Key Topic

Table 2-5 *Trunking Configuration Options That Lead to a Working Trunk*

Configuration Command on One Side[1]	Short Name	Meaning	To Trunk, Other Side Must Be
switchport mode trunk	Trunk	Always trunks on this end; sends DTP to help other side choose to trunk	On, desirable, auto
switchport mode trunk; switchport nonegotiate	Nonegotiate	Always trunks on this end; does not send nor process DTP messages (good when other switch is a non-Cisco switch)	On
switchport mode dynamic desirable	Desirable	Sends DTP messages indicating dynamic mode with preferred trunking, and trunks if negotiation succeeds	On, desirable, auto
switchport mode dynamic auto	Auto	Sends DTP messages indicating dynamic mode with preferred access, and trunks if negotiation succeeds	On, desirable
switchport mode access	Access	Never trunks; can send a single DTP message when entering the access mode to help other side reach same conclusion, ceases to send and process DTP messages afterward	(Never trunks)
switchport mode access; switchport nonegotiate	Access (with nonegotiate)	Never trunks; does not send or process DTP messages	(Never trunks)

1 When the **switchport nonegotiate** command is not listed in the first column, the default (DTP negotiation is active) is assumed.

Note If an interface trunks, the type of trunking (ISL or 802.1Q) is controlled by the setting on the **switchport trunk encapsulation** command if the switch supports multiple trunk encapsulations. This command includes an option for dynamically negotiating the type (using DTP) or configuring one of the two types.

Also, for DTP negotiation to succeed, both switches must either be configured with the same VTP domain name, or at least one switch must have its VTP domain name unconfigured (that is, NULL).

Configuring Trunking on Routers

VLAN trunking can be used on routers and hosts as well as on switches. However, routers do not support DTP, so you must manually configure them to support trunking. Additionally, you must manually configure a switch on the other end of the segment to trunk, because the router does not participate in DTP.

The majority of router trunking configurations use subinterfaces, with each subinterface being associated with one VLAN. The subinterface number does not have to match the VLAN ID; rather, the **encapsulation** command sits under each subinterface, with the associated VLAN ID being part of the **encapsulation** command. Use subinterface numbers starting with 1; the subinterface number 0 is the physical interface itself (for example, interface Fa0/0.0 is the Fa0/0 itself). Also, because good design calls for one IP subnet per VLAN, if the router wants to forward IP packets between the VLANs, the router needs to have an IP address associated with each trunking subinterface.

You can configure 802.1Q native VLANs under a subinterface or under the physical interface on a router. If they are configured under a subinterface, you use the **encapsulation dot1q** *vlan-id* **native** subcommand, with the inclusion of the **native** keyword meaning that frames exiting this subinterface should not be tagged, and incoming untagged frames shall be processed by this subinterface. As with other router trunking configurations, the associated IP address would be configured on that same subinterface. Alternately, if not configured on a subinterface, the router assumes that the native VLAN is associated with the physical interface. In this case, the **encapsulation** command is not needed nor supported under the physical interface; the associated IP address, however, would need to be configured under the physical interface. Configuring an (understandably distinct) IP address on both physical interface and a subinterface under the same physical interface using **encapsulation dot1q** *vlan-id* **native**, thereby technically resulting in two different interfaces for the native VLAN, is not supported. All incoming untagged frames will be processed by the subinterface configuration only. A notable exception to this rule can be seen on ISR G1 routers equipped with 10-Mbps Ethernet built-in interfaces. On these router platforms, settings for the native VLAN shall be configured on the physical Ethernet interface directly. While the router will accept the configuration of a subinterface with the **encapsulation dot1q** *vlan-id* **native** command, incoming untagged frames will be processed by the configuration of the physical interface. This exception applies only to ISR platforms with 10-Mbps Ethernet interfaces, and is not present on platforms with Fast Ethernet or faster interfaces.

If the router supports native VLAN configuration on a subinterface, it is recommended to use subinterfaces instead of putting the native VLAN configuration on a physical port. Aside from keeping the configuration more consistent (all configuration being placed on subinterfaces), this configuration allows the router to correctly process frames that, despite being originated in the native VLAN, carry an 802.1Q tag. Tagging such frames is done when using the CoS field inside an 802.1Q tag. If the native VLAN configuration was done on a physical interface, the router would not be able to recognize that a frame carrying an 802.1Q tag with a nonzero VLAN ID is really a CoS-marked frame in the native VLAN. When using subinterfaces, the **encapsulation dot1q** *vlan-id* **native** command allows the router to recognize that both untagged frames and CoS-marked frames tagged with the particular *vlan-id* should be processed as frames in the native VLAN.

Example 2-8 shows an example configuration for Router1 in Figure 2-1, both for ISL and 802.1Q. In this case, Router1 needs to forward packets between the subnets on VLANs 21 and 22. The first part of the example shows ISL configuration, with no native VLANs, and therefore only a subinterface being used for each VLAN. The second part of the example shows an alternative 802.1Q configuration, using the option of placing the native VLAN (VLAN 21) configuration on the physical interface.

Key Topic

Example 2-8 *Trunking Configuration on Router1*

```
! Note the subinterface on the Fa0/0 interface, with the encapsulation
! command noting the type of trunking, as well as the VLAN number. The subinterface
! number does not have to match the VLAN ID. Also note the IP addresses for
! each interface, allowing Router1 to route between VLANs.
! The encapsulation command must be entered on a subinterface before entering any
! other IP-related commands, such as configuring an IP address.

Router1(config)# interface fa0/0
Router1(config-if)# no shutdown
Router1(config-if)# interface fa0/0.1
Router1(config-subif)# encapsulation isl 21
Router1(config-subif)# ip address 10.1.21.1 255.255.255.0
Router1(config-subif)# interface fa0/0.2
Router1(config-subif)# encapsulation isl 22
Router1(config-subif)# ip address 10.1.22.1 255.255.255.0

! Next, an alternative 802.1Q configuration is shown. Note that this configuration
! places the IP address for VLAN 21 on the physical interface; the router simply
! associates the physical interface with the native VLAN. Alternatively,
! a subinterface could be used, with the encapsulation dot1q 21 native command
! specifying that the router should treat this VLAN as the native VLAN.

Router1(config)# interface fa0/0
Router1(config-if)# ip address 10.1.21.1 255.255.255.0
Router1(config-if)# no shutdown
Router1(config-if)# interface fa0/0.2
Router1(config-subif)# encapsulation dot1q 22
Router1(config-subif)# ip address 10.1.22.1 255.255.255.0
```

Note also that the router does not have an explicitly defined allowed VLAN list on an interface. However, the allowed VLAN list is implied based on the configured VLANs. For example, in this example, when using ISL, Router1 allows VLANs 21 and 22, while when using 802.1Q, it allows the native VLAN and VLAN 22.

802.1Q-in-Q Tunneling

Traditionally, VLANs have not extended beyond the WAN boundary. VLANs in one campus extend to a WAN edge router, but VLAN protocols are not used on the WAN.

Today, several emerging alternatives exist for the passage of VLAN traffic across a WAN, including 802.1Q-in-Q, its standardized version 802.1ad called Provider Bridges, another standard 802.1ah called Provider Backbone Bridges, Layer2 Tunneling Protocol (L2TPv3),

Foundation Topics

802.1D Spanning Tree Protocol and Improvements

Although many CCIE candidates already know STP well, the details are easily forgotten. For example, you can install a campus LAN, possibly turn on a few STP optimizations and security features out of habit, and have a working LAN using STP—without ever really contemplating how STP does what it does. And in a network that makes good use of Layer 3 switching, each STP instance might span only three to four switches, making the STP issues much more manageable—but more forgettable in terms of helping you remember things you need to know for the exam. This chapter reviews the details of IEEE 802.1D STP, and then goes on to related topics—802.1w RSTP, multiple spanning trees, STP optimizations, and STP security features. STP terminology refers to bridges in many places; in the following sections, the words *bridge* and *switch* will be used interchangeably with respect to STP. While the upcoming sections about various STP versions might appear lengthy and reiterate on many known facts, be sure to read them very carefully in their entirety. It is always tiresome to read an in-depth discussion about a protocol as notorious as STP—but as we know, it's details that matter, especially for a CCIE. This chapter tries to put several details about STP straight, cleaning up numerous misconceptions that have crept in the common understanding of STP over the years of its existence.

Before diving into STP internals, it is worthwhile to comment on a possible naming confusion regarding various STP versions. The first IEEE-standardized STP, also often called the "legacy" STP, was originally described in 802.1D. Its improvements were subsequently published in so-called amendments: The Rapid STP (RSTP) was standardized in amendment 802.1w, while Multiple STP (MSTP) was covered in amendment 802.1s. Since then, the amendments have been integrated into existing standards. The latest 802.1D-2004 standard no longer includes the legacy STP at all (which is considered obsolete), and instead, it covers the RSTP originally found in 802.1w. The 802.1s MSTP is integrated into 802.1Q-2005 and later revisions. With current standards, therefore, RSTP is covered in 802.1D while MSTP is covered in 802.1Q, and legacy STP has been dropped. Still, many people are used to the old naming, with 802.1D referring to STP, 802.1w referring to RSTP, and 802.1s referring to MSTP.

STP uses messaging between switches to stabilize the network into a logical loop-free topology. To do so, STP causes some interfaces (popularly called *ports* when discussing STP) to simply not forward or receive traffic—in other words, the ports are in a *Blocking* state. The remaining ports, in an STP *Forwarding* state, together provide a loop-free path to every Ethernet segment in the network.

STP protocol messages are called Bridge Protocol Data Units (BPDU), the basic structure for which is shown in Figure 3-1.

Configuration BPDU

BPDU Field	Length in Octets
Protocol Identifier	2
Protocol Version	1
BPDU Type	1
Flags	1
Root Bridge ID	8
Root Path Cost	4
Sending Bridge ID	8
Sending Port ID	2
Message Age	2
Max Age	2
Hello Time	2
Forward Delay	2

Topology Change Notification BPDU

BPDU Field	Length in Octets
Protocol Identifier	2
Protocol Version	1
BPDU Type	1

Figure 3-1 *Format of STP Bridge Protocol Data Units*

For STP, the Protocol Identifier value is set to 0x0000 and the Protocol Version is also set to 0x00. The BPDU Type field identifies two kinds of STP BPDUs: Configuration BPDUs (type 0x00) and Topology Change Notification BPDUs (type 0x80). The Flags field uses 2 bits out of 8 to handle topology change events: the Topology Change Acknowledgment flag and the Topology Change flag. Following the Flags, there is a series of fields identifying the root bridge, distance of the BPDU's sender from the root bridge, the sender bridge's own identifier, and the identifier of the port on the sender bridge that forwarded this BPDU. The MessageAge field is an estimation of the BPDU's age since it was originated by the root bridge. At the root bridge, it is set to 0. Any other switch will increment this value, usually by 1, before forwarding the BPDU further. The remaining lifetime of a BPDU after being received by a switch is MaxAge-MessageAge. Finally, the remaining fields carry the values of STP timers: MaxAge, HelloTime, ForwardDelay. These timer values always reflect the timer settings on the root switch. Timers configured on a nonroot switch are not used and would become effective only if the switch itself became the root switch.

Bridges and ports are identified by their IDs in BPDUs. Without discussing the exact format at this point, an object in STP that is called "identifier," or ID, always has a configurable part called the *priority*, and a fixed part that cannot be modified by management. Both bridges and ports have IDs with configurable priorities.

STP operation is based on the ability to compare any two arbitrary Configuration BPDUs and determine which one of them is better, or *superior*. The other BPDU is called *inferior*. To determine which BPDU out of a pair of BPDUs is superior, they are compared in the following sequence of values, looking for the first occurrence of a lower value:

- Root Bridge ID (RBID)
- Root Path Cost (RPC)

- Sender Bridge ID (SBID)
- Sender Port ID (SPID)
- Receiver Port ID (RPID; not included in the BPDU, evaluated locally)

First, the RBID value in both BPDUs is compared. If one of the BPDUs contains a lower RBID value, this BPDU is declared superior and the comparison process stops. Otherwise, both BPDUs carry the same RBID value and the RPC is compared. Again, if one of the BPDUs carries a lower RPC value, this BPDU is declared superior. In case both BPDUs carry an identical RPC value, the comparison process moves to the SBID. Should the SBID value be also found identical, the SPID will be compared. If even the SPID values in both BPDUs are the same, RPIDs of ports that received the same BPDU are compared. This very last step is very uncommon and would be seen in situations where a single BPDU was received by multiple ports of a single switch, possibly because of multiple connections to a hub or a non-STP switch being placed somewhere in between. In any case, precisely this capability of selecting a single superior BPDU out of a set of BPDUs is at the core of STP's capability to choose exactly one root bridge per a switched environment, exactly one Root Port on a nonroot bridge, and exactly one Designated Port for each connected network segment, as each of these roles is derived from the concept of a superior BPDU. Only Configuration BPDUs are compared; Topology Change Notification BPDUs do not convey information used to build a loop-free topology and are not compared. Therefore, whenever a comparison of BPDUs is discussed, it is implied that the BPDUs in question are Configuration BPDUs.

Additionally, an important fact to remember is that each port in STP stores (that is, remembers) the superior BPDU it has either sent or received. As you will see later, Root Ports and Blocking ports store the received BPDU sent by the "upstream" designated switch (because that BPDU is superior to the one that would be sent out from this port), while Designated Ports store their own sent BPDU (because that one is superior to any received BPDU). Essentially, each port stores the Designated Port's BPDU—whether it is the port itself that is Designated or it is a neighbor's port. Should a port store a received BPDU, it must be received again within a time interval of MaxAge-MessageAge seconds; otherwise it will expire after this period. This expiry is always driven by the timers in the BPDU, that is, according to timers of the root switch.

In the following sections, Configuration BPDUs will also be called simply Hello BPDUs or Hellos, as their origination is driven by the Hello timer.

Choosing Which Ports Forward: Choosing Root Ports and Designated Ports

To determine which ports forward and block, STP follows a three-step process, as listed in Table 3-2. Following the table, each of the three steps is explained in more detail.

Key Topic

Table 3-2 *Three Major 802.1D STP Process Steps*

Major Step	Description
Elect the root switch	The switch with the lowest bridge ID; the standard bridge ID is 2-byte priority followed by a MAC address unique to that switch.
Determine each switch's Root Port	The one port on each nonroot switch that receives the superior resulting BPDU from among all received BPDUs on all its ports.
Determine the Designated Port for each segment	When multiple switches connect to the same segment, this is the switch that forwards the superior BPDU from among all forwarded BPDUs onto that segment.

Electing a Root Switch

Only one switch can be the *root* of the spanning tree; to select the root, the switches hold an *election*. Each switch begins its STP logic by creating and sending an STP Hello bridge protocol data unit (BPDU) message, claiming itself to be the root switch. If a switch hears a *superior Hello* to its own Hello—namely, a Hello with a lower bridge ID—it stops claiming to be root by ceasing to originate and send Hellos. Instead, the switch starts forwarding the superior Hellos received from the superior candidate. Eventually, all switches except the switch with the lowest bridge ID cease to originate Hellos; that one switch wins the election and becomes the root switch.

The original IEEE 802.1D bridge ID held two fields:

- The 2-byte Priority field, which was designed to be configured on the various switches to affect the results of the STP election process.
- A 6-byte MAC Address field, which was included as a tiebreaker, because each switch's bridge ID includes a MAC address value that should be unique to each switch. As a result, some switch must win the root election.

The format of the original 802.1D bridge ID has been redefined in amendment 802.1t and since then integrated into 802.1D-2004. Figure 3-2 shows the original and new format of the bridge IDs.

Key Topic

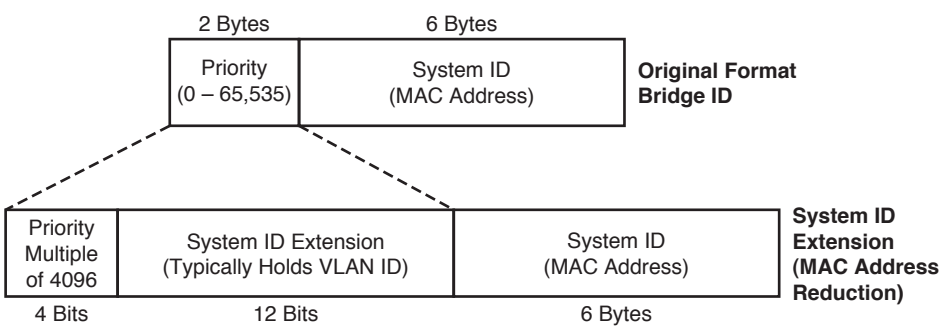

Figure 3-2 *IEEE 802.1D STP Bridge ID Formats*

The format was changed mainly because of the advent of multiple spanning trees as supported by Per VLAN Spanning Tree Plus (PVST+) and IEEE 802.1s Multiple Spanning Trees (MST). With the old-style bridge ID format, a switch's bridge ID for each STP instance (possibly one per VLAN) was identical if the switch used a single MAC address when building the bridge ID. Because VLANs cause a single physical switch to behave as multiple logical switches, having multiple STP instances with the same bridge ID was in violation of the 802.1D that required a distinct bridge ID for each switch. Vendors such as Cisco used a different MAC address for each VLAN when creating the old-style bridge IDs. This provided a different bridge ID per VLAN, but it consumed a large number of reserved MAC addresses in each switch.

The System ID Extension, originally described in IEEE 802.1t, allows a network to use multiple instances of STP, even one per VLAN, but without the need to consume a separate MAC address on each switch for each STP instance. The System ID Extension field allows the VLAN ID to be placed into what was formerly the last 12 bits of the Priority field. A switch can use a single MAC address to build bridge IDs and, with the VLAN number in the System ID Extension field, still have a unique bridge ID in each VLAN. The use of the System ID Extension field is also called *MAC address reduction*, because of the need for many fewer reserved MAC addresses on each switch. The use of the System ID Extension on a switch is indicated by the presence of the **spanning-tree extend system-id** command in global configuration mode. Older switches equipped with a larger reserve of MAC addresses allow this command to be removed, reverting to the old-style bridge IDs. Recent switches, however, do not allow this command to be removed even though it is displayed in the running config, and always use the System ID Extension.

Determining the Root Port

After the root switch is elected, the rest of the switches now need to determine their *Root Port (RP)*. The process proceeds as described in the following list:

1. The root switch creates and sends a Hello every Hello timer (2 seconds by default). This Hello contains the RBID and SBID fields set to the ID of the root, RPC set to 0, and SPID set to the identifier of the egress port.
2. Each nonroot switch receiving a BPDU on a particular port adds that port's cost to the RPC value in the received BPDU, yielding a *resulting* BPDU. Subsequently, the switch declares the port receiving the superior resulting BPDU as its Root Port.
3. Hellos received on the Root Port of a nonroot switch are forwarded through its remaining designated ports after updating the RPC, SBID, SPID, and MessageAge fields accordingly. Hellos received on other ports of a nonroot switch are processed but they are not forwarded.
4. Switches do not forward Hellos out Root Ports and ports that stabilize into a Blocking state. Hellos forwarded out these ports would be inferior (and therefore uninteresting) to Hellos originated by some neighboring switch's Designated Port on those segments.

The result of this process is that each nonroot switch chooses exactly one port as its Root Port, as there is always only a single received Hello that is superior over all other received Hellos. According to the sequence of compared fields in received Hellos when selecting a superior BPDU, a Root Port always provides the least-cost path toward the switch with the lowest Bridge ID (that is, the root switch). If there are multiple equal-cost paths, additional tiebreakers (SBID, SPID, RPID) will allow the receiving switch to always choose exactly one path in a deterministic fashion: first, port toward the neighbor with the lowest Bridge ID; then, if there are multiple links toward that neighbor, port connected to the neighbor's port with the lowest Port ID; and finally, if the same BPDU is received on multiple ports at once, the receiving port with the lowest Port ID.

In this sense, the STP operation is quite similar to the operation of the Routing Information Protocol (RIP), the simplest distance-vector routing protocol. Just like RIP, STP tries to find the least-cost path toward a particular destination, in this case, the root bridge, and has additional criteria to select a single path if there are multiple least-cost paths available. Hellos can be likened to RIP Update messages with RBID identifying the *destination*, RPC expressing the next hop's *metric* to the destination, SBID being the *next-hop identifier*, and SPID identifying the *next hop's interface*. Each time a Hello is received, the receiving switch can be thought to reevaluate its choice of a Root Port and updates the choice if necessary, just like a RIP router receives updates every 30 seconds and reevaluates its choice of least-cost paths to individual destinations. In fact, STP can be seen as a special case of a timer-driven distance-vector routing protocol, selecting exactly one path to exactly one particular destination, the root bridge. This makes STP similar to, though of course not entirely analogous to, RIP.

A switch must examine the RPC value in each Hello, plus the switch's STP port costs, to determine its least-cost path to reach the root. To do so, the switch adds the cost listed in the Hello message to the switch's port cost of the port on which the Hello was received. For example, Figure 3-3 shows the loop network design and details several STP cost calculations.

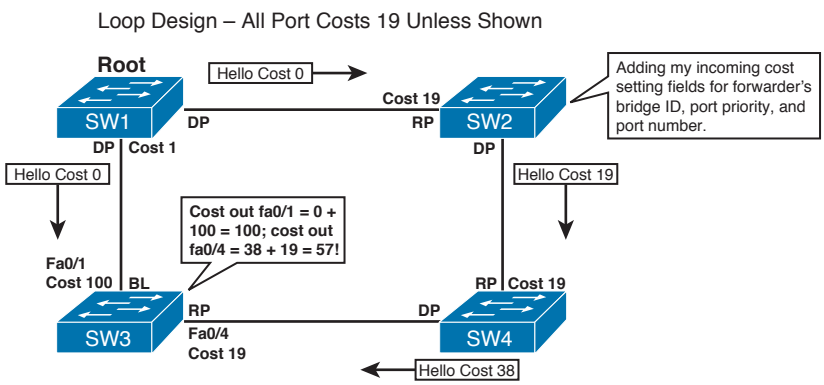

Figure 3-3 *Calculating STP Costs to Determine RPs*

In Figure 3-3, SW1 happened to become root, and is originating Hellos of cost 0. SW3 receives two Hellos, one with cost 0 and one with cost 38. However, SW3 must then calculate its cost to reach the root, which is the advertised cost (0 and 38, respectively) plus SW3's port costs (100 and 19, respectively). As a result, although SW3 has a direct link to SW1, the calculated cost is lower out interface Fa0/4 (cost 57) than it is out interface Fa0/1 (cost 100), so SW3 chooses its Fa0/4 interface as its RP.

Note Many people think of STP costs as being associated with a segment; however, the cost is actually associated with interfaces. Good design practices dictate using the same STP cost on each end of a point-to-point Ethernet segment, but the values can be different.

While the costs shown in Figure 3-3 might seem a bit contrived, the same result would happen with default port costs if the link from SW1 to SW3 were Fast Ethernet (default cost 19), and the other links were Gigabit Ethernet (default cost 4). Table 3-3 lists the default port costs according to various revisions of the IEEE 802.1D standard. Before 802.1D-1998, IEEE did not specify any recommended STP port cost values for different link speeds in their standard. Speeds shown in Table 3-3 were chosen by Cisco and used in its STP implementations of that time. The 802.1D-1998 revision of the standard provided a table of recommended values, but as the speeds of Ethernet links continued to increase dramatically, IEEE revised these recommended values again in its 802.1D-2004 revision of the standard. On recent Catalyst switches, the default costs correspond to the 802.1D-1998 version of the standard if PVST or Rapid PVST is used, and to the 802.1D-2004 version if MSTP is used. With PVST and Rapid PVST, the 802.1D-2004 costs can be activated using the **spanning-tree pathcost method long** global configuration command. By default, **spanning-tree pathcost method short** is configured, causing the switch to use the older revision of the costs.

Table 3-3 *Default Port Costs*

Port speed	Pre-802.1D-1998 Cost	802.1D-1998 Cost	802.1D-2004 Cost
10 Mbps	100	100	2000000
100 Mbps	10	19	200000
1 Gbps	1	4	20000
10 Gbps	1	2	2000

Determining the Designated Port

A converged STP topology results in only one switch forwarding Hellos onto each LAN segment. The switch that forwards Hellos onto a LAN segment is called the *designated switch* for that segment, and the port that it uses to forward frames onto that segment is

called the *Designated Port (DP)*. All remaining ports on a switch that have been determined as neither Root nor Designated will be moved to Blocking state. In the following text, they will be labeled as Non-Designated ports.

To win the right to be the DP, a switch must send superior Hellos onto the segment. For example, consider the segment between SW3 and SW4 in Figure 3-3 before the DP has been determined on that segment. SW3 would get Hellos directly from SW1, compute its cost to the root over that path, and then forward the Hello out its Fa0/4 interface to SW4, with RPC set to 100. Similarly, SW4 will forward a Hello with RPC of 38, as shown in Figure 3-3. SW4's port on this segment becomes the DP, as it sends superior Hellos because of their lower RPC value. Even after SW3 selects its Fa0/4 as the Root Port (as it receives superior resulting BPDUs from among all ports on SW3), any Hellos sent from SW3's Fa0/4 port would indicate the RPC of 57, still being inferior to SW4's Hellos.

Only the DP forwards Hellos onto a LAN segment. In the same example, SW4 keeps sending the Hellos with an RPC of 38 out the port, but SW3 stops sending its inferior Hellos. There would be no harm if SW3 continued to send its inferior BPDUs out its Fa0/1 and Fa0/4 ports, but because STP always cares only for superior BPDUs, this would be a waste of effort. Therefore, neither Root Ports nor ports in the Blocking state send BPDUs.

The tiebreakers during DP selection are the same as before: first, the switch with the least-cost path to the root identified by the lowest Bridge ID; then the neighboring switch with the lowest Bridge ID; and finally the port on the neighbor with the lowest Port ID.

To sum up the rules:

- The root switch is the switch that has the lowest Bridge ID in the topology.
- On each nonroot switch, a Root Port is the port receiving the best (that is, superior) resulting BPDUs from all received BPDUs on all ports. The adjective "resulting" refers to the addition of the port's cost to the BPDU's RPC value before comparing the received BPDUs.
- On each connected segment, a Designated Port is the port sending the best (that is, superior) BPDUs on the segment. No modifications to the BPDUs are performed; BPDUs are compared immediately.
- All ports that are neither Root Ports nor Designated Ports are superfluous in an active topology and will be put into the Blocking state.
- Configuration BPDUs are sent out only from Designated Ports. Root and Non-Designated ports do not emit Configuration BPDUs because they would be inferior to BPDUs of a Designated Port on this segment and hence ignored.

- Each port stores the best (that is, superior) BPDU it has received or sent itself. Designated Ports store the BPDU they send; Root and Blocking ports store the best BPDU they receive. The stored BPDU determines the role of the port and is used for comparisons.
- Received superior stored BPDUs will expire in MaxAge-MessageAge seconds if not received within this time period.

Converging to a New STP Topology

Although STP is very illustratively described in the three steps discussed earlier, this approach also gives an impression that after the three steps are completed, STP effectively goes dormant until a topology change occurs. Such impression would be incorrect, though. In reality, STP never stops working. With each received BPDU, a switch reevaluates its own choice of the root switch, Root Port, and Designated/Non-Designated Ports, effectively performing all three steps all over again. In a stable topology, received BPDUs do not change, and therefore, processing them yields the same results again and again. This is similar to the operation of the RIP that also never stops running—it's just that in a stable network which has converged, processing periodic received updates produces the same set of best paths, which gives off an impression that the protocol has done its job and has stopped. In reality, both STP and RIP continue running indefinitely, only in a stable and converged topology, each run produces the same results.

Of course, a topology in which STP runs can change over time, and STP has to react appropriately. In precise terms, for STP, a *topology change* is an event that occurs when

- A Topology Change Notification BPDU is received by a Designated Port of a switch
- A port moves to the Forwarding state and the switch has at least one Designated Port (meaning that it is not a standalone switch with just a Root Port connected to an upstream switch and no other connected ports)
- A port moves from Learning or Forwarding to Blocking
- A switch becomes the root switch

When a change to the topology occurs, the elementary reaction of switches that detect the topology change is to start originating BPDUs with appropriately updated contents, propagating the information to their neighbors. These neighbors will process the updated BPDUs, reevaluating their choice of the root switch, Root Port, and Designated/Non-Designated Ports with each received BPDU as usual, and forwarding the BPDU farther according to usual STP rules.

For an example, consider Figure 3-4, which shows the same loop network as in Figure 3-3. In this case, however, the link from SW1 to SW2 has just failed.

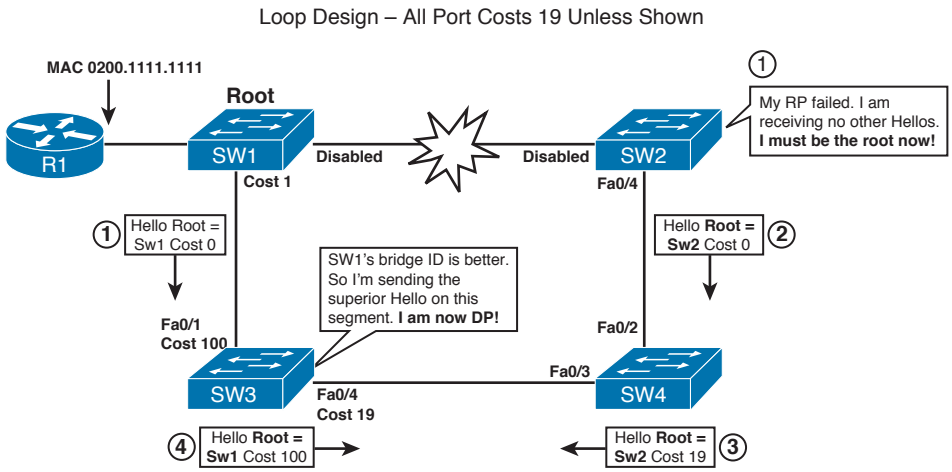

Figure 3-4 *Reacting to the Loss of Link Between SW1 and SW2*

The following list describes some of the key steps from Figure 3-4:

1. SW2's Root Port goes down. On SW2, the loss of a Root Port causes it to reelect its Root Port by choosing the port receiving superior resulting BPDUs. However, as the only remaining port is Fa0/4 connected to SW4, and SW4 did not send any BPDUs to SW2 from its Root Port Fa0/2, SW2 has no received BPDUs to choose from, and it will start considering itself a root switch, flooding its own Hellos through all its connected ports.

2. SW4 notices that the latest Hellos indicate a new root switch. However, these Hellos from SW2 received on SW4's Fa0/2 port, its current Root Port, are inferior to the BPDU stored on that port. When the link between SW1 and SW2 still worked, BPDUs arriving at SW4's Fa0/4 contained the SW1's Bridge ID as the RBID. After the link between SW1 and SW2 went down and SW2 started considering itself as the root bridge, its BPDUs arriving at SW4's Fa0/2 port contained SW2's Bridge ID as the RBID. However, SW2 has a higher Bridge ID than SW1; otherwise, it would be the root switch right away. Therefore, BPDUs claiming that SW2 is the root bridge are inferior to the BPDU stored on SW4's Fa0/2 that claims SW1 is the root bridge, and as a result, they are ignored until the BPDU stored on SW4's Fa0/2 expires. This expiry will take MaxAge-MessageAge, or 20–1=19 seconds. Until then, SW4 does not forward any BPDUs to SW3.

3. During the time SW4 receives inferior BPDUs from SW2 on its Fa0/2 port, it does not forward any BPDUs to SW3. As a result, SW3 ceases to receive BPDUs on its Fa0/4 port, which is its current Root Port. The BPDU stored on SW3's Fa0/4 port expires in MaxAge-MessageAge, or 20–2=18 seconds. After it expires, Fa0/4 becomes a Designated Port and moves to the Listening state. SW3 then searches for a new Root Port by looking for the superior received resulting BPDU, ultimately choosing Fa0/1 as its new port. Afterward, it will forward SW1's Hello out its Fa0/4 port after updating the necessary fields.

4. In the meantime, SW4 might have the BPDU expired from its Fa0/2, started accepting BPDUs from SW2, declared the Fa0/2 as its Root Port toward SW2, and started relaying the Hellos from SW2 to SW3. Even if that was the case, SW3 would treat these Hellos from SW4 as inferior because Hellos sent out from SW3's Fa0/4 claim that the root switch is SW1 having a lower Bridge ID than SW2. After SW4 receives the relayed Hello from SW3, it will learn about a better root switch than SW2, namely, SW1, and will choose its Fa0/3 as the Root Port. Afterward, it will forward the Hello out its Fa0/2 port.
5. After SW2 receives the forwarded Hello from SW4, it will also learn about SW1 being a better root switch than itself. Therefore, SW2 will stop considering itself as a root switch and will instead declare its Fa0/4 port as the Root Port, finally converging on the new loop-free topology.

Topology Change Notification and Updating the CAM

Simply updating the active topology by processing new BPDUs is not sufficient. When STP reconverges on a new active topology, some Content Addressable Memory (CAM) entries might be invalid (CAM is the Cisco term for what is more generically called the MAC address table, switching table, or bridging table on a switch). For example, before the link failure shown in Figure 3-4, SW3's CAM might have had an entry for 0200.1111.1111 (Router1's MAC address) pointing out Fa0/4 to SW4. Remember, at the beginning of the scenario described in Figure 3-4, SW3 was Blocking on its Fa0/1 interface back to SW1. When the link between SW1 and SW2 failed, SW3 would need to change its CAM entry for 0200.1111.111 to point out port Fa0/1.

STP is not a protocol that tries to find shortest paths toward individual MAC addresses, so it cannot be expected to fill the CAM tables with new correct entries. All STP can do is to instruct switches to age out unused entries prematurely, assuming that the unused entries are exactly those that need updating. Even if good entries are flushed from CAM tables, this does not impair basic connectivity—switches will flood frames to unknown destinations rather than dropping them.

To update the CAMs, two things need to occur:

- All switches need to be notified to time out their apparently unused CAM entries.
- Each switch needs to use a short timer, equivalent to the Forward Delay timer (default 15 seconds), to time out the CAM entries.

A topology change can start as a highly localized event—a port becoming Forwarding or transitioning from Learning or Forwarding to Blocking on a particular single switch. The information about this change must nevertheless be propagated to all switches in the topology. Therefore, a switch that detects a topology change must notify the root switch, and the root switch in turn can notify all switches in the topology. (Recall that it is the root switch's Hello that is propagated throughout the network to all switches; a nonroot switch has no way of sending its own Configuration BPDU to all remaining switches in a topology because that BPDU would be inferior, and thus ignored, by possibly many switches.) To do so, a switch detecting a topology change notifies the root switch using

a *Topology Change Notification (TCN)* BPDU. The TCN goes up the tree to the root. After that, the root notifies all the rest of the switches. The process is illustrated in Figure 3-5 and runs as follows:

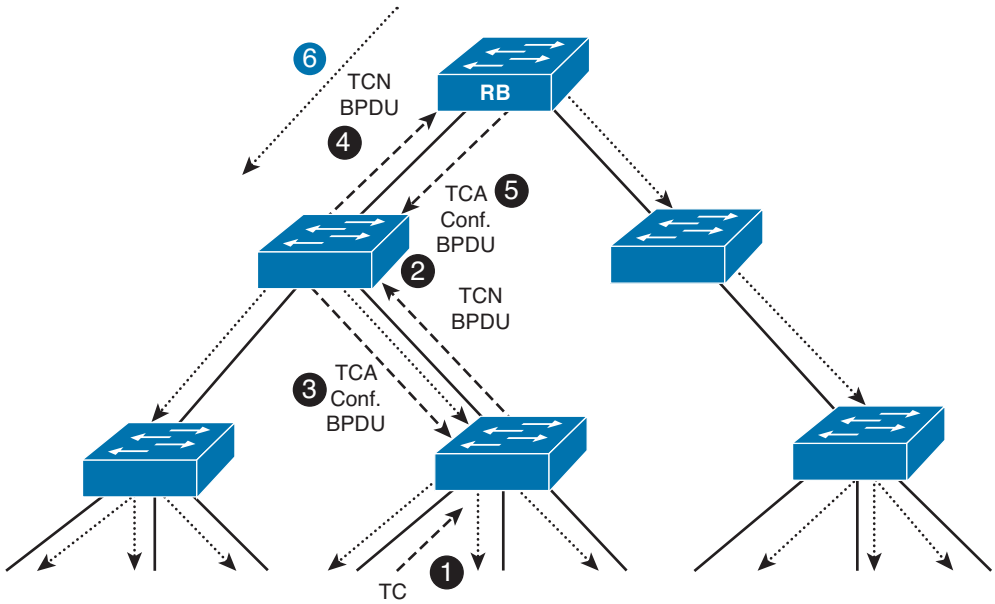

Figure 3-5 *Propagating Information About Topology Change*

Key Topic

1. A topology change event occurs on a port of a switch.
2. After detecting the event, the switch sends a TCN BPDU out its Root Port; it repeats this message every Hello time until it is acknowledged.
3. The next designated switch receiving that TCN BPDU sends back an acknowledgment through its next forwarded Hello BPDU by marking the Topology Change Acknowledgment (TCA) bit in the Flags field of the Hello.
4. The designated switch on the segment in the second step repeats the first two steps, sending a TCN BPDU out its Root Port, and awaits acknowledgment from the designated switch on that segment.
5. After the TCN arrives at the root switch, it also acknowledges its arrival through sending a BPDU with the Topology Change Acknowledgment bit set through the port through which the TCN BPDU came in. At this point, the root switch has been informed about a topology change that occurred somewhere in the network.
6. For the next MaxAge+ForwardDelay seconds, the root switch will originate BPDUs with the Topology Change (TC) bit set, instructing all switches to shorten the aging time for CAM entries to ForwardDelay seconds.

By each successive switch repeating Steps 2 and 3, eventually the root receives a TCN BPDU. After it is received, the root sets the *Topology Change (TC)* flag on the next several Hellos (during the next MaxAge+ForwardDelay seconds), which are forwarded to all switches in the network, notifying them that a change has occurred. A switch receiving a Hello BPDU with the TC flag set uses the short (ForwardDelay time derived from the value in the received BPDU, set by the root switch) timer to time out unused entries in the CAM.

Transitioning from Blocking to Forwarding

When STP reconverges to a new, stable topology, some ports that were Blocking might have been designated as DP or RP, so these ports need to be in a Forwarding state. However, the transition from Blocking to Forwarding state cannot be made immediately without the risk of causing loops.

To transition to Forwarding state but also prevent temporary loops, a switch first puts a formerly Blocking port into *Listening* state, and then into *Learning* state, with each state lasting for the length of time defined by the ForwardDelay timer (by default, 15 seconds). Table 3-4 summarizes the key points about all the 802.1D STP port states.

Key Topic

Table 3-4 *IEEE 802.1D Spanning Tree Interface States*

State	Forwards Data Frames?	Learns Source MACs of 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

In summary, when STP logic senses a change in the topology, it converges, possibly picking different ports as RP, DP, or neither. Any switch changing its RPs or DPs sends a TCN BPDU to the root at this point. For the ports newly designated as RP or DP, 802.1D STP first uses the Listening and Learning states before reaching the Forwarding state. (The transition from Forwarding to Blocking can be made immediately.)

Per-VLAN Spanning Tree and STP over Trunks

If only one instance of STP was used for a switched network with redundant links but with multiple VLANs, several ports would be in a Blocking state, unused under stable conditions. The redundant links would essentially be used for backup purposes.

always sends BPDUs each Hello interval, whether it is Root, Designated, Alternate, or Backup. BPDUs thus essentially become a Hello mechanism between pairs of interconnected switches. A Bridge Assurance–protected port is absolutely required to receive BPDUs. If no BPDUs are received, the port will be but into a *BA-inconsistent* blocking state until it starts receiving BPDUs again. Apart from unidirectional links, Bridge Assurance also protects against loops caused by malfunctioning switches that completely stop participating in RPVST+/MST (entirely ceasing to process and send BPDUs) while opening all their ports. At the time of this writing, Bridge Assurance was supported on selected Catalyst 6500 and Nexus 7000 platforms. Configuring it on Catalyst 6500 Series requires activating it both globally using **spanning-tree bridge assurance** and on ports on STP point-to-point link types toward other switches using the **spanning-tree portfast network** interface command. The neighboring device must also be configured for Bridge Assurance.

The Dispute mechanism is yet another and standardized means to detect a unidirectional link. Its functionality is based on the information encoded in the Flags field of RST and MST BPDUs, namely, the role and state of the port forwarding the BPDU. The principle of operation is very simple: If a port receives an inferior BPDU from a port that claims to be Designated Learning or Forwarding, it will itself move to the Discarding state. Cisco has also implemented the Dispute mechanism into its RPVST+. The Dispute mechanism is not available with legacy STP/PVST+, as these STP versions do not encode the port role and state into BPDUs. The Dispute mechanism is an integral part of RSTP/MST and requires no configuration.

Configuring and Troubleshooting EtherChannels

EtherChannel, also known as Link Aggregation, is a widely supported and deployed technology used to bundle several physical Ethernet links interconnecting a pair of devices into a single logical communication channel with increased total throughput. After an EtherChannel is established, it is represented to the devices as a single logical interface (called *Port-channel* in Cisco parlance), utilizing the bandwidth of all its member links. This allows for traffic load sharing between the member links, taking advantage of their combined bandwidth. Also, should a link in an EtherChannel bundle fail, the traffic will be spread over remaining working links without further influencing the state of the logical interface. Control plane protocols that see only the logical Port-channel interface and not its underlying physical members, such as STP, will only notice a change in the interface's bandwidth parameter (if not configured statically using the **bandwidth** command). The reaction to a failure or addition of a member link is therefore significantly more graceful than a reaction to a loss or reestablishment of a standalone link.

Load Balancing Across Port-Channels

EtherChannel increases the available bandwidth by carrying multiple frames over multiple links. A single Ethernet frame is always transmitted over a single link in an EtherChannel bundle. A hashing function performed over selected frames' address fields produces a number identifying the physical link in the bundle over which the frame will

be forwarded. The sequence of frames having an identical value in a particular address field (or a set of fields) fed into the hashing function is called a *conversation* or simply a *flow*. This hashing function is deterministic, meaning that all frames in a single flow produce the same hash value, and are therefore forwarded over the same physical link. Hence, the increase in the available bandwidth is never experienced by a single flow; rather, multiple flows have a chance of being distributed over multiple links, achieving higher aggregated throughput. The fact that a single flow is carried by a single link and thus does not benefit from a bandwidth increase can be considered a disadvantage; however, this approach also prevents frames from being reordered. This property is crucial, as EtherChannel—being a transparent technology—must not introduce impairments that would not be seen on plain Ethernet.

Load-balancing methods differ depending on the model of switch and software revision. Generally, load balancing is based on the contents of the Layer 2, 3, and/or 4 headers. If load balancing is based on only one header field in the frame, that single field is fed into the hashing function. If more than one header field is used, first, an XOR operation between the selected fields is used and only the result of this XOR is fed into the hashing function. The details of hashing functions in use are not public and can vary between different switch platforms.

For the best balancing effect, the header fields on which balancing is based need to vary among the mix of frames sent over the Port-channel. For example, for a Layer 2 Port-channel connected to an access layer switch, most of the traffic going from the access layer switch to the distribution layer switch is probably going from clients to the default router. So most of the frames have different source MAC addresses but the same destination MAC address. For packets coming back from a distribution switch toward the access layer switch, many of the frames might have a source address of that same router, with differing destination MAC addresses. So, you could balance based on source MAC at the access layer switch and based on destination MAC at the distribution layer switch—or balance based on both fields on both switches. The goal is simply to use a balancing method for which the fields in the frames vary.

The **port-channel load-balance** *type* global level command sets the type of load balancing. The *type* options include using source and destination MAC, IP addresses, and TCP and UDP ports—either a single field or both the source and destination. Because this command is global, it influences the operation of all EtherChannel bundles on a switch. Devices on opposite ends of an EtherChannel bundle can, and often do, use different load-balancing algorithms.

The maximum number of active member links in an EtherChannel bundle is eight. This limit is reasonable, considering that Ethernet variants differ in speed by orders of tens (10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps, 100 Gbps). More than eight links in an EtherChannel bundle is simply closing in on the next faster Ethernet variant, so it is reasonable to consider using a faster Ethernet variant in such cases instead. On many Catalyst switch platforms, the hashing function therefore produces a 3-bit result in the range of 0–7 whose values are assigned to the individual member links. With eight physical links in a bundle,

each link is assigned exactly one value from this range. If there are fewer physical links, some of the links will be assigned multiple values from this range, and as a result, some of the links will carry more traffic than the others. Table 3-7 describes the traffic amount ratios (P*n* denotes the *n*th port in a bundle).

Table 3-7 *Load Spread Ratios with Different Port Numbers in EtherChannel*

Number of Ports in the EtherChannel	Load-Balancing Ratios
8	P1:P2:P3:P4:P5:P6:P7:P8 → 1:1:1:1:1:1:1:1
7	P1:P2:P3:P4:P5:P6:P7:P1 → 2:1:1:1:1:1:1
6	P1:P2:P3:P4:P5:P6:P1:P2 → 2:2:1:1:1:1
5	P1:P2:P3:P4:P5:P1:P2:P3 → 2:2:2:1:1
4	P1:P2:P3:P4:P1:P2:P3:P4 → 2:2:2:2
3	P1:P2:P3:P1:P2:P3:P1:P2 → 3:3:2
2	P1:P2:P1:P2:P1:P2:P1:P2 → 4:4

Under ideal conditions, the traffic distribution across member links will be equal only if the number of links is eight, four, or two. With the eight resulting values from a 3-bit hash function, each value represents 1/8=12.5% of the traffic. The spread of the traffic by multiples of 12.5% is quite coarse. The indicated ratios can also be computed by using DIV and MOD operations: For example, with three links in a bundle, each link will be assigned 8 DIV 3 = 2 resulting hash values, plus 8 MOD 3 = 2 links will be handling an additional hash result value, yielding a ratio of (2+1):(2+1):2 = 3:3:2, or 37.5% : 37.5% : 25%.

On other Cisco switch platforms, an 8-bit hash result is used although the EtherChannel is still limited to a maximum of eight links. Because the hash value allows for 256 possible results, each value represents a mere 1/256 = 0.390625% of the traffic. The spread of the traffic across links in a bundle is thus much more fine-grained. With three links, each of them would be assigned 256 DIV 3 = 85 resulting hash values, plus a 256 MOD 3 = 1 link would be handling an additional hash result value. So the traffic split ratio would be 86:85:85, or approximately 33.6% : 33.2% : 33.2%, much more balanced than 3:2:2.

It is sometimes incorrectly stated that a Port-channel can only operate with two, four, or eight links. That is incorrect—a Port-channel can operate with any number of links between one and eight, inclusive. The spreading of total traffic across links can be uneven, however, if the number of links is not a power of 2, as previously explained.

Port-Channel Discovery and Configuration

When you are adding multiple ports to a particular Port-channel *on a single switch*, several configuration items must be identical, as follows:

Key Topic

- Same speed and duplex settings.
- Same operating mode (trunk, access, dynamic).
- If not trunking, same access VLAN.
- If trunking, same trunk type, allowed VLANs, and native VLAN.
- On a single switch, each port in a Port-channel must have the same STP cost per VLAN on all links in the Port-channel.
- No ports can have SPAN configured.

Some of these limitations can change over time—it is recommended to consult the Configuration Guide for your particular switch platform and IOS version to stay up to date.

When a new Port-channel is created, an **interface Port-channel** is automatically added to the configuration. This interface inherits the configuration of the first physical interface added to the Port-channel, and the configuration of all other physical interfaces added to the same Port-channel will be compared to the **interface Port-channel** configuration. If they differ, the physical interface will be considered as *suspended* from the Port-channel, and it will not become a working member until its configuration is made identical to that of the Port-channel interface. Configuration changes performed on the **interface Port-channel** apply only to nonsuspended member ports; that is, commands applied to the Port-channel interface are pushed down only to those physical member ports whose configuration matched the **interface Port-channel** configuration before making the change. Therefore, reentering the configuration on the Port-channel interface in hopes of unifying the configuration of all member ports will not have an effect on those ports whose current configuration differs from that of the Port-channel interface. It is therefore recommended to adhere to the following guidelines when configuring Port-channels:

Key Topic

- Do not create the interface Port-channel manually before bundling the physical ports under it. Let the switch create it and populate its configuration automatically.
- On the other hand, when removing a Port-channel, make sure to manually remove the **interface Port-channel** from the running config so that its configuration does not cause issues when a Port-channel with the same number is re-created later.
- Be sure to make the configuration of physical ports identical before adding them to the same Port-channel.
- If a physical port's configuration differs from the **interface Port-channel** configuration, correct the physical port's configuration first. Only then proceed to perform changes to the Port-channel interface configuration.

- A Port-channel interface can either be Layer 2 (switched) or Layer 3 (routed), depending on whether the physical bundled ports are configured as Layer 2 (**switchport**) or Layer 3 (**no switchport**). After a Port-channel has been created with a particular operating level, it is not possible to change it to the other mode without re-creating it. If it is necessary to change between Layer 2 and Layer 3 levels of operation, the Port-channel must be removed from configuration and re-created after the physical ports are reconfigured for the required level of operation. It is possible, though, to combine the Layer 2 Port-channel on one switch with the Layer 3 Port-channel on another, although not necessarily a best practice.
- Whenever resolving an issue with err-disabled ports under a Port-channel interface, be sure to shut down both the physical interfaces and the **interface Port-channel** itself. Only then try to reactivate them. If the problem persists, it is recommended to remove the Port-channel altogether from the configuration, unbundling the ports as a result, and re-create the Port-channel.

You can statically configure interfaces to be in a Port-channel by using the **channel-group** *number* **mode on** interface subcommand. You would simply put the same command under each of the physical interfaces inside the Port-channel, using the same Port-channel number. This configuration forces the ports to become members of the same Port-channel without negotiating with the neighboring switch. This way of creating a Port-channel is strongly discouraged, though. If one switch considers multiple physical ports to be bundled under a single Port-channel while the neighboring switch still treats them as individual or assigns them into several bundles, permanent switching loops can occur. Also, this static Port-channel configuration is not capable of detecting whether the bundled ports are all connected to the same neighboring device. Having individual ports in a single Port-channel connect to different neighboring switches can again lead to permanent switching loops. To understand how the switch loop ensues, consider the topology shown in Figure 3-16.

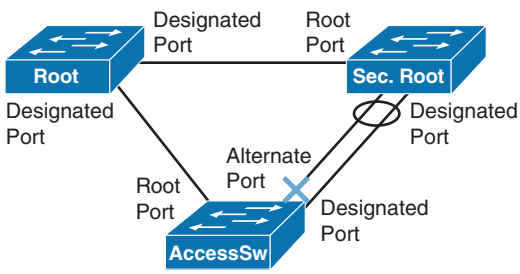

Figure 3-16 *Permanent Switching Loop in a Misconfigured EtherChannel*

In this topology, the ports on the Secondary Root switch toward AccessSw have already been bundled in a Port-channel using **mode on**, and the switch uses them as a single EtherChannel right away, without negotiating with AccessSw. However, AccessSw is not yet configured for Port-channel on these ports, and treats them as individual links.

Key Topic

Because Port-channel interfaces are treated as single ports by STP, only a single BPDU is sent for the entire Port-channel interface, regardless of how many physical links are bundled. This BPDU is also subject to the hashing function and forwarded over a single link in the entire Port-channel bundle. Assuming that the Secondary Root has the second-lowest priority in this network and that the BPDUs are forwarded over the left link toward AccessSw, the corresponding port on AccessSw is Alternate Discarding. However, the AccessSw port on the right link is not receiving any BPDUs and becomes Designated Forwarding as a result. Even though such a port sends BPDUs, they will be ignored by the Secondary Root switch because they are inferior to its own BPDUs. Hence, a permanent switching loop is created. This is also the reason why a switch shuts down all physical ports when **no interface Port-channel** is issued—to prevent switching loops when Port-channel configuration is being removed.

Note that if using RSTP/MST, the Dispute mechanism would detect this problem and put the Port-channel on the Secondary Root switch to the Discarding state, preventing this loop. In addition, Cisco has implemented yet another prevention mechanism called STP *EtherChannel Misconfig Guard* on its switches. This mechanism makes an assumption that if multiple ports are correctly bundled into a Port-channel at the neighbor side, all BPDUs received over links in this Port-channel must have the same source MAC address in their Ethernet header, as the Port-channel interface inherits the MAC address of one of its physical member ports. If BPDUs sourced from different MAC addresses are received on a Port-channel interface, it is an indication that the neighbor is still treating the links as individual, and the entire Port-channel will be err-disabled. Note that the detection abilities of the EtherChannel Misconfig Guard are limited. In the topology in Figure 3-16, this mechanism will not help because the Secondary Switch receives just a single BPDU from AccessSw over the right link, and has no other BPDU to compare the source MAC address to. The mechanism would be able to detect a problem if, for example, there were three or more links between Secondary Root and AccessSw, or if the two existing links were bundled at the AccessSw instead of Secondary Root. The EtherChannel Misconfig Guard is active by default and can be deactivated using the **no spanning-tree etherchannel guard misconfig** global configuration command.

It is therefore strongly recommended to use a dynamic negotiation protocol to allow switches to negotiate the creation of a Port-channel and verify whether the links are eligible for bundling. Those protocols are the Cisco-proprietary *Port Aggregation Protocol (PAgP)* and the open IEEE 802.1AX (formerly 802.3ad) *Link Aggregation Control Protocol (LACP)*. Both protocols offer relatively similar features though they are mutually incompatible. On a common Port-channel, both switches must use the same negotiation protocol; different Port-channel interfaces can use different negotiation protocols. Using LACP is generally preferred because of its open nature and widespread support.

PAgP allows a maximum of eight links in a Port-channel. A switch will refuse to add more than eight links to a PAgP Port-channel. On current Catalyst switches, PAgP has no user-configurable parameters apart from the frequency of sending PAgP messages. This frequency is configurable on a per-port basis using the **pagp timer** { **normal | fast** } command; **normal** frequency is 30 seconds after the Port-channel is established, and **fast** is a

1-second frequency. Other available commands related to PAgP have no effect on the forwarding behavior of the switch, and are kept only for backward compatibility with very old switches.

With LACP, a maximum of 16 links can be placed into a Port-channel. Out of these links, at most eight links will be active members of the Port-channel. Remaining links will be put into a so-called Standby (sometimes also called Hot-Standby) mode. If an active link fails, one of the Standby links will be used to replace it. A single switch is in charge of selecting which Standby link will be promoted to the active state—it is the switch with the lower LACP System ID that consists of a configurable priority and the switch MAC address (the same concept as in STP). If there are multiple Standby links, the switch in control will choose the link with the lowest Port ID that again consists of a configurable priority and the port number. LACP priority of a switch can be globally configured using the **lacp system-priority** command, and the priority of a port can be set up using the **lacp port-priority** command. Both priorities can be configured in the range of 0–65535.

To dynamically form a Port-channel using PAgP, you still use the **channel-group** command, with a mode of **auto** or **desirable**. To use LACP to dynamically create a Port-channel, use a mode of **active** or **passive**. Table 3-8 lists and describes the modes and their meanings.

Key Topic

Table 3-8 *PAgP and LACP Configuration Settings and Recommendations*

PAgP Setting	LACP 802.1AX Setting	Action
auto	passive	Uses PAgP or LACP, but waits on the other side to send the first PAgP or LACP message
desirable	active	Uses PAgP or LACP and initiates the negotiation

Note Using **auto** (PAgP) or **passive** (LACP) on both switches prevents a Port-channel from forming dynamically. Cisco recommends the use of **desirable** mode (PAgP) or **active** mode (LACP) on ports that you intend to be part of a Port-channel on both devices.

As remembering the mode keywords and the protocol they refer to (**desirable/auto** for PAgP; **active/passive** for LACP) can be awkward, Cisco implemented the helper command **channel-protocol** { **pagp | lacp** } that can be used on physical interfaces to limit the accepted mode keywords to the stated negotiation protocol. In other words, entering **channel-protocol pagp** will allow the subsequent use of **desirable** or **auto** modes only; the **active**, **passive**, and **on** modes will be rejected. Similarly, using **channel-protocol lacp** will only permit the subsequent use of **active** or **passive** modes; the **desirable**, **auto**, and **on** modes will be rejected.

Note A common misunderstanding is that the **channel-protocol** command can be used in combination with the **on** mode to start a particular negotiation protocol. This is incorrect. The **channel-protocol** command only causes the CLI to refuse any mode keywords that do not imply running the chosen negotiation protocol.

When PAgP or LACP negotiate to form a Port-channel, the messages include the exchange of key information that allows detecting whether all links to be bundled under a single Port-channel are connected to the same neighbor and whether the neighbor is also willing to bundle them under a single Port-channel. These values include system IDs of both interconnected devices, identifiers of physical ports, and aggregation groups these ports fall under. It is sometimes believed that PAgP and LACP carry detailed information about individual port settings; that is incorrect. While PAgP and LACP make sure that the links to be bundled are all connected to the same neighboring switch and that both switches are willing to bundle them into a common Port-channel, they are neither capable nor supposed to verify whether ports on opposite sides of bundled links are otherwise identically configured.

Note PAgP and LACP verify only whether the links to be bundled are consistently connected to the same neighboring device and are to be bundled into the same link aggregation group. However, neither of these protocols performs checks on whether the ports on this switch and its neighbor are configured identically with respect to their operating mode, allowed VLANs, native VLAN, encapsulation, and so on.

When PAgP or LACP completes the process, a new Port-channel interface exists and is used as if it were a single port for STP purposes, with balancing taking place based on the global load-balancing method configured on each switch.

Troubleshooting Complex Layer 2 Issues

Troubleshooting is one of the most challenging aspects of CCIE study. The truth is, we can't teach you to troubleshoot in the pages of a book; only time and experience bring strong troubleshooting skills. We can, however, provide you with two things that are indispensable in learning to troubleshoot effectively and efficiently: process and tools. The focus of this section is to provide you with a set of Cisco IOS–based tools, beyond the more common ones that you already know, as well as some guidance on the troubleshooting process for Layer 2 issues that you might encounter.

In the CCIE Routing and Switching lab exam, you will encounter an array of troubleshooting situations that require you to have mastered fast, efficient, and thorough troubleshooting skills. In the written exam, you'll need a different set of skills—mainly, the knowledge of troubleshooting techniques that are specific to Cisco routers and switches, and the ability to interpret the output of various **show** commands and possibly debug output. You can also expect to be given an example along with a problem statement. You

CHAPTER 6

IP Forwarding (Routing)

This chapter begins with coverage of the details of the forwarding plane—the actual forwarding of IP packets. This process of forwarding IP packets is often called *IP routing*, or simply *routing*. Also, many people also refer to IP routing as the *data plane*, meaning the plane (topic) related to the end-user data.

Chapters 7 through 11 cover the details of the IP *control plane*. In contrast to the term *data plane*, the control plane relates to the communication of control information—in short, routing protocols like OSPF and BGP. These chapters cover the routing protocols on the exam, plus an additional chapter on redistribution and route summarization.

"Do I Know This Already?" Quiz

Table 6-1 outlines the major headings in this chapter and the corresponding "Do I Know This Already?" quiz questions.

Table 6-1 *"Do I Know This Already?" Foundation Topics Section-to-Question Mapping*

Foundation Topics Section	Questions Covered in This Section	Score
IP Forwarding	1–6	
Multilayer Switching	7–9	
Policy Routing	10–11	
Total Score		

To best use this pre-chapter assessment, remember to score yourself strictly. You can find the answers in Appendix A, "Answers to the 'Do I Know This Already?' Quizzes."

1. What command is used to enable CEF globally for IPv4 packets?

   a. enable cef

   b. ip enable cef

   c. ip cef

   d. cef enable

   e. cef enable ip

   f. cef ip

2. What command is used to enable CEF globally for IPv6 packets?
   a. **enable cef6**
   b. **ipv6 enable cef**
   c. **ipv6 cef**
   d. **ip cef** (the command automatically enables CEF for IPv4 and IPv6)
3. Can CEF for IPv6 be enabled independently of CEF for IPv4?
   a. Yes
   b. No
4. Which of the following triggers an update to a CEF FIB?
   a. Receipt of an ICMPv6 Neighbor Advertisement message with previously unknown information
   b. Receipt of a LAN ARP reply message with previously unknown information
   c. Addition of a new route to the IP routing table by EIGRP
   d. Addition of a new route to the IP routing table by adding an **ip route** command
   e. The removal of a route from the IP routing table by EIGRP
5. Which of the following triggers an update to a CEF adjacency table?
   a. Receipt of a CDP multicast on the PVC connected to Router1
   b. Receipt of an ARP response with previously unknown information
   c. Receipt of a packet that needs to be routed to another router over a point-to-point interface
   d. Receipt of an ICMPv6 Neighbor Advertisement with previously unknown information
6. Which of the following packet-switching paths is considered to be the slowest?
   a. Process Switching
   b. Fast Switching
   c. Route Cache
   d. Cisco Express Forwarding
7. Which of the following commands is used on a Cisco IOS Layer 3 switch to use the interface as a *routed interface* instead of a *switched interface*?
   a. **ip routing** or **ipv6 unicast-routing** global command
   b. **ip routing** or **ipv6 unicast-routing** interface subcommand
   c. **ip address** interface subcommand
   d. **switchport mode routed** interface subcommand
   e. **no switchport** interface subcommand

8. On a Cisco Catalyst 3560 switch, the first line of the output of a **show interface vlan 55** command lists the state as "Vlan 55 is down, line protocol is down." Which of the following might be causing that state to occur?

   a. VLAN interface has not been **no shut** yet.

   b. The **ip routing** global command is missing from the configuration.

   c. On at least one interface in the VLAN, a cable that was previously plugged in has been unplugged.

   d. VTP mode is set to transparent.

   e. The VLAN has not yet been created on this switch, or is not in the active state.

9. On a Cisco Catalyst 3560 switch, the first line of the output of a **show interface vlan 55** command lists the state as "Vlan 55 is up, line protocol is down." Which of the following might be causing that state to occur?

   a. VLAN interface has not been **no shut** yet.

   b. The **ip routing** global command is missing from the configuration.

   c. There is no switch port on the switch with this VLAN allowed and in the STP forwarding state.

   d. STP has been administratively deactivated for this VLAN.

   e. The VLAN has not yet been created on this switch, or is not in the active state.

10. Imagine a route map used for policy routing, in which the route map has a **set default interface serial0/0** command. Serial0/0 is a point-to-point link to another router. A packet arrives at this router, and the packet matches the policy routing **route-map** clause whose only **set** command is the one just mentioned. Which of the following general characterizations is true?

    a. The packet will be routed out interface s0/0; if s0/0 is down, it will be routed using the default route from the routing table.

    b. The packet will be routed using the default route in the routing table; if there is no default, the packet will be routed out s0/0.

    c. The packet will be routed using the best match of the destination address with the routing table; if no match is found, the packet will be routed out s0/0.

    d. The packet will be routed out interface s0/0; if s0/0 is down, the packet will be discarded.

11. Router1 has an fa0/0 interface and two point-to-point WAN links back to the core of the network (s0/0 and s0/1, respectively). Router1 accepts routing information only over s0/0, which Router1 uses as its primary link. When s0/0 fails, Router1 uses policy routing to forward the traffic out the relatively slower s0/1 link. Which of the following **set** commands in Router1's policy routing route map could have been used to achieve this function?

a. **set ip default next-hop**

b. **set ip next-hop**

c. **set default interface**

d. **set interface**

Foundation Topics

IP Forwarding

IP forwarding, or *IP routing*, is the process of receiving an IP packet, making a decision of where to send the packet next, and then forwarding the packet. The forwarding process needs to be relatively simple, or at least streamlined, for a router to forward large volumes of packets. Ignoring the details of several Cisco optimizations to the forwarding process for a moment, the internal forwarding logic in a router works basically as shown in Figure 6-1.

Key Topic

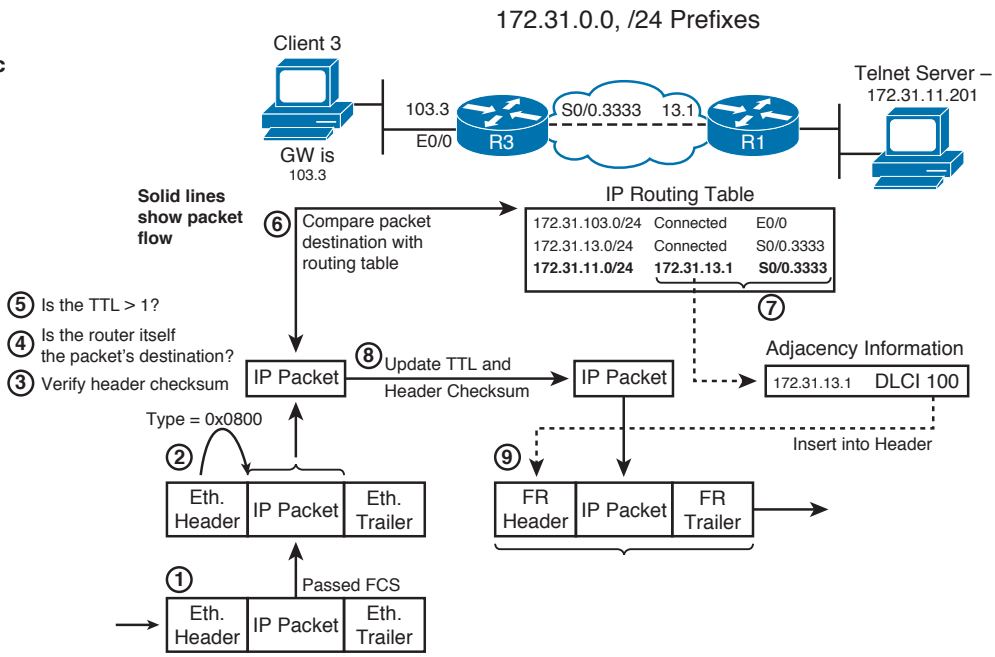

Figure 6-1 *Forwarding Process at Router3, Destination Telnet Server*

The following list summarizes the key steps shown in Figure 6-1:

Key Topic

1. A router receives the frame and checks the received frame check sequence (FCS); if errors occurred, the frame is discarded. The router makes no attempt to recover the lost packet.
2. If no errors occurred, the router checks the Ethernet Type field for the packet type and extracts the packet. The Data Link header and trailer can now be discarded.
3. Assuming an IPv4 packet, its header checksum is first verified. In case of mismatch, the packet is discarded. With IPv6 packets, this check is skipped, as IPv6 headers do not contain a checksum.

4. If the header checksum passed, the router checks whether the destination IP address is one of the addresses configured on the router itself. If it does, the packet has just arrived at its destination. The router analyzes the Protocol field in the IP header, identifying the upper-layer protocol, and hands the packet's payload over to the appropriate upper-protocol driver.
5. If the destination IP address does not match any of the router's configured addresses, the packet must be routed. The router first verifies whether the TTL of the packet is greater than 1. If not, the packet is dropped and an ICMP Time Exceeded message is sent to the packet's sender.
6. The router checks its IP routing table for the most specific prefix match of the packet's destination IP address.
7. The matched routing table entry includes the outgoing interface and next-hop router. This information is used by the router to look up the next-hop router's Layer 2 address in the appropriate mapping table, such as ARP, IP/DLCI, IP/VPI-VCI, dialer maps, and so on. This lookup is needed to build a new Data Link frame and optionally dial the proper number.
8. Before creating a new frame, the router updates the IP header TTL or Hop Count field, requiring a recomputation of the IPv4 header checksum.
9. The router encapsulates the IP packet in a new Data Link header (including the destination address) and trailer (including a new FCS) to create a new frame.

The preceding list is a generic view of the process. Next, a few words on how Cisco routers can optimize the routing process by using Cisco Express Forwarding (CEF).

Process Switching, Fast Switching, and Cisco Express Forwarding

Steps 6 and 7 from the generic routing logic shown in the preceding section are the most computation-intensive tasks in the routing process. A router must find the best route to use for every packet, requiring some form of table lookup of routing information. Also, a new Data Link header and trailer must be created, and the information to put in the header (like the destination Data Link address) must be found in another table.

Cisco has created several different methods to optimize the forwarding processing inside routers, termed *switching paths*. This section examines the two most likely methods to exist in Cisco router networks today: fast switching and CEF.

With fast switching, the first packet to a specific destination IP address is *process switched*, meaning that it follows the same general algorithm as shown in Figure 6-1. With the first packet, the router adds the results of this daunting lookup to the *fast-switching cache*, sometimes called the *route cache*, organized for fast lookups. The cache contains the destination IP address, the next-hop information, and the data-link header information that needs to be added to the packet before forwarding (as in Step 6 in Figure 6-1). Future packets to the same destination address match the cache entry, so it takes the router less time to process and forward the packet, as all results are already stored in the cache. This approach is also sometimes termed *route once, forward many times*.

Although it is much better than process switching, fast switching has significant drawbacks. The first packet must be process switched, because an entry can be added to the cache only when a packet is routed and the results of its routing (next hop, egress interface, Layer 2 rewrite information) are computed. A huge inflow of packets to destinations that are not yet recorded in the route cache can have a detrimental effect on the CPU and the router's performance, as they all need to be process switched. The cache entries are timed out relatively quickly, because otherwise the cache could get overly large as it has an entry per each destination address, not per destination subnet/prefix. If the routing table or Layer 3–to–Layer 2 tables change, parts of the route cache must be invalidated rather than updated, causing packets for affected destinations to become process switched again. Also, load balancing can only occur per destination with fast switching. Overall, fast switching was a great improvement at the time it was invented, but since that time, better switching mechanisms have been developed. One of them, Cisco Express Forwarding (CEF), has become the major packet-forwarding mechanism in all current Cisco IP routing implementations, with fast switching becoming practically unused. The support for unicast fast switching has therefore been discontinued and removed from IOS Releases 12.2(25)S and 12.4(20)T onward.

To learn the basic idea behind CEF as an efficient mechanism to perform routing decisions, it is important to understand that the crucial part of routing a packet through a router is finding out how to construct the Layer 2 frame header to allow the packet to be properly encapsulated toward its next hop, and forward the packet out the correct interface. Often, this operation is called a Layer 2 frame rewrite because that is what it resembles: A packet arrives at a router, and the router rewrites the Layer 2 frame, encapsulating the packet appropriately, and sends the packet toward the next hop. The packet's header does not change significantly—in IPv4, only the TTL and checksum are modified; with IPv6, only the Hop Count is decremented. An efficient routing mechanism should therefore focus on speeding up the construction of Layer 2 rewrite information and egress interface lookup. The process switching is highly inefficient in this aspect: The routing table lookup is relatively slow and might need recursive iterations until the directly attached next hop and egress interface are identified. The next-hop information must then be translated in ARP or other Layer 3–to–Layer 2 mapping tables to the appropriate Layer 2 address and the frame header must be constructed, and only then the packet can be encapsulated and forwarded. With each subsequent packet, this process repeats from the beginning.

One important observation is that while the routing table can hold tens of thousands of destination networks (prefixes), a router typically has only a handful of neighbors—the next hops toward all the known destinations. All destinations reachable through a particular next hop are using the same Layer 2 rewrite information. To reach any of the networks behind a particular *adjacent* next hop, the packets will be encapsulated into frames having the same Layer 2 header addresses and sent out the same egress interface. It makes sense, then, to trade memory for speed: Preconstruct the Layer 2 frame headers and egress interface information for each neighbor, and keep them ready in an *adjacency table* stored in the router's memory. This adjacency table can be constructed immediately as the routing table is populated, using IP addresses of next hops in the routing table and utilizing ARP or other Layer 3–to–Layer 2 mapping tables to translate next-hop

IP addresses into their corresponding Layer 2 addresses. A packet that is to be routed through a particular next hop will then simply use the preconstructed Layer 2 frame header for that next hop, without needing to visit the ARP or similar tables over and over again. The process of routing a packet will then transform itself to the process of deciding which entry from the adjacency table should be used to encapsulate and forward the packet. After the proper entry is selected, encapsulating the packet and forwarding it out the egress interface can be done in an extremely rapid way, as all necessary data is readily available.

Another important observation is that the routing table itself is not truly optimized for rapid lookups. It contains lots of information crucial to its construction but not that important for routing lookups, such as origin and administrative distances of routes, their metrics, age, and so on. Entries in the routing table can require recursive lookups: After matching a destination network entry, the next-hop information might contain only the IP address of the next hop but not the egress interface, so the next hop's IP address has to be looked up in the routing table in the next iteration—and the depth of this recursion is theoretically unlimited. Even after matching the ultimate entry in the routing table that finally identifies the egress interface, it does not really say anything about the Layer 2 rewrite that is necessary to forward the packet. The last found next-hop IP address during this lookup process has to be further matched in the ARP or similar mapping tables for the egress interface to find out how to construct the Layer 2 frame header. All these shortcomings can be improved, though: The destination prefixes alone from the routing table can be stored in a separate data structure called the *Forwarding Information Base*, or FIB, optimized for rapid lookups (usually, tree-based data structures meet this requirement). Instead of carrying the plain next hop's IP address from the routing table over into the FIB, each entry in the FIB that represents a destination prefix can instead contain a pointer toward the particular entry in the adjacency table that stores the appropriate rewrite information: Layer 2 frame header and egress interface indication. Any necessary recursion in the routing table is resolved while creating the FIB entries and setting up the pointers toward appropriate adjacency table entries. No other information needs to be carried over from the routing table into the FIB. In effect, the FIB stores only destination prefixes alone. The forwarding information itself is stored as Layer 2 rewrite information in the adjacency table, and entries in the FIB point toward appropriate entries in the adjacency table. All FIB entries that describe networks reachable through a particular next hop point to the same adjacency table entry that contains prepared Layer 2 header and egress information toward that next hop.

After the FIB and adjacency table are created, the routing table is not used anymore to route packets for which all forwarding information is found in the FIB/adjacency table. With FIB-based routers, the routing table can be used for packets that require more complex processing not available through straightforward Layer 2 rewrite; however, for plain packet routing, only the FIB and the adjacency table are used. The routing table therefore becomes more of a source of routing data to build the FIB and adjacency table contents but is not necessarily used to route packets anymore. Therefore, such a routing table is

called the *Routing Information Base (RIB)*—it is the master copy of routing information from which the FIB and other structures are populated, but it is not necessarily used to route packets itself. Note that many routing protocols including Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) construct their own internal routing tables that are also called RIBs. These per-protocol RIBs are usually separate from the router's routing table and shall not be confused with the RIB discussed in this chapter.

Advantages of this approach should be immediately obvious. The FIB contains only the essential information to match a packet's destination address to a known prefix. A single lookup in the FIB immediately produces a pointer toward complete Layer 2 rewrite information for the packet to be forwarded. If the next hop for a destination changes, only the pointer in the respective FIB entry needs to be updated to point toward the new adjacency table entry; the FIB entry itself that represents the destination prefix is unchanged. Both FIB and adjacency tables can be readily constructed from the routing table and the available Layer 3–to–Layer 2 mapping tables, without requiring any packet flows as was the case in fast switching. To those readers familiar with database systems, the FIB can be seen as an index over the adjacency table, with IP prefixes being the lookup keys and the indexed data being the Layer 2 rewrite entries in the adjacency table.

These ideas are at the core of Cisco Express Forwarding, or CEF. Conceptually, CEF consists of two parts—the *Forwarding Information Base* and the *adjacency table*. The FIB contains all known destination prefixes from the routing table, plus additional specific entries, organized as a so-called *mtrie* or a *multiway prefix tree*. The adjacency table contains a Layer 2 frame header prepared for each known next hop or directly attached destination.

The CEF as just described can be implemented in a relatively straightforward way in software, and this is exactly what all software-based Cisco routers do: They implement CEF purely in software, as part of the operating system they run. Both FIB and adjacency tables are maintained in router's memory, and lookups in these structures are done by the CPU as part of interrupt handler executed when a packet is received. Figure 6-2, reused from the Cisco document "How to Choose the Best Router Switching Path for Your Network," Document ID 13706 available on the Cisco website, illustrates the concept.

Multilayer switches and high-end Cisco router platforms go even further, and instead of software-based FIB, they use specialized circuits (specifically, Ternary Content Addressable Memory [TCAM]) to store the FIB contents and perform even faster lookups. Using a TCAM, an address lookup is performed in an extremely short time that does not depend on the number of FIB entries, as the TCAM performs the matching on its entire contents in parallel. On these platforms, the CEF structures are distributed to individual linecards if present, and stored in TCAMs and forwarding ASICs.

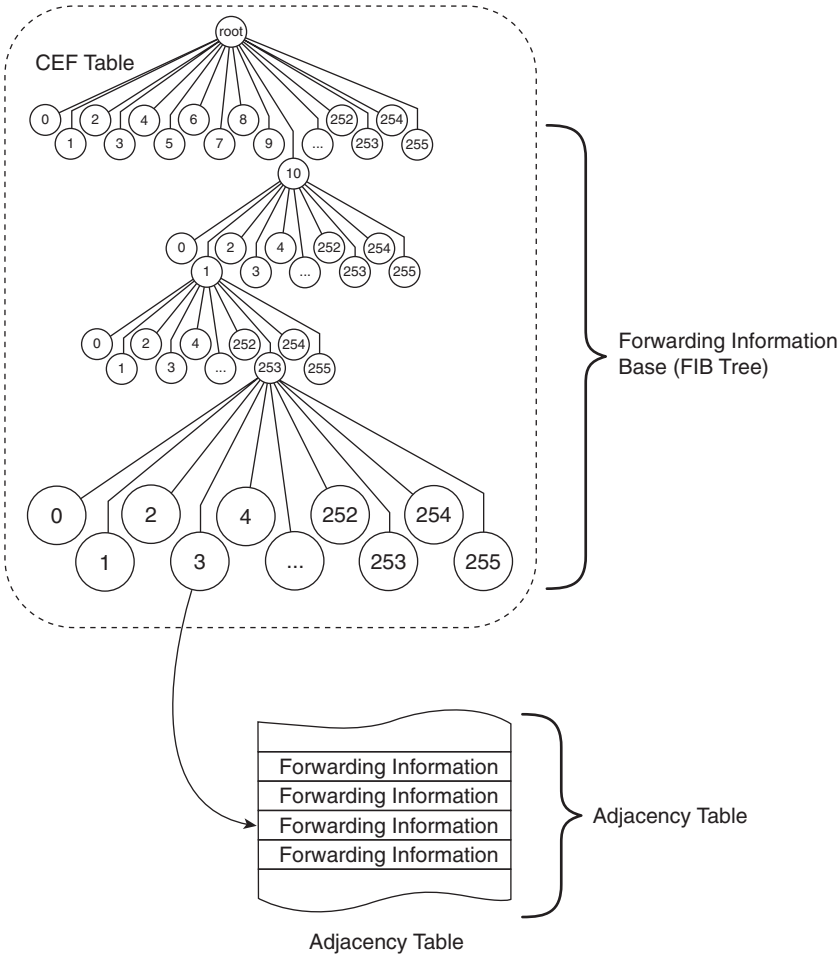

Figure 6-2 *Cisco Express Forwarding Basic Architecture*

To illustrate the CEF in action, consider the network in Figure 6-3 and related Example 6-1.

In this network, Router R1 is connected to two other routers and one multilayer switch. The Data Link Layer technologies interconnecting the devices are diverse: Between R1 and R2, HDLC is used; R1 and R3 are connected over a PPP link; R1 and MLS4 are using Ethernet interconnection in two VLANs—native VLAN and VLAN 2. OSPF is the routing protocol in use. R2 advertises networks 10.2.0.0/24 through 10.2.3.0/24 and FD00:2::/64 through FD00:2:3::/64. In a similar fashion, R3 advertises networks 10.3.4.0/24 through 10.3.7.0/24 and FD00:3:4::/64 through FD00:3:7::/64. MLS4 advertises networks 10.4.8.0/24 and 10.4.9.0/24, and FD00:4:8::/64 and FD00:4:9::/64, over both VLANs. Multiple interface encapsulations and multiple networks reachable over a single next hop are used in this example to show how potentially numerous destination prefixes map to a single adjacent next hop and how the Layer 2 rewrite information is built depending on the Data Link Layer technology. CEF is activated for both IPv4 and IPv6 using the **ip cef** and **ipv6 cef** global configuration commands on R1.

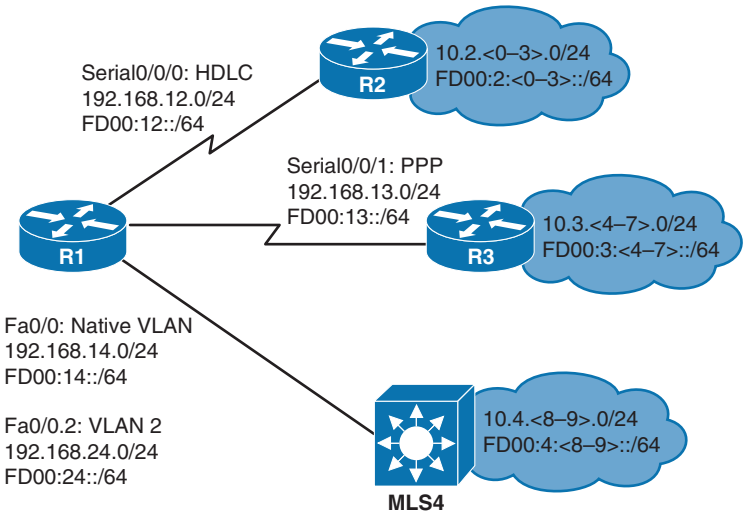

Figure 6-3 *Example Network Showcasing CEF Operation*

Example 6-1 *CEF FIB and Adjacency Table*

```
! On R1, show ip route ospf shows a portion of the RIB

R1# show ip route ospf
     10.0.0.0/8 is variably subnetted, 12 subnets, 2 masks
O       10.2.0.0/24 [110/782] via 192.168.12.2, 00:07:06, Serial0/0/0
O       10.2.1.0/24 [110/782] via 192.168.12.2, 00:07:06, Serial0/0/0
O       10.2.2.0/24 [110/782] via 192.168.12.2, 00:07:06, Serial0/0/0
O       10.2.3.0/24 [110/782] via 192.168.12.2, 00:07:06, Serial0/0/0
O       10.3.4.1/32 [110/782] via 192.168.13.3, 00:07:06, Serial0/0/1
O       10.3.5.0/24 [110/782] via 192.168.13.3, 00:07:06, Serial0/0/1
O       10.3.6.0/24 [110/782] via 192.168.13.3, 00:07:06, Serial0/0/1
O       10.3.7.0/24 [110/782] via 192.168.13.3, 00:07:06, Serial0/0/1
O       10.4.8.0/24 [110/2] via 192.168.24.4, 00:07:06, FastEthernet0/0.2
                    [110/2] via 192.168.14.4, 00:07:06, FastEthernet0/0
O       10.4.9.0/24 [110/2] via 192.168.24.4, 00:07:06, FastEthernet0/0.2
                    [110/2] via 192.168.14.4, 00:07:06, FastEthernet0/0

! Another crucial part of information is the ARP table that resolves
! next hop IP addresses of hosts connected via Ethernet to MAC addresses
! Serial interface technologies in this example are point-to-point and
! hence require no Layer 3-to-Layer 2 mapping tables. This information will
! be used in construction of adjacency table entries

R1# show ip arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.14.4           41   0017.9446.b340  ARPA   FastEthernet0/0
```

```
Internet  192.168.14.1            -   0019.e87f.38e4  ARPA   FastEthernet0/0
Internet  192.168.24.1            -   0019.e87f.38e4  ARPA   FastEthernet0/0.2
Internet  192.168.24.4           41   0017.9446.b341  ARPA   FastEthernet0/0.2

! show ip cef shows the FIB contents. In the following output, only routes
! learned via OSPF are shown for brevity reasons. Note how a set of prefixes
! resolves through a particular adjacency (next hop IP and egress interface).

R1# show ip cef 10.0.0.0 255.0.0.0 longer-prefixes
Prefix              Next Hop             Interface
10.2.0.0/24         192.168.12.2         Serial0/0/0
10.2.1.0/24         192.168.12.2         Serial0/0/0
10.2.2.0/24         192.168.12.2         Serial0/0/0
10.2.3.0/24         192.168.12.2         Serial0/0/0
10.3.4.1/32         192.168.13.3         Serial0/0/1
10.3.5.0/24         192.168.13.3         Serial0/0/1
10.3.6.0/24         192.168.13.3         Serial0/0/1
10.3.7.0/24         192.168.13.3         Serial0/0/1
10.4.8.0/24         192.168.24.4         FastEthernet0/0.2
                    192.168.14.4         FastEthernet0/0
10.4.9.0/24         192.168.24.4         FastEthernet0/0.2
                    192.168.14.4         FastEthernet0/0

! Similarly, for IPv6, the relevant outputs are:

R1# show ipv6 route ospf
! Output shortened and reformatted for brevity
O   FD00:2::/64 [110/782]   via FE80::2, Serial0/0/0
O   FD00:2:1::/64 [110/782] via FE80::2, Serial0/0/0
O   FD00:2:2::/64 [110/782] via FE80::2, Serial0/0/0
O   FD00:2:3::/64 [110/782] via FE80::2, Serial0/0/0
O   FD00:3:4::/64 [110/782] via FE80::3, Serial0/0/1
O   FD00:3:5::/64 [110/782] via FE80::3, Serial0/0/1
O   FD00:3:6::/64 [110/782] via FE80::3, Serial0/0/1
O   FD00:3:7::/64 [110/782] via FE80::3, Serial0/0/1
O   FD00:4:8::/64 [110/2]   via FE80:24::4, FastEthernet0/0.2
                            via FE80:14::4, FastEthernet0/0
O   FD00:4:9::/64 [110/2]   via FE80:24::4, FastEthernet0/0.2
                            via FE80:14::4, FastEthernet0/0

R1# show ipv6 neighbors
IPv6 Address                              Age Link-layer Addr State Interface
FD00:14::4                                  1 0017.9446.b340  STALE Fa0/0
FD00:24::4                                  1 0017.9446.b341  STALE Fa0/0.2
FE80::3                                     - -               REACH Se0/0/1
FE80:14::4                                  2 0017.9446.b340  STALE Fa0/0
```

```
FE80:24::4                                1 0017.9446.b341  STALE Fa0/0.2

R1# show ipv6 cef
! Output shortened and reformatted for brevity
FD00:2::/64   nexthop FE80::2 Serial0/0/0
FD00:2:1::/64 nexthop FE80::2 Serial0/0/0
FD00:2:2::/64 nexthop FE80::2 Serial0/0/0
FD00:2:3::/64 nexthop FE80::2 Serial0/0/0
FD00:3:4::/64 nexthop FE80::3 Serial0/0/1
FD00:3:5::/64 nexthop FE80::3 Serial0/0/1
FD00:3:6::/64 nexthop FE80::3 Serial0/0/1
FD00:3:7::/64 nexthop FE80::3 Serial0/0/1
FD00:4:8::/64 nexthop FE80:24::4 FastEthernet0/0.2
              nexthop FE80:14::4 FastEthernet0/0
FD00:4:9::/64 nexthop FE80:24::4 FastEthernet0/0.2
              nexthop FE80:14::4 FastEthernet0/0

! The show adjacency shows an abbreviated list of adjacency table entries
! Note that separate entries are created for IPv4 and IPv6 adjacencies,
! as the Protocol or EtherType field value in pre-constructed frame headers
! is different for IPv4 and IPv6

R1# show adjacency
Protocol Interface                 Address
IPV6     Serial0/0/0               point2point(12)
IP       Serial0/0/0               point2point(13)
IPV6     Serial0/0/1               point2point(10)
IP       Serial0/0/1               point2point(15)
IPV6     FastEthernet0/0.2         FE80:24::4(12)
IP       FastEthernet0/0           192.168.14.4(23)
IPV6     FastEthernet0/0           FE80:14::4(12)
IP       FastEthernet0/0.2         192.168.24.4(23)
IPV6     Serial0/0/1               point2point(4)
IPV6     FastEthernet0/0.2         FD00:24::4(5)
IPV6     FastEthernet0/0           FD00:14::4(7)

! Now focus on the adjacency table details. There are adjacencies via multiple
! interfaces. Serial0/0/0 is running HDLC. Note in the show adjacency detail
! command output the prepared HDLC header for all IPv6 prefixes (0F0086DD)
! and IP prefixes (0F000800) resolving through this adjacency.

R1# show adjacency s0/0/0 detail
Protocol Interface                 Address
IPV6     Serial0/0/0               point2point(12)
                                   0 packets, 0 bytes
                                   0F0086DD
```