet
- Packets sent to the server subnet
- Packets with TCP destination port 21 (FTP server control port)

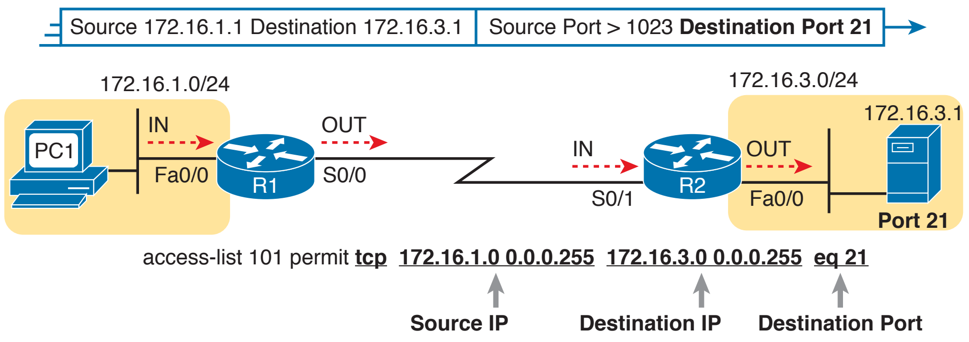
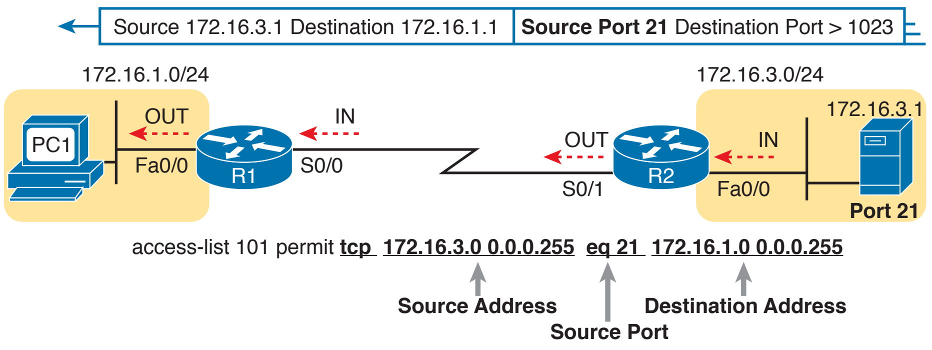

Source 172.16.1.1 Destination 172.16.3.1
Source Port > 1023 Destination Port 21
172.16.1.0/24
172.16.3.0/24
172.16.3.1
PC1
IN
Fa0/0
R1
OUT
S0/0
IN
S0/1
R2
OUT
Fa0/0
Port 21
access-list 101 permit tcp 172.16.1.0 0.0.0.255 172.16.3.0 0.0.0.255 eq 21
Source IP
Destination IP
Destination Port

Figure 17-6 *Filtering Packets Based on Destination Port*

To fully appreciate the matching of the destination port with the **eq 21** parameters, consider packets moving from left to right, from PC1 to the server. Assuming the server uses well-known port 21 (FTP control port), the packet's TCP header has a destination port value of 21. The ACL syntax includes the **eq 21** parameters after the destination IP address. The position after the destination address parameters is important: That position identifies the fact that the **eq 21** parameters should be compared to the packet's destination port. As a result, the ACL statement shown in Figure 17-6 would match this packet, and the destination port of 21, if used in any of the four locations implied by the four dashed arrowed lines in the figure.

Conversely, Figure 17-7 shows the reverse flow, with a packet sent by the server back toward PC1. In this case, the packet's TCP header has a source port of 21, so the ACL must check the source port value of 21, and the ACL must be located on different interfaces. In this case, the **eq 21** parameters follow the source address field, but come before the destination address field.

Key Topic



```
access-list 101 permit tcp 172.16.3.0 0.0.0.255 eq 21 172.16.1.0 0.0.0.255
```

Figure 17-7 *Filtering Packets Based on Source Port*

When examining ACLs that match port numbers, first consider the location and direction in which the ACL will be applied. That direction determines whether the packet is being sent to the server, or from the server. At that point, you can decide whether you need to check the source or destination port in the packet. For reference, Table 17-3 lists many of the popular port numbers and their transport layer protocols and applications. Note that the syntax of the **access-list** commands accepts both the port numbers and a shorthand version of the application name.

Table 17-3 Popular Applications and Their Well-Known Port Numbers

Port Number(s)	Protocol	Application	access-list Command Keyword
20	TCP	FTP data	**ftp-data**
21	TCP	FTP control	**ftp**
22	TCP	SSH	—
23	TCP	Telnet	**telnet**
25	TCP	SMTP	**smtp**
53	UDP, TCP	DNS	**domain**
67	UDP	DHCP Server	**bootps**
68	UDP	DHCP Client	**bootpc**
69	UDP	TFTP	**tftp**
80	TCP	HTTP (WWW)	**www**
110	TCP	POP3	**pop3**
161	UDP	SNMP	**snmp**
443	TCP	SSL	—
514	UDP	Syslog	—
16,384 – 32,767	UDP	RTP (voice, video)	—

Table 17-4 lists several example **access-list** commands that match based on port numbers. Cover the right side of the table, and try to characterize the packets matched by each command. Then, check the right side of the table to see if you agree with the assessment.

Table 17-4 Extended **access-list** Command Examples and Logic Explanations

access-list Statement	What It Matches
access-list 101 deny tcp any gt 1023 host 10.1.1.1 eq 23	Packets with a TCP header, any source IP address, with a source port greater than (gt) 1023, a destination IP address of exactly 10.1.1.1, and a destination port equal to (eq) 23.
access-list 101 deny tcp any host 10.1.1.1 eq 23	The same as the preceding example, but any source port matches, because that parameter is omitted in this case.
access-list 101 deny tcp any host 10.1.1.1 eq telnet	The same as the preceding example. The **telnet** keyword is used instead of port 23.
access-list 101 deny udp 1.0.0.0 0.255.255.255 lt 1023 any	A packet with a source in network 1.0.0.0/8, using UDP with a source port less than (lt) 1023, with any destination IP address.

Extended IP ACL Configuration

Because extended ACLs can match so many different fields in the various headers in an IP packet, the command syntax cannot be easily summarized in a single generic command. However, the two commands in Table 17-5 summarize the syntax options as covered in this book.

Table 17-5 Extended IP Access List Configuration Commands

Command	Configuration Mode and Description
access-list *access-list-number* {**deny** | **permit**} *protocol source source-wildcard destination destination-wildcard* [**log** | **log-input**]	Global command for extended numbered **access lists**. Use a number between 100 and 199 or 2000 and 2699, inclusive.
access-list *access-list-number* {**deny** | **permit**} {**tcp** | **udp**} *source source-wildcard* [*operator* [*port*]] *destination destination-wildcard* [*operator* [*port*]] [**established**] [**log**]	A version of the **access-list** command with parameters specific to TCP and/or UDP.

The configuration process for extended ACLs mostly matches the same process used for standard ACLs. You must choose the location and direction in which to enable the ACL, particularly the direction, so that you can characterize whether certain addresses and ports will be either the source or destination. Configure the ACL using **access-list** commands, and when complete, then enable the ACL using the same **ip access-group** command used with standard ACLs. All these steps mirror what you do with standard ACLs; however, when configuring, keep the following differences in mind:

Key Topic

- Place extended ACLs as close as possible to the source of the packets that will be filtered. Filtering close to the source of the packets saves some bandwidth.

- Remember that all fields in one **access-list** command must match a packet for the packet to be considered to match that **access-list** statement.
- Use numbers of 100–199 and 2000–2699 on the **access-list** commands; no one number is inherently better than another.

Extended IP Access Lists: Example 1

This example focuses on understanding basic syntax. In this case, the ACL denies Bob access to all FTP servers on R1's Ethernet, and it denies Larry access to server1's web server. Figure 17-8 shows the network topology; Example 17-1 shows the configuration on R1.

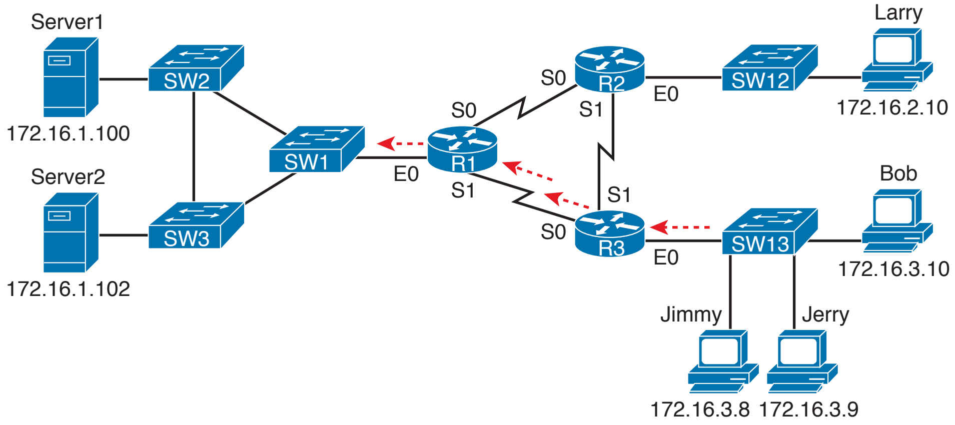


Figure 17-8 *Network Diagram for Extended Access List Example 1*

Example 17-1 *R1's Extended Access List: Example 1*

```
interface Serial0
 ip address 172.16.12.1 255.255.255.0
 ip access-group 101 in
!
interface Serial1
 ip address 172.16.13.1 255.255.255.0
 ip access-group 101 in
!
access-list 101 remark Stop Bob to FTP servers, and Larry to Server1 web
access-list 101 deny tcp host 172.16.3.10 172.16.1.0 0.0.0.255 eq ftp
access-list 101 deny tcp host 172.16.2.10 host 172.16.1.100 eq www
access-list 101 permit ip any any
```

The first ACL statement prevents Bob's access to FTP servers in subnet 172.16.1.0. The second statement prevents Larry's access to web services on Server1. The final statement permits all other traffic.

Focusing on the syntax for a moment, there are several new items to review. First, the access-list number for extended access lists falls in the range of 100 to 199 or 2000 to 2699. Following the **permit** or **deny** action, the *protocol* parameter defines whether you want to check for all IP packets or specific headers, such as TCP or UDP headers. When you check for TCP or UDP port numbers, you must specify the TCP or UDP protocol. Both FTP and the web use TCP.

This example uses the **eq** parameter, meaning "equals," to check the destination port numbers for FTP control (keyword **ftp**) and HTTP traffic (keyword **www**). You can use the numeric values—or, for the more popular options, a more obvious text version is valid. (If you were to type **eq 80**, the config would show **eq www**.)

This example enables the ACL in two places on R1: inbound on each serial interface. These locations achieve the goal of the ACL. However, that initial placement was made to make the point that Cisco suggests that you locate them as close as possible to the source of the packet. Therefore, Example 17-2 achieves the same goal as Example 17-1 of stopping Bob's access to FTP servers at the main site, and it does so with an ACL on R3.

Example 17-2 *R3's Extended Access List Stopping Bob from Reaching FTP Servers Near R1*

```
interface Ethernet0
 ip address 172.16.3.1 255.255.255.0
 ip access-group 103 in

access-list 103 remark deny Bob to FTP servers in subnet 172.16.1.0/24
access-list 103 deny tcp host 172.16.3.10 172.16.1.0 0.0.0.255 eq ftp
access-list 103 permit ip any any
```

The new configuration on R3 meets the goals to filter Bob's traffic, while also meeting the overarching design goal of keeping the ACL close to the source of the packets. ACL 103 on R3 looks a lot like ACL 101 on R1 from Example 17-1, but this time, the ACL does not bother to check for the criteria to match Larry's traffic, because Larry's traffic will never enter R3's Ethernet 0 interface. ACL 103 filters Bob's FTP traffic to destinations in subnet 172.16.1.0/24, with all other traffic entering R3's E0 interface making it into the network.

Extended IP Access Lists: Example 2

Example 17-3, based on the network shown in Figure 17-9, shows another example of how to use extended IP access lists. This example uses the following criteria:

- Sam is not allowed access to the subnet of Bugs or Daffy.
- Hosts on the Seville Ethernet are not allowed access to hosts on the Yosemite Ethernet.
- All other combinations are allowed.

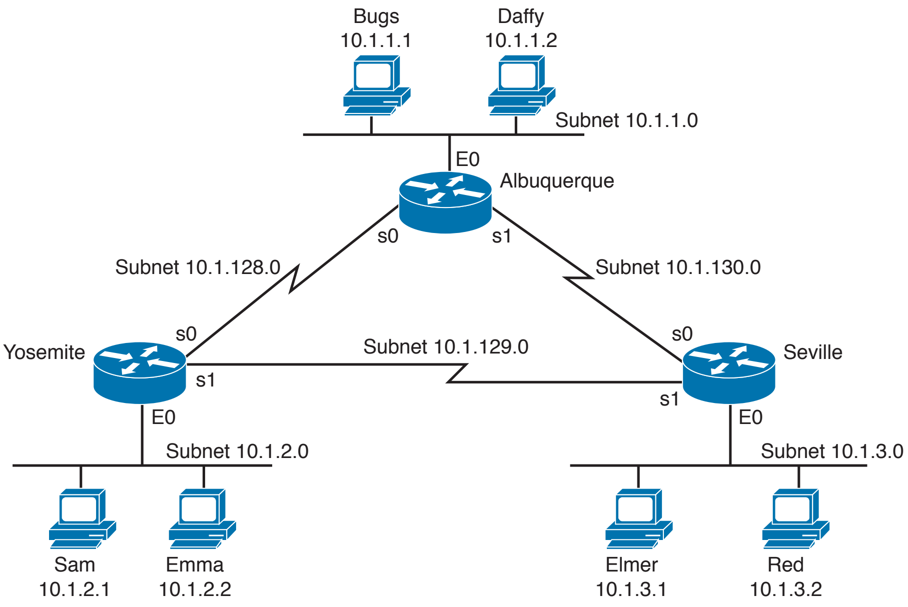


Figure 17-9 *Network Diagram for Extended Access List Example 2*

Example 17-3 *Yosemite Configuration for Extended Access List Example*

```
interface ethernet 0
 ip access-group 110 in
!
access-list 110 deny ip host 10.1.2.1 10.1.1.0 0.0.0.255
access-list 110 deny ip 10.1.2.0 0.0.0.255 10.1.3.0 0.0.0.255
access-list 110 permit ip any any
```

This configuration solves the problem with few statements while keeping to the Cisco design guideline of placing extended ACLs as close as possible to the source of the traffic. The ACL filters packets that enter Yosemite's E0 interface, which is the first router interface that packets sent by Sam enter. If the route between Yosemite and the other subnets changes over time, the ACL still applies. Also, the filtering mandated by the second requirement (to disallow Seville's LAN hosts from accessing Yosemite's) is met by the second **access-list** statement. Stopping packet flow from Yosemite's LAN subnet to Seville's LAN subnet stops effective communication between the two subnets. Alternatively, the opposite logic could have been configured at Seville.

Practice Building access-list Commands

Table 17-6 supplies a practice exercise to help you get comfortable with the syntax of the extended **access-list** command, particularly with choosing the correct matching logic. Your job: create a one-line extended ACL that matches the packets. The answers are in the section "Answers to Earlier Practice Problems," later in this chapter. Note that if the criteria mentions a particular application protocol, for example, "web client," that means to specifically match for that application protocol.

Table 17-6 Building One-Line Extended ACLs: Practice

Problem	Criteria
1	From web client 10.1.1.1, sent to a web server in subnet 10.1.2.0/24.
2	From Telnet client 172.16.4.3/25, sent to a Telnet server in subnet 172.16.3.0/25. Match all hosts in the client's subnet as well.
3	ICMP messages from the subnet in which 192.168.7.200/26 resides to all hosts in the subnet where 192.168.7.14/29 resides.
4	From web server 10.2.3.4/23's subnet to clients in the same subnet as host 10.4.5.6/22.
5	From Telnet server 172.20.1.0/24's subnet, sent to any host in the same subnet as host 172.20.44.1/23.
6	From web client 192.168.99.99/28, sent to a web server in subnet 192.168.176.0/28. Match all hosts in the client's subnet as well.
7	ICMP messages from the subnet in which 10.55.66.77/25 resides to all hosts in the subnet where 10.66.55.44/26 resides.
8	Any and every IPv4 packet.

Named ACLs and ACL Editing

Now that you have a good understanding of the core concepts in IOS IP ACLs, this section examines a few enhancements to IOS support for ACLs: named ACLs and ACL editing with sequence numbers. Although both features are useful and important, neither adds any function as to what a router can and cannot filter. Instead, named ACLs and ACL sequence numbers make it easier to remember ACL names and edit existing ACLs when an ACL needs to change.

Named IP Access Lists

Named IP ACLs have many similarities with numbered IP ACLs. They can be used for filtering packets, plus for many other purposes. They can match the same fields as well: Standard numbered ACLs can match the same fields as a standard named ACL, and extended numbered ACLs can match the same fields as an extended named ACL.

Of course, there are differences between named and numbered ACLs. Named ACLs originally had three big differences compared to numbered ACLs:

Key Topic

- Using names instead of numbers to identify the ACL, making it easier to remember the reason for the ACL
- Using ACL subcommands, not global commands, to define the action and matching parameters
- Using ACL editing features that allow the CLI user to delete individual lines from the ACL and insert new lines

You can easily learn named ACL configuration by just converting numbered ACLs to use the equivalent named ACL configuration. Figure 17-10 shows just such a conversion, using a simple three-line standard ACL number 1. To create the three **permit** subcommands for the named ACL, you literally copy parts of the three numbered ACL commands, beginning with the **permit** keyword.

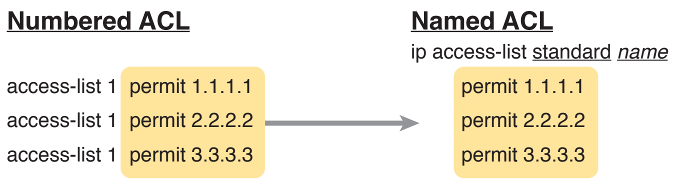

Numbered ACL
access-list 1 permit 1.1.1.1
access-list 1 permit 2.2.2.2
access-list 1 permit 3.3.3.3
Named ACL
ip access-list standard name
permit 1.1.1.1
permit 2.2.2.2
permit 3.3.3.3

Figure 17-10 *Named ACL Versus Numbered ACL Configuration*

The only truly new part of the named ACL configuration is the **ip access-list** global configuration command. This command defines whether an ACL is a standard or extended ACL, and defines the name. It also moves the user to ACL configuration mode, as shown in upcoming Example 17-4. Once in ACL configuration mode, you configure **permit**, **deny**, and **remark** commands that mirror the syntax of numbered ACL **access-list** commands. If you're configuring a standard named ACL, these commands match the syntax of standard numbered ACLs; if you're configuring extended named ACLs, they match the syntax of extended numbered ACLs.

Example 17-4 shows the configuration of a named extended ACL. Pay particular attention to the configuration mode prompts, which show ACL configuration mode.

Example 17-4 *Named Access List Configuration*

```
Router# configure terminal
Enter configuration commands, one per line.  End with Ctrl-Z.
Router(config)# ip access-list extended barney
Router(config-ext-nacl)# permit tcp host 10.1.1.2 eq www any
Router(config-ext-nacl)# deny udp host 10.1.1.1 10.1.2.0 0.0.0.255
Router(config-ext-nacl)# deny ip 10.1.3.0 0.0.0.255 10.1.2.0 0.0.0.255
Router(config-ext-nacl)# deny ip 10.1.2.0 0.0.0.255 10.2.3.0 0.0.0.255
Router(config-ext-nacl)# permit ip any any
Router(config-ext-nacl)# interface serial1
Router(config-if)# ip access-group barney out
Router(config-if)# ^Z
Router# show running-config
Building configuration...

Current configuration:

! lines omitted for brevity

interface serial 1
 ip access-group barney out
!
ip access-list extended barney
 permit tcp host 10.1.1.2 eq www any
 deny   udp host 10.1.1.1 10.1.2.0 0.0.0.255
 deny   ip 10.1.3.0 0.0.0.255 10.1.2.0 0.0.0.255
 deny   ip 10.1.2.0 0.0.0.255 10.2.3.0 0.0.0.255
 permit ip any any
```

Example 17-4 begins with the creation of an ACL named barney. The **ip access-list extended barney** command creates the ACL, naming it barney and placing the user in ACL configuration mode. This command also tells the IOS that barney is an extended ACL. Next, five different **permit** and **deny** statements define the matching logic and action to be taken upon a match. The **show running-config** command output lists the named ACL configuration before the single entry is deleted.

Named ACLs allow the user to delete and add new lines to the ACL from within ACL configuration mode. Example 17-5 shows how, with the **no deny ip . . .** command deleting a single entry from the ACL. Notice that the output of the **show access-list** command at the end of the example still lists the ACL, with four **permit** and **deny** commands instead of five.

Example 17-5 *Removing One Command from a Named ACL*

```
Router# configure terminal
Enter configuration commands, one per line.  End with Ctrl-Z.
Router(config)# ip access-list extended barney
Router(config-ext-nacl)# no deny ip 10.1.2.0 0.0.0.255 10.2.3.0 0.0.0.255
Router(config-ext-nacl)# ^Z
Router# show access-list

Extended IP access list barney
    10 permit tcp host 10.1.1.2 eq www any
    20 deny   udp host 10.1.1.1 10.1.2.0 0.0.0.255
    30 deny   ip 10.1.3.0 0.0.0.255 10.1.2.0 0.0.0.255
    50 permit ip any any
```

Editing ACLs Using Sequence Numbers

Numbered ACLs have existed in IOS since the early days of Cisco routers and IOS; however, for many years, through many IOS versions, the ability to edit a numbered IP ACL was poor. For example, to simply delete a line from the ACL, the user had to delete the entire ACL and then reconfigure it.

The ACL editing feature uses an ACL sequence number that is added to each ACL **permit** or **deny** statement, with the numbers representing the sequence of statements in the ACL. ACL sequence numbers provide the following features for both numbered and named ACLs:

Key Topic

New configuration style for numbered: Numbered ACLs use a configuration style like named ACLs, as well as the traditional style, for the same ACL; the new style is required to perform advanced ACL editing.

Deleting single lines: An individual ACL **permit** or **deny** statement can be deleted with a **no** *sequence-number* subcommand.

Inserting new lines: Newly added **permit** and **deny** commands can be configured with a sequence number before the **deny** or **permit** command, dictating the location of the statement within the ACL.

Automatic sequence numbering: IOS adds sequence numbers to commands as you configure them, even if you do not include the sequence numbers.

To take advantage of the ability to delete and insert lines in an ACL, both numbered and named ACLs must use the same overall configuration style and commands used for named ACLs. The only difference in syntax is whether a name or number is used. Example 17-6 shows the configuration of a standard numbered IP ACL, using this alternative configuration style. The example shows the power of the ACL sequence number for editing. In this example, the following occurs:

Step 1. Numbered ACL 24 is configured using this new-style configuration, with three **permit** commands.

Step 2. The **show ip access-lists** command shows the three **permit** commands with sequence numbers 10, 20, and 30.

Step 3. The engineer deletes only the second **permit** command using the **no 20** ACL subcommand, which simply refers to sequence number 20.

Step 4. The **show ip access-lists** command confirms that the ACL now has only two lines (sequence numbers 10 and 30).

Step 5. The engineer adds a new **deny** command to the beginning of the ACL, using the **5 deny 10.1.1.1** ACL subcommand.

Step 6. The **show ip access-lists** command again confirms the changes, this time listing three commands, sequence numbers 5, 10, and 30.

NOTE For this example, note that the user does not leave configuration mode, instead using the **do** command to tell IOS to issue the **show ip access-lists** EXEC command from configuration mode.

Example 17-6 *Editing ACLs Using Sequence Numbers*

```
! Step 1: The 3-line Standard Numbered IP ACL is configured.
R1# configure terminal
Enter configuration commands, one per line.  End with Ctrl-Z.
R1(config)# ip access-list standard 24
R1(config-std-nacl)# permit 10.1.1.0 0.0.0.255
R1(config-std-nacl)# permit 10.1.2.0 0.0.0.255
R1(config-std-nacl)# permit 10.1.3.0 0.0.0.255

! Step 2: Displaying the ACL's contents, without leaving configuration mode.
R1(config-std-nacl)# do show ip access-lists 24
Standard IP access list 24
    10 permit 10.1.1.0, wildcard bits 0.0.0.255
    20 permit 10.1.2.0, wildcard bits 0.0.0.255
    30 permit 10.1.3.0, wildcard bits 0.0.0.255

! Step 3: Still in ACL 24 configuration mode, the line with sequence number 20 is
  deleted.
R1(config-std-nacl)# no 20

! Step 4: Displaying the ACL's contents again, without leaving configuration mode.
! Note that line number 20 is no longer listed.
R1(config-std-nacl)#do show ip access-lists 24
Standard IP access list 24
    10 permit 10.1.1.0, wildcard bits 0.0.0.255
    30 permit 10.1.3.0, wildcard bits 0.0.0.255

! Step 5: Inserting a new first line in the ACL.
R1(config-std-nacl)# 5 deny 10.1.1.1

! Step 6: Displaying the ACL's contents one last time, with the new statement
!(sequence number 5) listed first.
R1(config-std-nacl)# do show ip access-lists 24
Standard IP access list 24
     5 deny   10.1.1.1
    10 permit 10.1.1.0, wildcard bits 0.0.0.255
    30 permit 10.1.3.0, wildcard bits 0.0.0.255
```

Note that although Example 17-6 uses a numbered ACL, named ACLs use the same process to edit (add and remove) entries.

Numbered ACL Configuration Versus Named ACL Configuration

As a brief aside about numbered ACLs, note that IOS actually allows two ways to configure numbered ACLs in the more recent versions of IOS. First, IOS supports the traditional method, using the **access-list** global commands shown earlier in Examples 17-1, 17-2, and 17-3. IOS also supports the numbered ACL configuration with commands just like named ACLs, as shown in Example 17-6.

Oddly, IOS always stores numbered ACLs with the original style of configuration, as global **access-list** commands, no matter which method is used to configure the ACL. Example 17-7 demonstrates these facts, picking up where Example 17-6 ended, with the following additional steps:

Step 7. The engineer lists the configuration (**show running-config**), which lists the old-style configuration commands—even though the ACL was created with the new-style commands.

Step 8. The engineer adds a new statement to the end of the ACL using the old-style **access-list 24 permit 10.1.4.0 0.0.0.255** global configuration command.

Step 9. The **show ip access-lists** command confirms that the old-style **access-list** command from the previous step followed the rule of being added only to the end of the ACL.

Step 10. The engineer displays the configuration to confirm that the parts of ACL 24 configured with both new-style commands and old-style commands are all listed in the same old-style ACL (**show running-config**).

Example 17-7 *Adding to and Displaying a Numbered ACL Configuration*

```
! Step 7: A configuration snippet for ACL 24.
R1# show running-config
```

```
! The only lines shown are the lines from ACL 24
access-list 24 deny   10.1.1.1
access-list 24 permit 10.1.1.0 0.0.0.255
access-list 24 permit 10.1.3.0 0.0.0.255

! Step 8: Adding a new access-list 24 global command
R1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# access-list 24 permit 10.1.4.0 0.0.0.255
R1(config)# ^Z

! Step 9: Displaying the ACL's contents again, with sequence numbers. Note that even
! the new statement has been automatically assigned a sequence number.
R1# show ip access-lists 24
Standard IP access list 24
    5 deny   10.1.1.1
    10 permit 10.1.1.0, wildcard bits 0.0.0.255
    30 permit 10.1.3.0, wildcard bits 0.0.0.255
    40 permit 10.1.4.0, wildcard bits 0.0.0.255

! Step 10: The numbered ACL configuration remains in old-style configuration commands.
R1# show running-config
! The only lines shown are the lines from ACL 24
access-list 24 deny   10.1.1.1
access-list 24 permit 10.1.1.0 0.0.0.255
access-list 24 permit 10.1.3.0 0.0.0.255
access-list 24 permit 10.1.4.0 0.0.0.255
```

ACL Implementation Considerations

ACLs can be a great tool to enhance the security of a network, but engineers should think about some broader issues before simply configuring an ACL to fix a problem. To help, Cisco makes the following general recommendations in the courses on which the CCNA R&S exams are based:

Key Topic

- Place extended ACLs as close as possible to the source of the packet. This strategy allows ACLs to discard the packets early.
- Place standard ACLs as close as possible to the destination of the packet. This strategy avoids the mistake with standard ACLs (which match the source IPv4 address only) of unintentionally discarding packets that did not need to be discarded.
- Place more specific statements early in the ACL.
- Disable an ACL from its interface (using the **no ip access-group** interface subcommand) before making changes to the ACL.

The first point deals with the concept of where to locate your ACLs. If you intend to filter a packet, filtering closer to the packet's source means that the packet takes up less bandwidth in the network, which seems to be more efficient—and it is. Therefore, Cisco suggests locating extended ACLs as close to the source as possible.

However, the second point seems to contradict the first point, at least for standard ACLs, to locate them close to the destination. Why? Well, because standard ACLs look only at the source IP address, they tend to filter more than you want filtered when placed close to the source. For example, imagine that Fred and Barney are separated by four routers. If you filter Barney's traffic sent to Fred on the first router, Barney can't reach any hosts near the other three routers. So, the Cisco courses make a blanket recommendation to locate standard ACLs closer to the destination to avoid filtering traffic you do not mean to filter.

For the third item in the list, by placing more specific matching parameters early in each list, you are less likely to make mistakes in the ACL. For example, imagine that the ACL first listed a command that permitted traffic going to 10.1.1.0/24, and the second command denied traffic going to host 10.1.1.1. Packets sent to host 10.1.1.1 would match the first command, and never match the more specific second command. Note that later IOS versions prevent this mistake during configuration in some cases, as shown later in this chapter in Example 17-11.

Finally, Cisco recommends that you disable the ACLs on the interfaces before you change the statements in the list. By doing so, you avoid issues with the ACL during an interim state. First, if you delete an entire ACL, and leave the IP ACL enabled on an interface with the **ip access-group** command, IOS does not filter any packets (that was not always the case in far earlier IOS versions)! As soon as you add one ACL command to that enabled ACL, however, IOS starts filtering packets based on that ACL. Those interim ACL configurations could cause problems.

For example, suppose you have ACL 101 enabled on S0/0/0 for output packets. You delete list 101 so that all packets are allowed through. Then, you enter a single **access-list 101** command. As soon as you press Enter, the list exists, and the router filters all packets exiting S0/0/0 based on the one-line list. If you want to enter a long ACL, you might temporarily filter packets you don't want to filter! Therefore, the better way is to disable the list from the interface, make the changes to the list, and then reenable it on the interface.

Troubleshooting with IPv4 ACLs

The use of IPv4 ACLs makes troubleshooting IPv4 routing more difficult. Any data plane troubleshooting process can include a catchall phrase to include checking for ACLs. A network can have all hosts working, DHCP settings correct, all LANs working, all router interfaces working, and all routers having learned all routes to all subnets—and ACLs can still filter packets. Although ACLs provide that important service of filtering some packets, ACLs can make the troubleshooting process that much more difficult.

This third of the three major sections of this chapter focuses on troubleshooting in the presence of IPv4 ACLs. It breaks the discussion into two parts. The first part gives advice about common problems you might see on the exam, and how to find those with **show** commands and some analysis. The second part then looks at how ACLs impact the **ping** command.

Analyzing ACL Behavior in a Network

ACLs cause some of the biggest challenges when troubleshooting problems in real networking jobs. The packets created by commands like **ping** and **traceroute** do not exactly match the fields in packets created by end users. The ACLs sometimes filter the **ping** and **traceroute** traffic, making the network engineer think some other kind of problems exists when no problems exist at all. Or, the problem with the end-user traffic really is caused by the ACL, but the ping and traceroute traffic works fine, because the ACL matches the end-user traffic with a **deny** action but matches the ping and traceroute traffic with a **permit** action.

As a result, much of ACL troubleshooting requires thinking about ACL configuration versus the packets that flow in a network, rather than using a couple of IOS commands that identify the root cause of the problem. The **show** commands that help are those that give you the configuration of the ACL, and on what interfaces the ACL is enabled. You can also see statistics about which ACL statements have been matched. And using pings and traceroutes can help—as long as you remember that ACLs may apply different actions to those packets versus the end-user traffic.

The following phrases the ACL troubleshooting steps into a list for easier study. The list also expands on the idea of analyzing each ACL in Step 3. None of the ideas in the list are new compared to this chapter and the previous chapter, but it acts more as a summary of the common issues:

Key Topic

Step 1. Determine on which interfaces ACLs are enabled, and in which direction (**show running-config, show ip interfaces**).

Step 2. Find the configuration of each ACL (**show access-lists, show ip access-lists, show running-config**).

Step 3. Analyze the ACLs to predict which packets should match the ACL, focusing on the following points:

A. **Misordered ACLs:** Look for misordered ACL statements. IOS uses first-match logic when searching an ACL.

B. **Reversed source/destination addresses:** Analyze the router interface, the direction in which the ACL is enabled, compared to the location of the IP address ranges matched by the ACL statements. Make sure the source IP address field could match packets with that source IP address, rather than the destination, and vice versa for the destination IP address field.

C. **Reversed source/destination ports:** For extended ACLs that reference UDP or TCP port numbers, continue to analyze the location and direction of the ACL versus the hosts, focusing on which host acts as the server using a well-known port. Ensure that the ACL statement matches the correct source or destination port depending on whether the server sent or will receive the packet.

D. **Syntax:** Remember that extended ACL commands must use the **tcp** and **udp** keywords if the command needs to check the port numbers.

E. **Syntax:** Note that ICMP packets do not use UDP or TCP; ICMP is considered to be another protocol matchable with the **icmp** keyword (instead of **tcp** or **udp**).

F. **Explicit deny any:** Instead of using the implicit **deny any** at the end of each ACL, use an explicit configuration command to deny all traffic at the end of the ACL so that the **show** command counters increment when that action is taken.

G. **Dangerous inbound ACLs:** Watch for inbound ACLs, especially those with deny all logic at the end of the ACL. These ACLs may discard incoming overhead protocols, like routing protocol messages.

H. **Standard ACL location:** Standard ACLs enabled close to the source of matched addresses can discard the packets as intended, but also discard packets that should be allowed through. Always pay close attention to the requirements of the ACL in these cases.

This chapter (and the previous) have already discussed the details of Step 3. The first two steps are important for Simlet questions in case you are not allowed to look at the configuration; you can use other **show** commands to determine all the relevant ACL configuration. The next few pages show some of the related commands and how they can uncover some of the issues described in the just-completed ACL troubleshooting checklist.

ACL Troubleshooting Commands

If you suspect ACLs are causing a problem, the first problem-isolation step is to find the location and direction of the ACLs. The fastest way to do this is to look at the output of the **show running-config** command and to look for **ip access-group** commands under each interface. However, in some cases, enable mode access may not be allowed, and **show** commands are required. Instead, use the **show ip interfaces** command to find which ACLs are enabled on which interfaces, as shown in Example 17-8.

Example 17-8 *Sample* **show ip interface** *Command*

```
R1> show ip interface s0/0/1
Serial0/0/1 is up, line protocol is up
  Internet address is 10.1.2.1/24
  Broadcast address is 255.255.255.255
  Address determined by setup command
  MTU is 1500 bytes
  Helper address is not set
  Directed broadcast forwarding is disabled
  Multicast reserved groups joined: 224.0.0.9
  Outgoing access list is not set
  Inbound  access list is 102
! roughly 26 more lines omitted for brevity
```

Note that the command output lists whether an ACL is enabled, in both directions, and which ACL it is. The example shows an abbreviated version of the **show ip interface S0/0/1** command, which lists messages for just this one interface. The **show ip interface** command would list the same messages for every interface in the router.

Step 2 of the ACL troubleshooting checklist then says that the contents of the ACL must be found. Again, the quickest way to look at the ACL is to use the **show running-config** command. If it's not available, the **show access-lists** and **show ip access-lists** commands list the same details shown in the configuration. These commands also list a useful counter that lists the number of packets that have matched each line in the ACL. Example 17-9 shows an example.

Example 17-9 **show ip access-lists** *Command Example*

```
R1# show ip access-lists
Extended IP access list 102
    10 permit ip 10.1.2.0 0.0.0.255 10.1.4.0 0.0.1.255 (15 matches)
```

The counter can be very useful for troubleshooting. If you can generate traffic that you think should match a particular line in an ACL, then you should see the matches increment on that counter. If you keep generating traffic that should match, but that line's counter never goes up, then those packets do not match that line in that ACL. Those packets could be matching an earlier line in the same ACL, or might not even be reaching that router (for any reason).

After the locations, directions, and configuration details of the various ACLs have been discovered in Steps 1 and 2, the hard part begins—analyzing what the ACL really does. For example, one of the most common tasks you will do is to look at the address fields and decide the range of addresses matched by that field. Remember, for an ACL that sits in a router configuration, you can easily find the address range. The low end of the range is the address (the first number), and the high end of the range is the sum of the address and wildcard mask. For instance, with ACL 102 in Example 17-9, which is obviously configured in some router, the ranges are as follows:

Source 10.1.2.0, wildcard 0.0.0.255: Matches from 10.1.2.0 through 10.1.2.255

Destination 10.1.4.0, wildcard 0.0.1.255: Matches from 10.1.4.0 through 10.1.5.255

The next few pages work through some analysis of a few of the items from Step 3 in the troubleshooting checklist.

Example Issue: Reversed Source/Destination IP Addresses

IOS cannot recognize a case in which you attempt to match the wrong addresses in the source or destination address field. So, be ready to analyze the enabled ACLs and their direction versus the location of different subnets in the network. Then ask yourself about the packets that drive that ACL: what could the source and destination addresses of those packets be? And does the ACL match the correct address ranges, or not?

For example, consider Figure 17-11, a figure that will be used in several troubleshooting examples in this chapter. The requirements for the next ACL follow the figure.

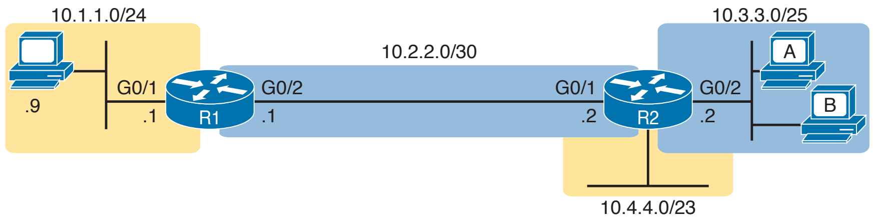


Figure 17-11 *Example Network Used in IPv4 ACL Troubleshooting Examples*

For this next ACL, the requirements ask that you allow and prevent various flows, as follows:

- Allow hosts in subnet 10.3.3.0/25 and subnet 10.1.1.0/24 to communicate
- Prevent hosts in subnet 10.4.4.0/23 and subnet 10.1.1.0/24 from communicating
- Allow all other communications between hosts in network 10.0.0.0
- Prevent all other communications

Example 17-10 shows the ACL used in this case on R2. At first glance, it meets all those requirements straight down the list.

Example 17-10 *Troubleshooting Example 2 per Step 3B: Source and Destination Mismatch*

```
R2# show ip access-lists
Standard IP access list Step3B
 10 permit 10.3.3.0 0.0.0.127
 20 deny 10.4.4.0 0.0.1.255
 30 permit 10.0.0.0 0.255.255.255 (12 matches)
R2#
R2# show ip interface G0/2 | include Inbound
  Inbound access list is Step3B
```

The problem in this case is that the ACL has been enabled on R2's G0/2 interface, inbound. Per the figure, packets coming from a source address in subnets 10.3.3.0/25 and 10.4.4.0/23 should be forwarded out R2's G0/2 interface, rather than coming in that interface. So, do not let the matching logic in the ACL that perfectly mirrors the requirements fool you; make sure and check the location of the ACL, direction, and the location of the IP addresses.

Note that Step 3C suggests a similar issue regarding matching well-known ports with TCP and UDP. The earlier section in this chapter titled "Matching TCP and UDP Port Numbers" has already discussed those ideas in plenty of detail. Just make sure to check where the server sits versus the location and direction of the ACL.

Steps 3D and 3E: Common Syntax Mistakes

Steps 3D and 3E describe a couple of common syntax mistakes. First, to match a TCP port in an ACL statement, you must use a **tcp** protocol keyword instead of **ip** or any other value. Otherwise, IOS rejects the command as having incorrect syntax. Same issue with trying to match UDP ports: a **udp** protocol keyword is required.

To match ICMP, IOS includes an **icmp** protocol keyword to use instead of **tcp** or **udp**. In fact, the main conceptual mistake is to think of ICMP as an application protocol that uses either UDP or TCP; it uses neither. To match all ICMP messages, for instance, use the **permit icmp any any** command in an extended named ACL.

Example Issue: Inbound ACL Filters Routing Protocol Packets

A router bypasses outbound ACL logic for packets the router itself generates. That might sound like common sense, but it is important to stop and think about that fact in context. A router can have an outgoing ACL, and that ACL can and will discard packets that the router receives in one interface and then tries to forward out some other interface. But if the router creates the packet, for instance, for a routing protocol message, the router bypasses the outbound ACL logic for that packet.

However, a router does not bypass inbound ACL logic. If an ACL has an inbound ACL enabled, and a packet arrives in that interface, the router checks the ACL. Any and all IPv4 packets are considered by the ACL—including important overhead packets like routing protocol updates.

For example, consider a seemingly good ACL on a router, like the Step3G ACL in Example 17-11. That ACL lists a couple of **permit** commands, and has an implicit deny any at the end of the list. At first, it looks like any other reasonable ACL.

Example 17-11 *Troubleshooting Example 2 per Step 3G: Filtering RIP by Accident*

```
R1# show ip access-lists
Standard IP access list Step3G
 10 permit host 10.4.4.1
 20 permit 10.3.3.0 0.0.0.127 (12 matches)
! using the implicit deny to match everything else
R1#
! On router R1:
R1# show ip interface G0/2 | include Inbound
  Inbound access list is Step3G
```

Now look at the location and direction (inbound on R1, on R1's G0/2) and consider that location versus the topology Figure 17-11 for a moment. None of those **permit** statements match the RIP updates sent by R2, sent out R2's G0/1 interface toward R1. RIP messages use UDP (well-known port 520), and R2's G0/1 interface is 10.2.2.2 per the figure. R1 would match incoming RIP messages with the implicit deny all at the end of the list. The symptoms in this case, assuming only that one ACL exists, would be that R1 would not learn routes from R2, but R2 could still learn RIP routes from R1.

Of the three routing protocols discussed in the ICND1 and ICND2 books, RIPv2 uses UDP as a transport, while OSPF and EIGRP do not even use a transport protocol. As a result, to match RIPv2 packets with an ACL, you need the **udp** keyword and you need to match well-known port 520. OSPF and EIGRP can be matched with special keywords as noted in Table 17-7. The table also list the addresses used by each protocol.

Table 17-7 Key Fields for Matching Routing Protocol Messages

Protocol	Source IP Address	Destination IP Addresses	ACL Protocol Keyword
RIPv2	Source interface	224.0.0.9	**udp** (port 520)
OSPF	Source interface	224.0.0.5, 224.0.0.6	**ospf**
EIGRP	Source interface	224.0.0.10	**eigrp**

Example 17-12 shows a sample ACL with three lines, one to match each routing protocol, just to show the syntax. Note that in this case, the ACL matches the address fields with the **any** keyword. You could include lines like these in any inbound ACL to ensure that routing protocol packets would be permitted.

Example 17-12 *Example ACL that Matches all RIPv2, OSPF, and EIGRP with a Permit*

```
R1# show ip access-lists
ip access-list extended RoutingProtocolExample
 10 permit udp any any eq 520
 20 permit ospf any any
 30 permit eigrp any any
 remark a complete ACL would also need more statements here
R1#
```

ACL Interactions with Router-Generated Packets

Routers bypass outbound ACL logic for packets generated by that same router. This logic helps avoid cases in which a router discards its own overhead traffic. This logic applies to packets that a router creates for overhead processes like routing protocols, as well as for commands, like **ping** and **traceroute**. This section adds a few perspectives about how ACLs impact troubleshooting, and how this exception to outbound ACL logic applies, particularly commands used from the router CLI.

Local ACLs and a Ping from a Router

For the first scenario, think about a **ping** command issued by a router. The command generates packets, and the router sends those packets (holding the ICMP echo request messages) out one of the router interfaces, and typically some ICMP echo reply messages are received back. As it turns out, not all ACLs will attempt to filter those packets.

As a backdrop to discuss what happens, Figure 17-12 illustrates a simple network topology with two routers connected to a serial link. Note that in this figure four IP ACLs exist, named A, B, C, and D, as noted by the thick arrows in the drawing. That is, ACL A is an outbound ACL on R1's S0/0/0, ACL B is an inbound ACL on R2's S0/0/1, and so on.

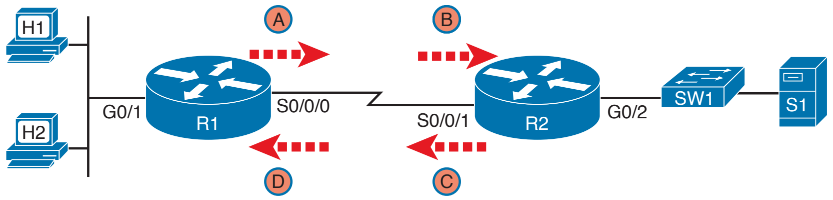


Figure 17-12 *Sample Network with IP ACLs in Four Locations*

As an example, consider a **ping** command issued from R1's CLI (after a user connects to R1's CLI using SSH). The **ping** command pings server S1's IP address. The IPv4 packets with the ICMP messages flow from R1 to S1 and back again. Which of those four ACLs could possibly filter the ICMP Echo Request toward S1, and the ICMP Echo Reply back toward R1?

Routers bypass their own outbound ACLs for packets generated by the router, as shown in Figure 17-13. Even though ACL A exists as an outgoing ACL on Router R1, R1 bypasses its own outgoing ACL logic of ACL A for the ICMP Echo Requests generated by R1.

Key Topic

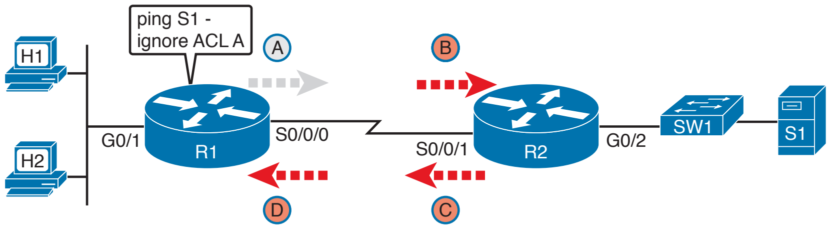


Figure 17-13 *R1 Ignores Outgoing ACL for Packets Created by Its Own* **ping** *Command*

Router Self-Ping of a Serial Interface IPv4 Address

The previous example uses a router's **ping** command when pinging a host. However, network engineers often need to ping router IP addresses, including using a self-ping. The term *self-ping* refers to a ping of a device's own IPv4 address. And for point-to-point serial links, a self-ping actually sends packets over the serial link, which causes some interesting effects with ACLs.

When a user issues a self-ping for that local router's serial IP address, the router actually sends the ICMP echo request out the link to the other router. The neighboring router then receives the packet and routes the packet with the ICMP echo request back to the original router. Figure 17-14 shows an example of a self-ping (**ping 172.16.4.1**) of Router R1's own IP address on a point-to-point serial link, with the ICMP echo request out the link to Router R2. At Step 2, R2 treats it like any other packet not destined for one of R2's own IPv4 addresses: R2 routes the packet. Where? Right back to Router R1, as shown in the figure.

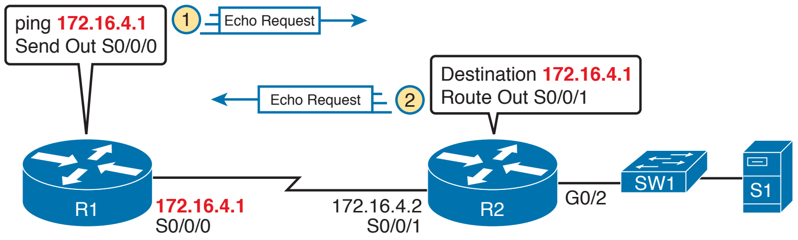


Figure 17-14 *The First Steps in a Self-Ping on R1, for R1's S0/0/0 IP Address*

Now think about those four ACLs in the earlier figures compared to Figure 17-14. R1 generates the ICMP echo request, so R1 bypasses outbound ACL A. ACLs B, C, and D could filter the packet. Note that the packet sent by R2 back to R1 is not generated by R2 in this case; R2 is just routing R1's original packet back to R1.

A self-ping of a serial interface actually tests many parts of a point-to-point serial link, as follows:

- The link must work at Layers 1, 2, and 3. Specifically, both routers must have a working (up/up) serial interface, with correct IPv4 addresses configured.
- ACLs B, C, and D must permit the ICMP echo request and reply packets.

So, when troubleshooting, if you choose to use self-pings and they fail, but the serial interfaces are in an up/up state, do not forget to check to see whether the ACLs have filtered the Internet Control Management Protocol (ICMP) traffic.

Router Self-Ping of an Ethernet Interface IPv4 Address

A self-ping of a router's own Ethernet interface IP address works a little like a self-ping of a router's serial IP address, but with a couple of twists:

- Like with serial interface, the local router interface must be working (in an up/up state); otherwise, the ping fails.
- Unlike serial interfaces, the router does not forward the ICMP messages physically out the interface, so security features on neighboring switches (like port security) or routers (like ACLs) cannot possibly filter the messages used by the **ping** command.
- Like serial interfaces, an incoming IP ACL on the local router does process the router self-ping of an Ethernet-based IP address.

Figure 17-15 walks through an example. In this case, R2 issues a **ping 172.16.2.2** command to ping its own G0/2 IP address. Just like with a self-ping on serial links, R2 creates the

ICMP echo request. However, R2 basically processes the ping down its own TCP/IP stack and back up again, with the ICMP echo never leaving the router's Ethernet interface. R2 does check the Ethernet interface status, showing a failure if the interface is not up/up. R2 does not apply outbound ACL logic to the packet, because R2 created the packet, but R2 will apply inbound ACL logic to the packet, as if the packet had been physically received on the interface.

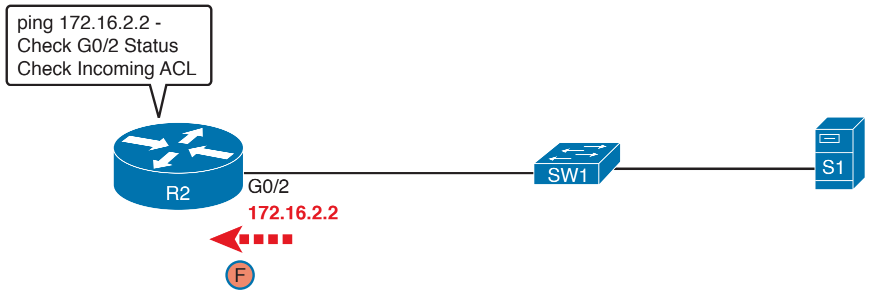


Figure 17-15 *Self-Ping of a Router's Ethernet Address*

Chapter Review

One key to doing well on the exams is to perform repetitive spaced review sessions. Review this chapter's material using either the tools in the book, DVD, or interactive tools for the same material found on the book's companion website. Refer to the "Your Study Plan" element for more details. Table 17-8 outlines the key review elements and where you can find them. To better track your study progress, record when you completed these activities in the second column.

Table 17-8 Chapter Review Tracking

Review Element	Review Date(s)	Resource Used
Review key topics		Book, DVD/website
Review key terms		Book, DVD/website
Repeat DIKTA questions		Book, PCPT
Review memory tables		Book, DVD/website
Review command tables		Book

Review All the Key Topics

Key Topic

Table 17-9 Key Topics for Chapter 17

Key Topic Element	Description	Page Number
Figure 17-3	Syntax and notes about the three required matching fields in the extended ACL **access-list** command	463
Paragraph	Summary of extended ACL logic that all parameters must match in a single **access-list** statement for a match to occur	464

Key Topic Element	Description	Page Number
Figure 17-4	Drawing of the IP header followed by a TCP header	464
Figure 17-5	Syntax and notes about matching TCP and UDP ports with extended ACL **access-list** commands	465
Figure 17-7	Logic and syntax to match TCP source ports	466
List	Guidelines for using extended numbered IP ACLs	467
List	Differences between named and numbered ACLs when named ACLs introduced	471
List	Features enabled by IOS 12.3 ACL sequence numbers	473
List	ACL implementation recommendations	476
Checklist	ACL troubleshooting checklist	478
Figure 17-13	Example of a router bypassing its outbound ACL logic for packets the router generates	483

Key Terms You Should Know

extended access list, named access list

Command References

Tables 17-10 and 17-11 list configuration and verification commands used in this chapter. As an easy review exercise, cover the left column in a table, read the right column, and try to recall the command without looking. Then repeat the exercise, covering the right column, and try to recall what the command does.

Table 17-10 Chapter 17 ACL Configuration Command Reference

Command	Description
access-list *access-list-number* {**deny** | **permit**} *protocol source source-wildcard destination destination-wildcard* [**log**]	Global command for extended numbered **access lists**. Use a number between 100 and 199 or 2000 and 2699, inclusive.
access-list *access-list-number* {**deny** | **permit**} **tcp** *source source-wildcard* [*operator* [*port*]] *destination destination-wildcard* [*operator* [*port*]] [**log**]	A version of the **access-list** command with TCP-specific parameters.
access-list *access-list-number* **remark** *text*	Defines a remark that helps you remember what the ACL is supposed to do.
ip access-group {*number* | *name* [**in** | **out**]}	Interface subcommand to enable access lists.
access-class *number* | *name* [**in** | **out**]	Line subcommand to enable either standard or extended access lists on vty lines.
ip access-list {**standard** | **extended**} *name*	Global command to configure a named standard or extended ACL and enter ACL configuration mode.
{**deny** | **permit**} *source* [*source wildcard*] [**log**]	ACL mode subcommand to configure the matching details and action for a standard named ACL.

Command	Description
{**deny** | **permit**} *protocol source source-wildcard destination destination-wildcard* [**log**]	ACL mode subcommand to configure the matching details and action for an extended named ACL.
{**deny** | **permit**} **tcp** *source source-wildcard* [*operator* [*port*]] *destination destination-wildcard* [*operator* [*port*]] [**log**]	ACL mode subcommand to configure the matching details and action for a named ACL that matches TCP segments.
remark *text*	ACL mode subcommand to configure a description of a named ACL.

Table 17-11 Chapter 17 EXEC Command Reference

Command	Description
show ip *interface* [*type number*]	Includes a reference to the access lists enabled on the interface
show access-lists [*access-list-number* | *access-list-name*]	Shows details of configured access lists for all protocols
show ip access-lists [*access-list-number* | *access-list-name*]	Shows IP access lists

Answers to Earlier Practice Problems

Table 17-12 lists the answers to the practice problems listed in Table 17-6. Note that for any question that references a client, you might have chosen to match port numbers greater than 1023. The answers in this table mostly ignore that option, but just to show one sample, the answer to the first problem lists one with a reference to client ports greater than 1023 and one without. The remaining answers simply omit this part of the logic.

Table 17-12 Building One-Line Extended ACLs: Answers

	Criteria
1	**access-list 101 permit tcp host 10.1.1.1 10.1.2.0 0.0.0.255 eq www** or **access-list 101 permit tcp host 10.1.1.1 gt 1023 10.1.2.0 0.0.0.255 eq www**
2	**access-list 102 permit tcp 172.16.4.0 0.0.0.127 172.16.3.0 0.0.0.127 eq telnet**
3	**access-list 103 permit icmp 192.168.7.192 0.0.0.63 192.168.7.8 0.0.0.7**
4	**access-list 104 permit tcp 10.2.2.0 0.0.1.255 eq www 10.4.4.0 0.0.3.255**
5	**access-list 105 permit tcp 172.20.1.0 0.0.0.255 eq 23 172.20.44.0 0.0.1.255**
6	**access-list 106 permit tcp 192.168.99.96 0.0.0.15 192.168.176.0 0.0.0.15 eq www**
7	**access-list 107 permit icmp 10.55.66.0 0.0.0.127 10.66.55.0 0.0.0.63**
8	**access-list 108 permit ip any any**