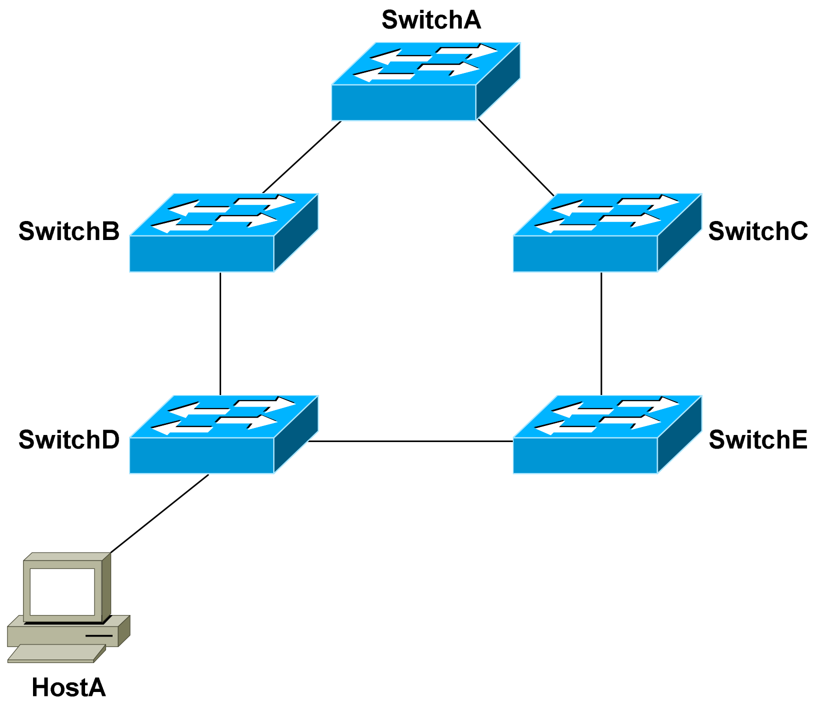
- Spanning Tree Protocol -

Switching Loops

A Layer-2 switch belongs to only *one* **broadcast domain,** and will forward both broadcasts and multicasts out *every* port but the originating port.

When a **switching loop** is introduced into the network, a destructive **broadcast storm** will develop within *seconds*. A storm occurs when broadcasts are endlessly forwarded through the loop. Eventually, the storm will choke off all other network traffic.

Consider the following example:



If HostA sends out a broadcast, SwitchD will forward the broadcast out all ports in the same VLAN, including the trunk ports connecting to SwitchB and SwitchE. In turn, those two switches will forward that broadcast out all ports, including the trunks to the neighboring SwitchA and SwitchC.

The broadcast will loop around the switches *infinitely*. In fact, there will be *two* separate broadcast storms cycling in opposite directions through the switching loop. Only powering off the switches or physically removing the loop will stop the storm.

Spanning Tree Protocol (STP)

Spanning Tree Protocol (STP) was developed to prevent the broadcast storms caused by switching loops. STP was originally defined in **IEEE 802.1D**.

Switches running STP will build a map or **topology** of the entire switching network. STP will identify if there are any loops, and then disable or *block* as many ports as necessary to eliminate all loops in the topology.

A blocked port can be reactivated if another port goes down. This allows STP to maintain redundancy and fault-tolerance.

However, because ports are blocked to eliminate loops, STP does not support load balancing unless an EtherChannel is used. EtherChannel is covered in great detail in another guide.

STP switches exchange **Bridge Protocol Data Units (BPDU's)** to build the topology database. BPDU's are forwarded out all ports every **two seconds,** to a dedicated MAC multicast address of 0180.c200.0000.

Building the STP topology is a multistep **convergence** process:
- A **Root Bridge** is elected
- **Root ports** are identified
- **Designated ports** are identified
- Ports are placed in a **blocking** state as required, to eliminate loops

The **Root Bridge** serves as the central reference point for the STP topology. STP was originally developed when Layer-2 *bridges* were still prevalent, and thus the term Root *Bridge* is still used for nostalgic reasons. It is also acceptable to use the term **Root Switch**, though this is less common.

Once the full topology is determined, and loops are eliminated, the switches are considered **converged.**

STP is **enabled** by default on all Cisco switches, for all VLANs.

Electing an STP Root Bridge

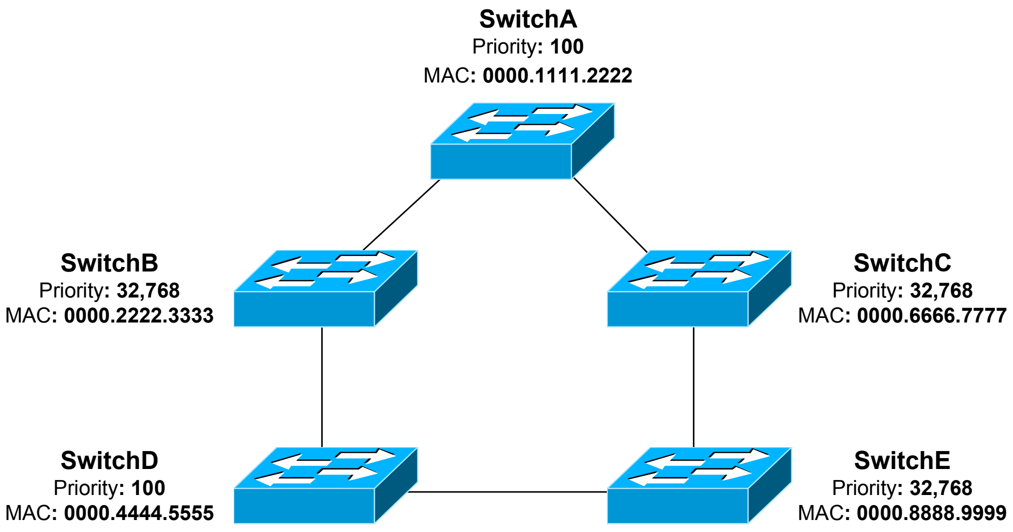
The first step in the STP convergence process is electing a **Root Bridge**, which is the central reference point for the STP topology. As a best practice, the Root Bridge should be the most centralized switch in the STP topology.

A Root Bridge is elected based on its **Bridge ID,** comprised of two components in the original 802.1D standard:

- 16-bit **Bridge priority**
- 48-bit **MAC address**

The default priority is **32,768**, and the **lowest priority wins**. If there is a tie in priority, the **lowest MAC address** is used as the tie-breaker.

Consider the following example:



Switches exchange BPDU's to perform the election process, and the lowest Bridge ID determines the Root Bridge:

- SwitchB, SwitchC, and SwitchE have the default priority of 32,768.
- SwitchA and SwitchD are tied with a lower priority of 100.
- SwitchA has the lowest MAC address, and will be elected the Root Bridge.

By default, a switch will always believe it is the Root Bridge, until it receives a BPDU from a switch with a lower Bridge ID. This is referred to as a **superior BPDU.** The election process is continuous – if a new switch with the lowest Bridge ID is added to the topology, it will be elected as the Root Bridge.

Identifying Root Ports

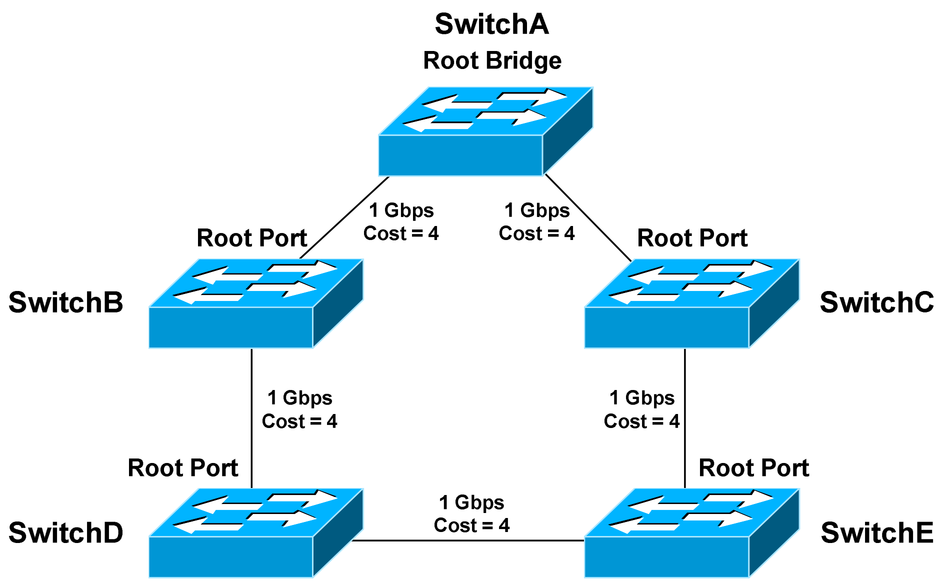
The second step in the STP convergence process is to identify **root ports.** The root port of each switch has the lowest **root path cost** to get to the Root Bridge.

Each switch can only have *one* root port. The Root Bridge *cannot* have a root port, as the purpose of a root port is to *point* to the Root Bridge.

Path cost is a *cumulative* cost to the Root Bridge, based on the bandwidth of the links. The *higher* the bandwidth, the *lower* the path cost:

Bandwidth	***Cost***
4 Mbps	250
10 Mbps	100
16 Mbps	62
45 Mbps	39
100 Mbps	19
155 Mbps	14
1 Gbps	4
10 Gbps	2

A lower cost is preferred. Consider the following example:

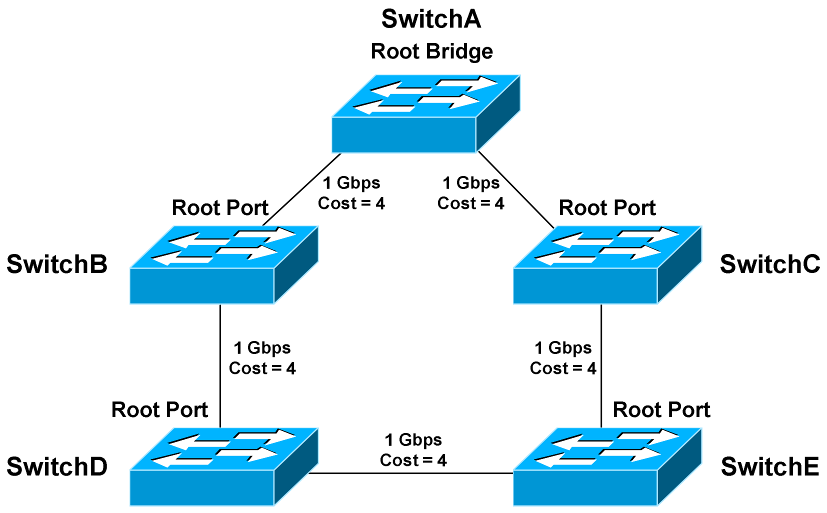


Each 1Gbps link has a path cost of *4*. SwitchA has a cumulative path cost of *0*, because it is the Root Bridge. Thus, when SwitchA sends out BPDU's, it advertises a root path cost of 0.

Identifying Root Ports (continued)

SwitchB has two paths to the Root Bridge:

- A direct connection to SwitchA, with a path cost of *4*.
- Another path through SwitchD, with a path cost of *16*.



The **lowest cumulative path cost** is considered **superior**, thus the port directly connecting to SwitchA will become the root port. A BPDU advertising a *higher* path cost is often referred to as an **inferior BPDU.**

SwitchD also has two paths to the Root Bridge:

- A path through SwitchB, with a path cost of *8*.
- A path through SwitchE, with a path cost of *12*.
- The port to SwitchB is preferred, and will become the root port.

Recall that the Root Bridge will advertise BPDU's with a path cost of *0*. As the downstream switches *receive* these BPDU's, they will add the path cost of the *receiving port*, and then advertise the cumulative cost to neighbors.

For example, SwitchC will receive a BPDU with a path cost of *0* from SwitchA, which is the Root Bridge. SwitchC will add the path cost of its receiving port, and thus SwitchC's cumulative path cost will be *4*.

SwitchC will advertise a path cost of *4* to SwitchE, which will add the path cost of its receiving port. SwitchE's cumulative path cost will thus be *8*.

Path cost can be artificially adjusted on a per-port basis:

```
SwitchD(config)# int gi2/22
SwitchD(config-if)# spanning-tree vlan 101 cost 42
```

Identifying Designated Ports

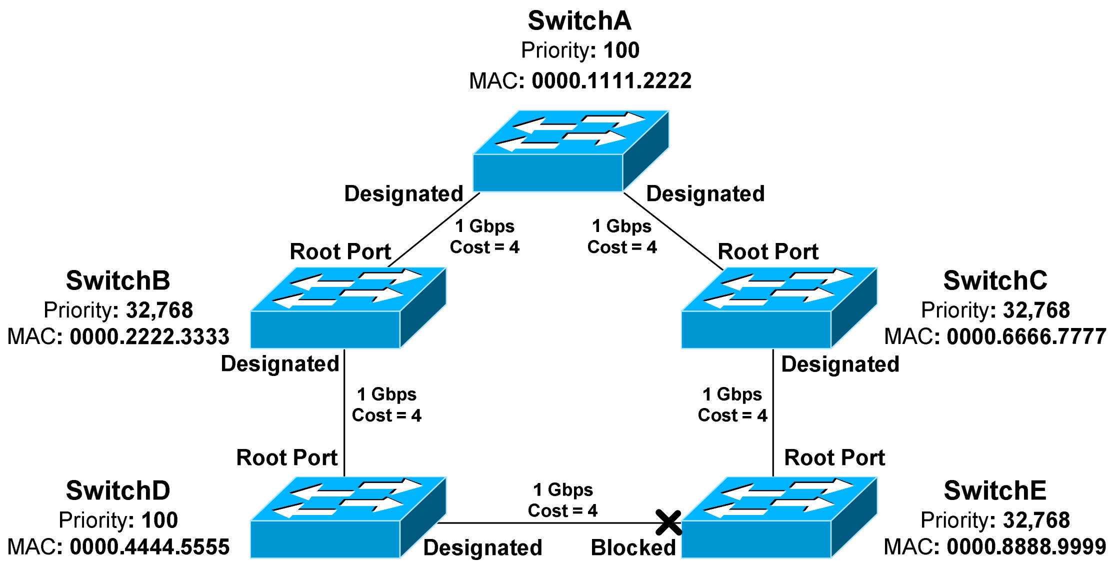
The third step in the STP convergence process is to identify **designated ports.** A single designated port is identified for each network *segment.* This port is responsible for forwarding BPDUs and frames to that segment.

If two ports are eligible to become the designated port, then there is a **loop**. One of the ports will be placed in a blocking state to eliminate the loop.

Similar to a root port, the designated port is determined by the lowest cumulative path cost leading the Root Bridge. A designated port will *never* be placed in a blocking state, unless there is a change to the switching topology and a more preferred designated port is elected.

Note: A port can never be both a designated port *and* a root port.

Consider the following example:



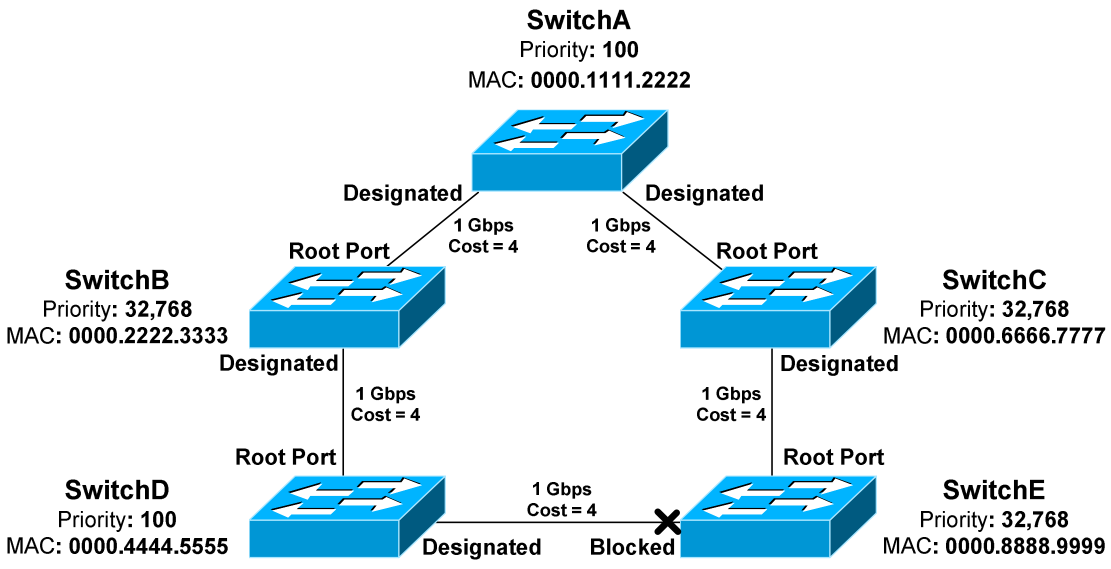
Ports on the Root Bridge are *never* placed in a blocking state. Thus, the two ports off of SwitchA will automatically become designated ports.

Remember, every network segment **must have one designated port**, regardless if a root port already exists on that segment.

Thus, the network segments between SwitchB and SwitchD, and between SwitchC and SwitchE, both require a designated port. The ports on SwitchD and Switch E have already been identified as root ports, thus the ports on Switch B and C will become the designated ports.

Identifying Designated Ports (continued)

Note that the network segment between SwitchD and SwitchE does not contain a root port:



Because two ports on this segment are eligible to become the designated port, STP recognizes that a loop exists. One of the ports must be elected as the designated port, and the other must be placed in a blocking state.

Normally, whichever switch has the lowest cumulative path cost will have its port become designated. The switch with the highest path cost will have its port blocked.

In the above example, there is a **tie** in cumulative path cost. Both SwitchD and SwitchE have a path cost of *12* to reach the Root Bridge on that segment.

The **lowest Bridge ID** is used as the tiebreaker. SwitchD has a priority of *100,* and SwitchE has the default priority of *32,768*.

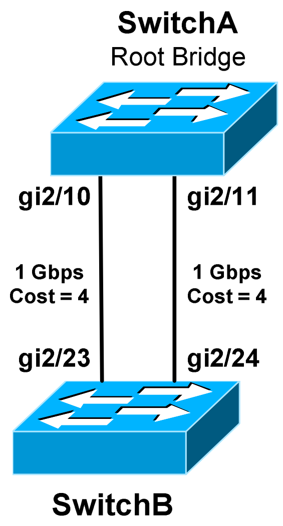
Thus, the port on SwitchD will become the designated port. The port on SwitchE will be placed in a blocking state.

As with electing the Root Bridge, if there is a tie in priority, the **lowest MAC address** is used as the tie breaker.

Remember: Any port not elected as a root or designated port will be placed in a blocking state.

Port ID

When electing root and designated ports, it is possible to have a tie in *both* path cost *and* Bridge ID. Consider the following example:



The bandwidth of both links is equal, thus both ports on SwitchB have an equal path cost to the Root Bridge. Which port will become the root port then? Normally, the lowest Bridge ID is used as the tiebreaker, but that is not possible in this circumstance.

Port ID is used as the final tiebreaker, and consists of two components:

- 4-bit **port priority**
- 12-bit **port number**, derived from the physical port number

By default, the port priority of an interface is **128**, and a *lower* priority is preferred. If there is a tie in priority, the lowest port number is preferred.

The *sender* port ID determines the tie break, and not the *local* port ID. In the above example, SwitchB must decide whether *gi2/23* or *gi2/24* becomes the root port. SwitchB will observe BPDU's from SwitchA, which will contain the port ID's for *gi2/10* and *gi2/11.*

If priorities are equal, the sender Port ID from *gi2/10* is preferred, due to the lower port number. Thus, *gi2/23* on SwitchB will become the root port.

The port number is a fixed value, but port priority can be changed on a per-interface basis:

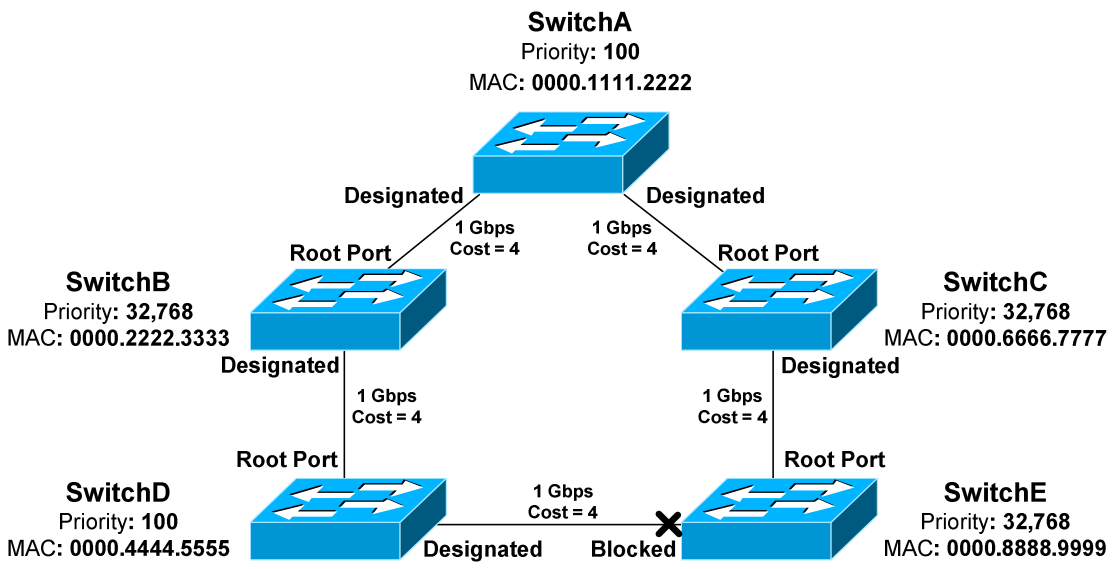
```
Switch(config)#  int gi2/11
Switch(config-if)#  spanning-tree vlan 101 port-priority 32
```

Port ID (continued)

Note: Some reference material may state that the Port ID is comprised of an *8-bit* priority and *8-bit* port number. This was accurate in the *original* 802.1D specification.

However, **IEEE 802.1t** revised the original specification to provide the larger *12-bit* port number field, to accommodate modular switches with high port density.

Even more confusing – some whitepapers on Cisco's website will define the Port ID as a combination of port priority and *MAC address*, instead of port number. This is not accurate in modern STP implementations.



Remember: Port ID is the *last* tiebreaker STP will consider. STP determines root and designated ports using the following criteria, *in order*:

- Lowest path cost to the Root Bridge
- Lowest bridge ID
- Lowest sender port ID

Lowest Bridge ID is *always* used to determine the Root Bridge.

Versions of STP

There are three flavors of the original 802.1D version of STP:

- **Common Spanning Tree (CST)**
- **Per-VLAN Spanning Tree (PVST)**
- **Per-VLAN Spanning Tree Plus (PVST+)**

CST utilizes a *single* STP instance for *all* VLANs, and is sometimes referred to as *mono* spanning tree. All CST BPDU's are sent over the **native VLAN** on a trunk port, and thus are untagged.

PVST employs a *separate* STP instance for *each* VLAN, improving flexibility and performance. PVST requires trunk ports to use **ISL encapsulation.** PVST and CST are not compatible.

The enhanced **PVST+** is compatible with both CST and PVST, and supports both ISL and 802.1Q encapsulation. PVST+ is the *default* mode on many Cisco platforms.

STP has continued to evolve over time. Modern extensions of STP will be covered later in this guide:

- **Rapid Spanning Tree Protocol (RSTP)**
- **Multiple Spanning Tree (MST)**

Extended System IDs

In the original 802.1D standard, the 64-bit Bridge ID consisted of two components:

- 16-bit **Bridge priority**
- 48-bit **MAC address**

As STP evolved to operate on a per-VLAN basis, a unique Bridge ID became mandatory for each VLAN. This was originally accomplished by assigning a unique switch MAC address to the Bridge ID of each VLAN.

This approach suffered from scalability issues, requiring that a switch support at least 1024 unique system MAC addresses – at least one per VLAN.

IEEE 802.1t altered the Bridge ID to include an **extended system ID**, which identifies the VLAN number of the STP instance. The Bridge ID remained 64 bits, but now consisted of three components:

- 4-bit **Bridge priority**
- 12-bit **System** or **VLAN ID**
- 48-bit **MAC address**

By stealing 12 bits from the bridge priority, the *range* of priorities is altered:

- The original priority ranged from *0* to *65,535,* with **32,768** as default.
- With extended system IDs, the new priority range is *0* to *61,440*, and the priority must be in multiples of 4,096.
- The default is still 32,768.

Extended system ID's are **enabled by default** and **cannot be disabled** if a switch platform does not support 1024 system MAC addresses.

For platforms that support 1024 MAC addresses, the extended system ID can be manually enabled:

```
Switch(config)#  spanning-tree extend system-id
```

Extended system IDs increase the number of supported VLANs in the STP topology from *1005* to *4094*.

Basic STP Configuration

STP is **enabled** by default on all Cisco switches, for all VLANs and ports. PVST+ is the default STP mode on most modern Cisco platforms, allowing each VLAN to run a separate STP instance.

STP can be disabled. This should be done with caution - any switching loop will result in a broadcast storm. To disable STP for an entire VLAN:

```
Switch(config)#  no spanning-tree vlan 101
```

A range of VLANs can be specified:

```
Switch(config)#  no spanning-tree vlan 1 - 4094
```

STP can also be disabled on a per-port basis, for a specific VLAN:

```
Switch(config)#  interface gi2/23
Switch(config-if)#  no spanning-tree vlan 101
```

The switch with the lowest Bridge ID is elected as the Root Bridge. The priority can be adjusted from its default of 32,768, to increase the likelihood that a switch is elected as the Root Bridge.

Priority can be configured on a per-VLAN basis. Remember that the priority must be in multiples of 4,096 when extended system IDs are enabled:

```
SwitchA(config)#  spanning-tree vlan 101 priority 8192
```

A switch can be indirectly *forced* to become the Root Bridge for a specific VLAN:

```
SwitchA(config)#  spanning-tree vlan 101 root primary
```

The *root primary* parameter automatically lowers the priority to 24,576. If another switch has a priority *lower* than 24,576, the priority will be lowered to 4,096 less than the current Root Bridge.

STP does not technically support a *backup* Root Bridge. However, the *root secondary* command can increase the likelihood that a specified switch will succeed as the new Root Bridge in the event of a failure:

```
SwitchB(config)#  spanning-tree vlan 101 root secondary
```

The *root secondary* parameter in the above command automatically lowers the switch's priority to 28,672.

STP Port States

As STP *converges* the switching topology, a switch port will progress through a series of **states**:

- **Blocking**
- **Listening**
- **Learning**
- **Forwarding**

Initially, a switch port will start in a **blocking** state:

- A blocking port *will not* forward frames or learn MAC addresses.
- A blocking port *will* still *listen* for BPDUs from other switches, to learn about changes to the switching topology.

A port will then transition from a blocking to a **listening** state:

- The switch must believe that the port *will not be shut down* to eliminate a loop. In other words, the port may become a root or designated port.
- A listening port *will not* forward frames or learn MAC addresses.
- A listening port *will* send and listen for BPDUs, to participate in the election of the Root Bridge, root ports, and designated ports.
- If a listening port is *not elected* as a root or a designated Port, it will **transition back to a blocking state.**

If a listening port is elected as a *root* or *designated* port, it will transition to a **learning** state:

- A port must wait a brief period of time, referred to as the **forward delay**, before transitioning from a listening to learning state.
- A learning port *will continue* to send and listen for BPDUs.
- A learning port *will begin* to add MAC addresses to the CAM table.
- However, a learning port *cannot* forward frames quite yet.

Finally, a learning port will transition to a **forwarding** state:

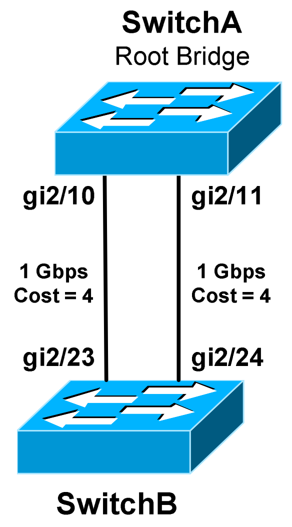
- A port must wait *another* forward delay before transitioning from learning to forwarding.
- A forwarding port is fully functional – it will send and listen for BPDUs, learn MAC addresses, and forward frames.
- Root and designated ports will eventually transition to a forwarding state.

STP Port States (continued)

Technically, there is a fifth port state – **disabled.** A port in a disabled state has been *administratively shutdown*. A disabled port does not forward frames or participate in STP convergence.

Why does a port start in a blocking state? STP *must* initially assume that a loop exists. A broadcast storm can form in seconds, and requires physical intervention to stop.

Thus, STP will always take a proactive approach. Starting in a blocking state allows STP to complete its convergence process *before* any traffic is forwarded. In perfect STP operation, a broadcast storm should never occur.



To view the current state of a port:

SwitchA# *show spanning-tree interface gi2/10*

```
Vlan                 Role Sts Cost      Prio.Nbr Type
-------------------- ---- --- --------- -------- ------------------
VLAN0101             Desg FWD 4         128.34   P2p
VLAN0102             Desg FWD 4         128.34   P2p
```

SwitchB# *show spanning-tree interface gi2/24*

```
Vlan                 Role Sts Cost      Prio.Nbr Type
-------------------- ---- --- --------- -------- ------------------
VLAN0101             Root FWD 4         128.48   P2p
VLAN0102             Altn BLK 4         128.48   P2p
```

STP Timers

Switches running STP exchange BPDUs to build and converge the topology database. There are three **timers** that are crucial to the STP process:

- **Hello timer**
- **Forward delay timer**
- **Max age timer**

The **hello timer** determines how often switches send BPDUs. By default, BPDUs are sent every **2 seconds.**

The **forward delay timer** determines how long a port must spend in both a *learning* and *listening* state:

- Introducing this delay period ensures that STP will have enough time to detect and eliminate loops.
- By default, the forward delay is **15 seconds**.
- Because a port must transition through *two* forward delays, the *total* delay time is 30 seconds.

The **max age timer** indicates how long a switch will retain BPDU information from a neighbor switch, before discarding it:

- Remember that BPDUs are sent every two seconds.
- If a switch fails to receive a BPDU from a neighboring switch for the max age period, it will assume there was a change in the switching topology.
- STP will then purge that neighbor's BPDU information.
- By default, the max age timer is **20 seconds.**

Timer values can be adjusted. However, this is rarely necessary, and can negatively impact STP performance and reliability.

Timers must be changed on the **Root Bridge.** The Root Bridge will propagate the new timer values to all switches **using BPDUs**. Non-root switches will **ignore** their locally configured timer values.

To manually adjust the three STP timers for a specific VLAN:

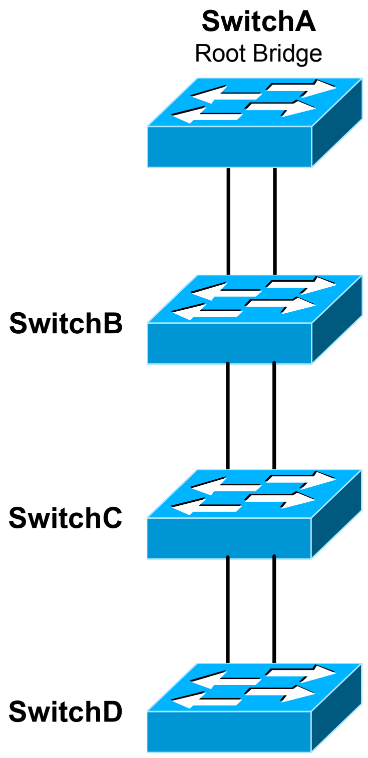
```
Switch(config)#  spanning-tree vlan 101 hello-time 10
Switch(config)#  spanning-tree vlan 101 forward-time 20
Switch(config)#  spanning-tree vlan 101 max-age 40
```

The timer values are measured in seconds, and the above represents the *maximum* possible value for each timer.

STP Diameter

The default values of each STP timer are based on the **diameter** of the switching topology.

The diameter is the *length* of the topology, measured in the number of switches including the Root Bridge. The following example has a diameter of *4* switches:



By default, STP assumes a switching diameter of **7.** This is also the *maximum* diameter.

Note: The switching topology can contain more than seven switches. However, each *branch* of the switching *tree* can only extend seven switches deep, with the Root Bridge always at the top of the branch.

The *diameter* should be configured on the Root Bridge:

```
SwitchA(config)#  spanning-tree vlan 101 root primary diameter 5
```

The *diameter* command adjusts the hello, forward delay, and max age timers. This is the **recommended way** to adjust timers, as the timers are tuned specifically to the diameter of the switching network.

STP Topology Changes

Switches exchange two types of BPDUs when building and converging the topology database:

- **Configuration BPDUs**
- **Topology Change Notification (TCN) BPDUs**

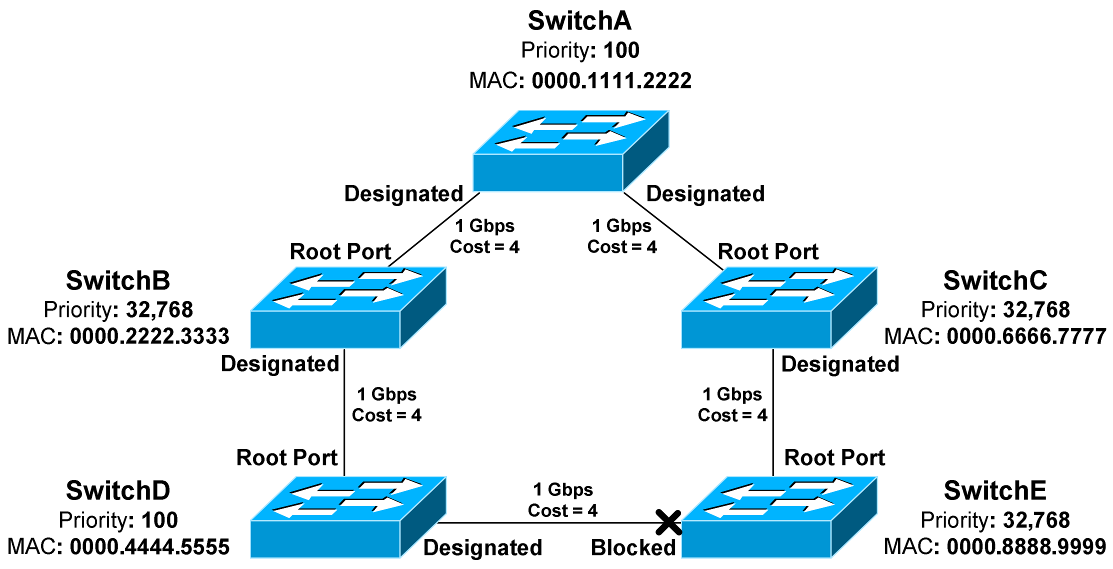
Configuration BPDUs are used to elect Root Bridges, root ports, and designated ports.

A TCN will be sent under two circumstances:

- When a port transitions into a *forwarding* state.
- When a *forwarding* or *learning* port transitions into a *blocking* or *down* state.

When a topology change occurs, a switch will send a TCN BPDU out its **root port**, destined for the Root Bridge. The TCN contains no information about the change – it only indicates that a change *occurred.*

Consider the following example:

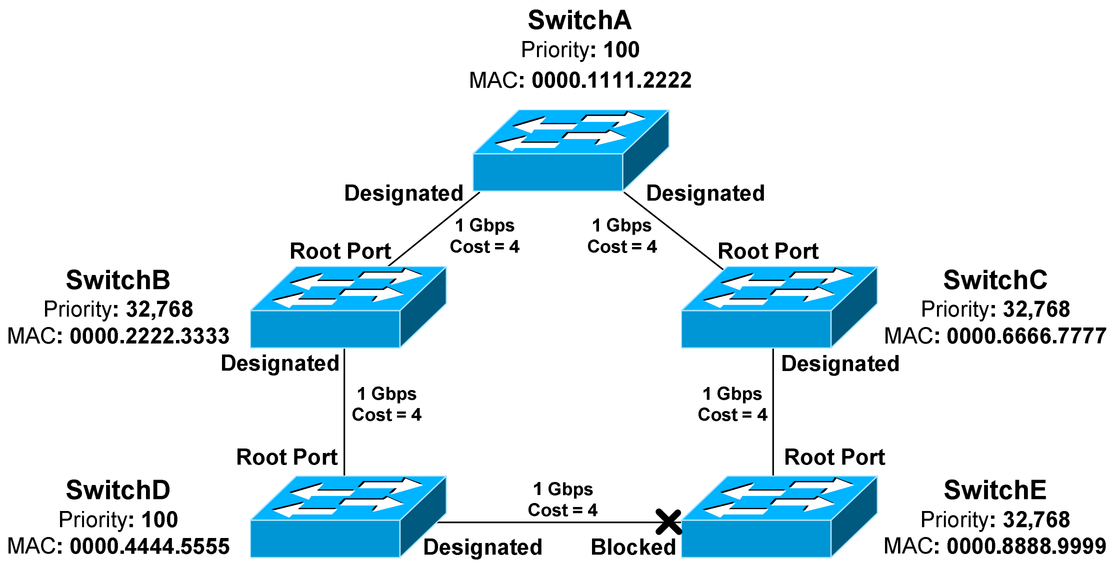


If the port on SwitchD connecting to SwitchE went down:

- SwitchD would send a TCN out its root port to SwitchB.
- SwitchB will *acknowledge* this TCN, by responding with a TCN with the **Topology Change Acknowledgement (TCA) flag** set.
- SwitchB then forward the TCN out *its* root port to SwitchA, the Root Bridge.

STP Topology Changes (continued)

Once the Root Bridge receives the TCN, it will send out a *configuration* BPDU to *all* switches, with the **Topology Change (TC) flag** set. This ensures that all switches in the STP topology are informed of the change.



When a switch receives this root BPDU, it will temporarily reduce its CAM aging timer from 300 seconds to a value equal to the forward delay timer - **15 seconds** by default. This allows any erroneous MAC addresses to be quickly flushed from the CAM table.

The CAM aging timer will remain at a reduced value for the duration of one forward delay *plus* one max age period – a total of **35 seconds** by default.

Two types of failures can occur in the STP topology, depending on the *perspective* of a switch:

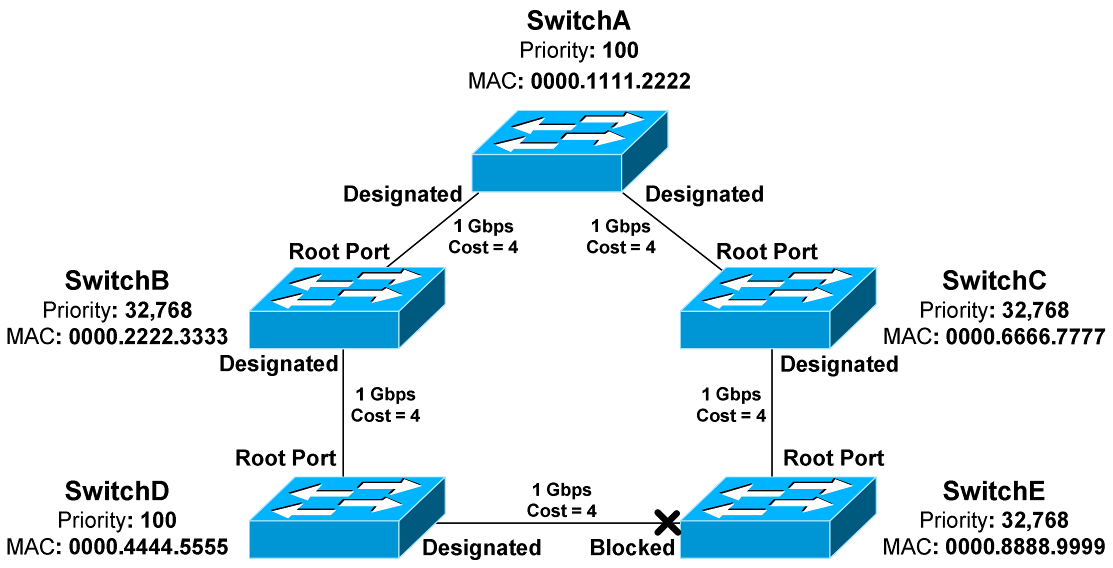
- **Direct** failures
- **Indirect** failures

For example, if the root port on SwitchB fails, SwitchB would consider this a **direct failure.** SwitchB will detect immediately that the physical port is down, and STP will react accordingly.

That same port failing would represent an **indirect failure** for SwitchD. SwitchD would lose its path to the Root Bridge. However, because the port is not local on SwitchD, it must learn of the topology change from its neighbors.

STP Topology Changes (continued)

By detecting and reacting to link failures, STP can take advantage of the redundancy provided by loops. However, the failover is *not* instantaneous.



If the root port on SwitchE were to fail:

- SwitchE would immediately purge any BPDU information received from SwitchC.
- SwitchC would send a TCN to the Root Bridge.
- The Root Bridge would send a configuration BPDU to all switches, with the TC flag set.
- All switches would reduce their CAM aging timer to 15 seconds.
- SwitchE would eventually receive a BPDU from SwitchD. **Remember,** blocked ports *still* receive BPDUs to learn about topology changes.
- The blocked port to SwitchD now represents the best and only path for SwitchE to reach the Root Bridge.
- The blocked port will transition first to a listening state, and then to a learning state. The port will wait the forward delay time in both states, for a total of 30 seconds.
- The port will finally transition to a forwarding state.

Thus, hosts on SwitchE will be impacted by this failure for a *minimum* of **30 seconds.** STP will maintain redundancy if there is a loop, but a link failure will still negatively impact the network for a short period.

Improving STP Convergence

In many environments, a 30 second outage for every topology change is unacceptable. Cisco developed three proprietary features that improve STP convergence time:

- **PortFast**
- **UplinkFast**
- **BackboneFast**

Each feature will be covered in detail in the following sections.

PortFast

By default, all ports on a switch participate in the STP topology. This includes any port that connects to a *host*, such as a workstation. In most circumstances, a host represents no risk of a loop.

The host port will transition through the normal STP states, including waiting two forward delay times. Thus, a host will be without network connectivity for a *minimum* of 30 seconds when first powered on.

This is not ideal for a couple reasons:

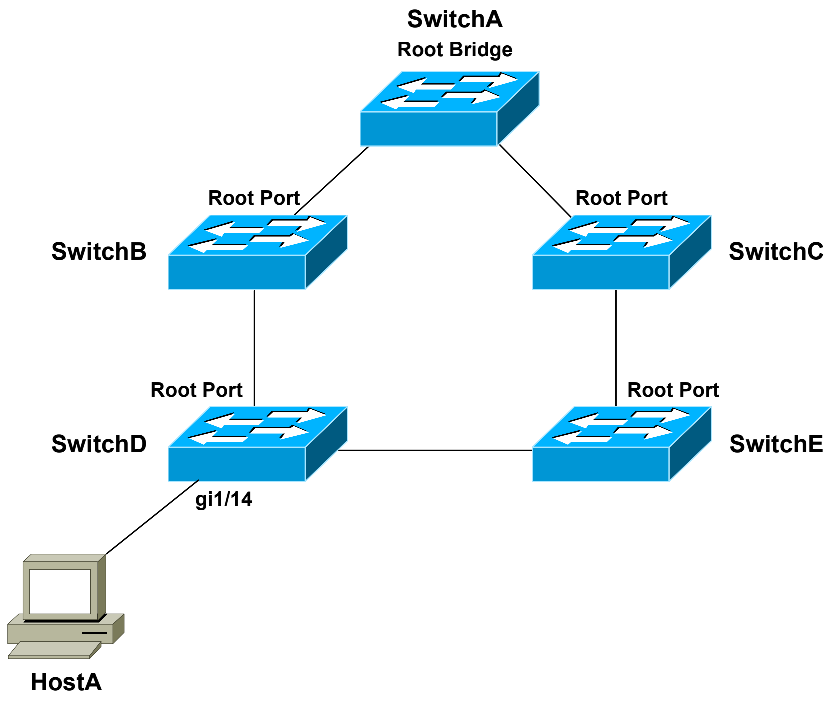
- Users will be annoyed by the brief outage.
- A host will often request an IP address through DHCP during bootup. If the switch port is not *forwarding* quickly enough, the DHCP request may fail.
- Devices that boot from network may fail as well.

PortFast allows a switch port to bypass the usual progression of STP states. The port will instead transition from a *blocking* to a *forwarding* state **immediately**, eliminating the typical 30 second delay.

PortFast should *only* be enabled on ports connected to a host. If enabled on a port connecting to a switch or hub, any loop may result in a broadcast storm.

Note: PortFast does *not* disable STP on a port - it merely accelerates STP convergence. If a PortFast-enabled port receives a BPDU, it will transition through the normal process of STP states.

PortFast (continued)



PortFast provides an additional benefit. Remember that a switch will generate a TCN if a port transitions to a *forwarding* or *blocked* state. This is true even if the port connects to a host device, such as a workstation.

Thus, powering on or off a workstation will cause TCNs to reach the Root Bridge, which will send out configuration BPDUs in response. Because the switching topology did not technically *change*, no outage will occur.

However, all switches will reduce the CAM aging timer to 15 seconds, thus purging MAC addresses from the table very quickly. This will increase frame flooding and reduce the efficiency and performance.

PortFast eliminates this unnecessary BPDU traffic and frame flooding. A TCN will not be generated for state changes on a PortFast-enabled port.

Portfast is **disabled** by default. To enable PortFast on a switch port:

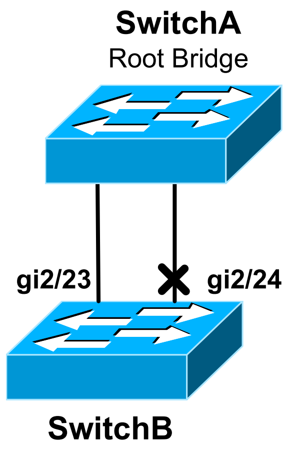
```
SwitchD(config)#  int gi1/14
SwitchD(config-if)#  spanning-tree portfast
```

PortFast can also be globally enabled for all interfaces:

```
SwitchD(config)#  spanning-tree portfast default
```

UplinkFast

Often, a switch will have multiple uplinks to another *upstream* switch:



If the links are not bundled using an EtherChannel, at least one of the ports will transition to a *blocking* state to eliminate the loop. In the above example, port *gi2/24* was placed into a blocking state on SwitchB.

Normally, if the root port fails on the local switch, STP will need to perform a recalculation to transition the *other* port out of a blocking state. At a minimum, this process will take **30 seconds.**

UplinkFast allows a blocking port to be held in a *standby* state. If the root port fails, the blocking port can *immediately* transition to a forwarding state. Thus, UplinkFast improves convergence time for *direct* failures in the STP topology.

If *multiple* ports are in a blocking state, whichever port has the lowest root path cost will transition to forwarding.

UplinkFast is *disabled* by default, and must be enabled globally for all VLANs on the switch:

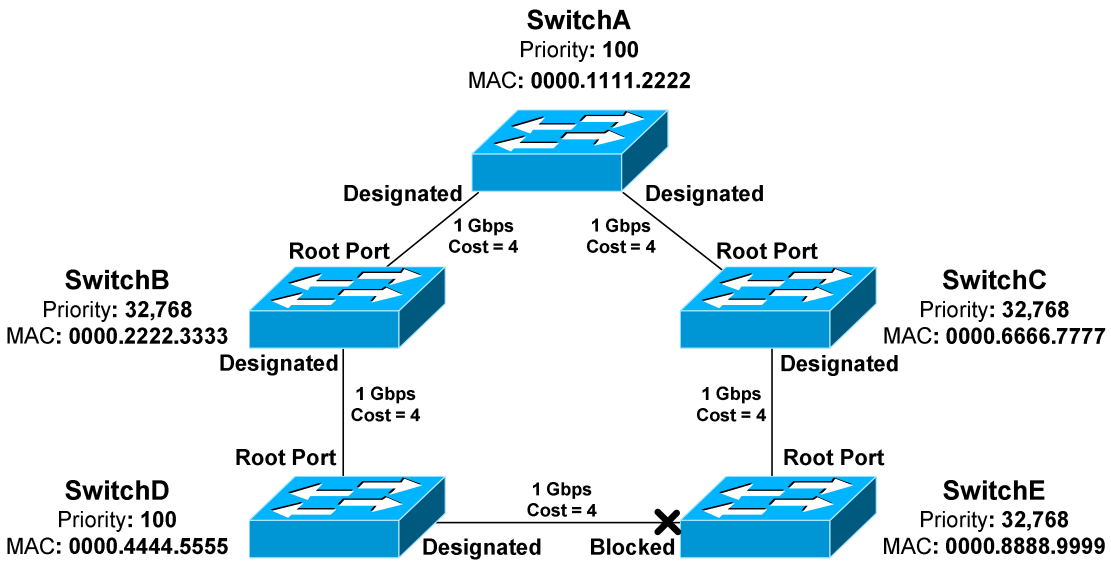
```
Switch(config)#  spanning-tree uplinkfast
```

UplinkFast functions by tracking all possible links *to* the Root Bridge. Thus, UplinkFast is **not supported** on the Root Bridge. In fact, enabling this feature will automatically increase a switch's bridge priority to 49,152.

UplinkFast is intended for the *furthest downstream* switches in the STP topology.

BackboneFast

UplinkFast provides faster convergence if a *directly-connected* port fails. In contrast, **BackboneFast** provides improved convergence if there is an *indirect* failure in the STP topology.



If the link between SwitchB and SwitchA fails, SwitchD will eventually recalculate a path through SwitchE to reach the Root Bridge. However, SwitchD must wait the max age timer before purging SwitchB's superior BPDU information. By default, this is **20 seconds**.

BackboneFast allows a switch to bypass the max age timer. The switch will accept SwitchE's inferior BPDU's immediately. The blocked port on SwitchE must *still* transition to a forwarding state. Thus, BackboneFast essentially reduces total convergence time from 50 seconds to 30 seconds for an indirect failure.

This is accomplished by sending out **Root Link Queries (RLQs)**. The Root Bridge will respond to these queries with a **RLQ Reply**:

- If a RLQ Reply is received on a root port, the switch knows that the root path is stable.
- If a RLQ Reply is received on a non-root port, the switch knows that the root path has failed. The max age timer is immediately expired to allow a new root port to be elected.

BackboneFast is a *global* command, and should be enabled on *every* switch:

```
Switch(config)#  spanning-tree backbonefast
```

* * *

Protecting STP

STP is vulnerable to attack for two reasons:

- STP builds the topology by accepting BPDUs from neighboring switches.
- The Root Bridge is always determined by the lowest Bridge ID.

A switch with a low priority can be maliciously or inadvertently installed on the network, and then elected as the Root Bridge. STP will reconverge, often resulting in instability or a suboptimal topology.

Cisco implemented three mechanisms to protect the STP topology:

- **Root Guard**
- **BPDU Guard**
- **BPDU Filtering**

All three mechanisms are configured on a per-port basis, and are *disabled* by default.

Root Guard

Root Guard prevents an unauthorized switch from advertising itself as a Root Bridge. If a BPDU *superior* to the Root Bridge is received on a port with Root Guard enabled, the port is placed in a *root-inconsistent* state.

In this state, the port is essentially in a *blocking* state, and will not forward frames. The port can still listen for BPDUs.

Root Guard is enabled on a per-port basis, and is disabled by default:

```
Switch(config)#  interface gi1/14
Switch(config-if)#  spanning-tree guard root
```

To view all ports that have been placed in a *root-inconsistent* state:

```
Switch#  show spanning-tree inconsistentports

Name                 Interface            Inconsistency
-------------------- -------------------- ------------------
VLAN100              GigabitEthernet1/14  Root Inconsistent
```

Root Guard can automatically recover. As soon as superior BPDUs are no longer received, the port will transition normally through STP states.

BPDU Guard

Recall that **PortFast** allows a switch port to bypass the usual progression of STP states. However, PortFast does *not* disable STP on a port - it merely accelerates STP convergence. However, a PortFast-enabled port will still accept BPDUs.

PortFast should *only* be enabled on ports connected to a host. If enabled on a port connecting to a switch, any loop may result in a broadcast storm.

To prevent such a scenario, **BPDU Guard** can be used in conjunction with PortFast. Under normal circumstances, a port with PortFast enabled should *never* receive a BPDU, as it is intended only for hosts.

BPDU Guard will place a port in an **errdisable** state if a BPDU is received, regardless if the BPDU is superior or inferior. The STP topology will not be impacted by another switch that is inadvertently connected to that port.

BPDU Guard should be enabled on any port with PortFast enabled. It is *disabled* by default, and can be enabled on a per-interface basis:

```
Switch(config)#  interface gi1/14
Switch(config-if)#  spanning-tree bpduguard enable
```

If BPDU Guard is enabled *globally*, it will only apply to PortFast ports:

```
Switch(config)#  spanning-tree portfast bpduguard default
```

An interface can be *manually* recovered from an errdisable state by performing a *shutdown* and then *no shutdown*:

```
Switch(config)#  interface gi1/14
Switch(config-if)#  shutdown
Switch(config-if)#  no shutdown
```

BPDUs will still be *sent* out ports enabled with BPDU Guard.

BPDU Filtering

BPDU Filtering prevents BPDUs from being *sent* out a port, and must be enabled in conjunction with PortFast.

If a BPDU is *received* on a port, BPDU Filtering will react one of two ways, depending on how it was *configured.*

- If filtering is enabled *globally*, a received BPDU will disable PortFast on the port. The port will then **transition normally** through the STP process.
- If filtering is enabled on a *per-interface* basis, a received BPDU is **ignored.**

Great care must be taken when manually enabling BPDU Filtering on a port. Because the port will *ignore* a received BPDU, **STP is essentially disabled.** The port will neither be err-disabled nor progress through the STP process, and thus the port is susceptible to loops.

If BPDU Filtering is enabled *globally*, it will only apply to PortFast ports:

```
Switch(config)#  spanning-tree portfast bpdufilter default
```

To enable BPDU Filtering on a *per-interface* basis:

```
Switch(config)#  interface gi1/15
Switch(config-if)#  spanning-tree bpdufilter enable
```

Unidirectional Link Detection (UDLD)

Most network communication is **bidirectional**. Occasionally, a hardware fault will cause traffic to be transmitted in only one direction, or **unidirectional.** Fiber ports are the most susceptible to this type of fault.

STP requires that switches exchange BPDUs bidirectionally. If a port becomes unidirectional, BPDUs will not be received by one of the switches. That switch may then incorrectly transition a *blocking* port to a *forwarding* state, and create a loop.

Cisco developed **Unidirectional Link Detection (UDLD)** to ensure that bidirectional communication is maintained. UDLD sends out **ID frames** on a port, and waits for the remote switch to respond with its own ID frame. If the remote switch does not respond, UDLD assumes the port has a unidirectional fault.

By default, UDLD sends out ID frames every **15 seconds** on most Cisco platforms. Some platforms default to every **7 seconds.** UDLD must be enabled on *both* sides of a link.

UDLD reacts one of two ways when a unidirectional link is detected:

- **Normal Mode –** the port is *not* shut down, but is flagged as being in an *undetermined* state.
- **Aggressive Mode –** the port is placed in an *errdisable* state

UDLD can be enabled globally, though it will only apply for fiber ports:

```
Switch(config)#  udld enable message time 20
Switch(config)#  udld aggressive message time 20
```

The *enable* parameter sets UDLD into normal mode, and the *aggressive* parameter is for aggressive mode. The *message time* parameter modifies how often ID frames are sent out, measured in seconds.

UDLD can be configured on a per-interface basis:

```
Switch(config-if)#  udld enable
Switch(config-if)#  udld aggressive
Switch(config-if)#  udld disable
```

To view UDLD status on ports, and reset any ports disabled by UDLD:

```
Switch#  show udld
Switch#  udld reset
```

Loop Guard

STP relies on the exchange of BPDUs to maintain a loop free environment.

If a software or hardware failure causes a switch to stop receiving BPDUs, a switch will eventually discard that BPDU information, after the max age timer has expired.

This may result in the switch incorrectly transitioning a *blocking* port to a *forwarding* state, thus creating a loop.

UDLD addresses only one of the possible causes of this scenario – a unidirectional link. Other issues may prevent BPDUs from being received or processed, such as the CPU on a switch being at max utilization.

Loop Guard provides a more comprehensive solution – if a blocking port stops receiving BPDUs on a VLAN, it is moved into a *loop-inconsistent* state for that VLAN.

A port in a *loop-inconsistent* state cannot forward traffic for the affected VLANs, and is essentially in a pseudo-errdisable state.

However, Loop Guard can automatically recover. As soon as BPDUs are received again, the port will transition normally through STP states.

Loop Guard can be enabled globally:

```
Switch(config)#  spanning-tree loopguard default
```

Loop Guard can also be enabled on a per-interface basis:

```
Switch(config)#  interface gi2/23
Switch(config-if)#  spanning-tree guard loop
```

Loop Guard should only be enabled on trunk ports, or ports that connect to other switches. Loop Guard should never be enabled on a port connecting to a host, as an access port should never receive a BPDU.

(Reference: http://astorinonetworks.com/2011/09/01/understanding-spanning-tree-loopguard/)

Troubleshooting STP

To view general STP information for all VLANs:

Switch# *show spanning-tree*

```
VLAN0101
  Spanning tree enabled protocol ieee
  Root ID    Priority    32869
             Address     000a.f43b.1b80
             This bridge is the root
             Hello Time  2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    32869  (priority 32768 sys-id-ext 101)
             Address     000a.f43b.1b80
             Hello Time  2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time 300

  Interface        Role Sts Cost      Prio.Nbr Type
  ---------------- ---- --- --------- -------- --------------------------
  Gi2/23           Desg FWD 4         128.47   P2p

  <snipped for brevity>
```

To view more detailed STP information:

Switch# *show spanning-tree*

```
VLAN0101 is executing the ieee compatible Spanning Tree protocol
  Bridge Identifier has priority 32768, sysid 101, address 000a.f43b.1b80
  Configured hello time 2, max age 20, forward delay 15
  We are the root of the spanning tree
  Topology change flag not set, detected flag not set
  Number of topology changes 1 last change occurred 1w6d ago
          from GigabitEthernet2/23
  Times:  hold 1, topology change 35, notification 2
          hello 2, max age 20, forward delay 15
  Timers: hello 0, topology change 0, notification 0, aging 300

 Port 47 (GigabitEthernet2/23) of VLAN0101 is forwarding
   Port path cost 4, Port priority 128, Port Identifier 128.47.
   Designated root has priority 32869, address 000a.f43b.1b80
   Designated bridge has priority 32869, address 000a.f43b.1b80
   Designated port id is 128.47, designated path cost 0
   Timers: message age 0, forward delay 0, hold 0
   Number of transitions to forwarding state: 1
   Link type is point-to-point by default
   BPDU: sent 1129012, received 0
```

To view STP information specific to an interface:

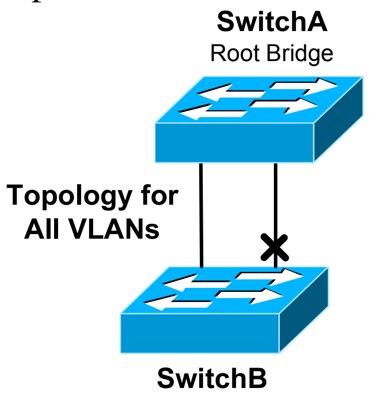
Switch# *show spanning-tree interface gi2/24*

```
Vlan                 Role Sts Cost      Prio.Nbr Type
-------------------- ---- --- --------- -------- ------------------
VLAN0101             Root FWD 4         128.48   P2p
VLAN0102             Altn BLK 4         128.48   P2p
```

Per-VLAN Spanning Tree (PVST) Load Balancing

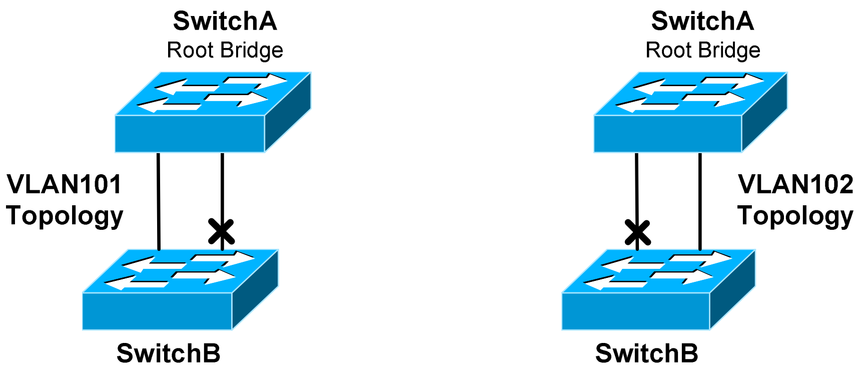
PVST and **PVST+** employ a *separate* STP instance for *each* VLAN. This provides superior flexibility over CST, which only supports a single STP instance for *all* VLANs.

Consider the following example:



If a port on SwitchB enters a *blocking* state to eliminate the loop, that port will block traffic from *all* VLANs. Redundancy is not lost, as STP will recognize if the non-blocked port goes down, and reactivate the blocked port.

However, this is *inefficient*, as the potential bandwidth of the blocked port is unavailable for any VLAN. In contrast, PVST supports load balancing VLANs across the switching topology:



PVST runs a separate instance for each VLAN, allowing a port to enter a blocking state **only for that specific VLAN.** This provides both redundancy and more efficient use of available bandwidth.

Note: An even better solution for the above example is to use an **EtherChannel**, which STP will treat as a single logical interface.

Rapid Spanning Tree Protocol (RSTP)

In modern networks, a 30 to 50 second convergence delay is unacceptable. Enhancements were made to the original IEEE 802.1D standard to address this. The result was **802.1w**, or **Rapid Spanning Tree Protocol (RSTP)**.

RSTP is similar in many respects to STP:

- BPDUs are forwarded between switches
- A Root Bridge is elected, based on the lowest Bridge ID.
- Root and designated ports are elected and function identically to STP.

RSTP defines four **port roles**:

- **Root Port –** Port on each switch that has the best path cost to the Root Bridge. A switch can only have one root port.
- **Alternate Port –** Backup root port that has a less desirable path cost.
- **Designated Port –** Non-root port that represents the best path cost for each *network segment* to the Root Bridge.
- **Backup Port –** Backup designated port that has a less desirable path cost.

802.1D STP supported five port states, while RSTP supports **three:**

- **Discarding**
- **Learning**
- **Forwarding**

Initially, a switch port starts in a **discarding** state:

- A discarding port *will not* forward frames or learn MAC addresses.
- A discarding port will listen for BPDUs.
- Alternate and backup ports will remain in a discarding state.

RSTP does not need a listening state. Instead, if a port is elected as a *root* or *designated* port, it will transition from discarding to a **learning** state:

- A learning port *will begin* to add MAC addresses to the CAM table.
- However, a learning port *cannot* forward frames quite yet.

Finally, a learning port will transition to a **forwarding** state:

- A forwarding port is fully functional – it will send and listen for BPDUs, learn MAC addresses, and forward frames.
- Root and designated ports will eventually transition to a forwarding state.

Rapid Spanning Tree Protocol (RSTP) (continued)

The key benefit of RSTP is faster convergence:

- BPDUs are generated by *every* switch, and sent out at the hello interval.
- Switches no longer require artificial forward delay timers.

In 802.1D, BPDUs are generated by the Root Bridge. If a switch receives a BPDU from the Root Bridge on its root port, it will propagate the BPDU downstream to *its* neighbors. This convergence process is *slow*, and STP relies on forward delay timers to ensure a loop-free environment.

In RSTP, switches will **handshake** directly with their neighbors, allowing the topology to be quickly synchronized. This allows ports to rapidly transition from a discarding state to a forwarding state without a delay timer.

A key component of the RSTP process is the **type** of each port:

- **Edge –** port that connects to a *host*. This port behaves exactly like a PortFast-enabled port, transitioning to a forwarding state immediately.
- **Root –** port that connects to another *switch,* and has the best path cost to the Root Bridge.
- **Point-to-Point** – port that connects to another *switch*, with the potential to become the designated port for a segment.

Note: If an edge port receives a BPDU, it will lose its edge port status and transition normally through the RSTP process. On Cisco switches, any port configured with PortFast becomes an Edge Port.

The RSTP convergence process is as follows:

- Switches exchange BPDUs to elect the Root Bridge.
- Edge ports immediately transition into a forwarding state.
- All potential root and point-to-point ports start in a *discarding* state.
- If a port receives a *superior BPDU*, it will become a root port, and transition immediately to a *forwarding* state.
- For a point-to-point port, each switch will exchange a *handshake* proposal to determine which port becomes designated.
- Once the switches agree, the designated port is moved immediately into a forwarding state.

Every switch will perform this handshaking process with each of its neighbors, until all switches are synchronized. Complete convergence happens very quickly – within seconds.

Rapid Spanning Tree Protocol (RSTP) (continued)

RSTP handles topology *changes* more efficiently than 802.1D STP, which generates a Topology Change Notification (TCN) in two circumstances:

- When a port transitions into a *forwarding* state.
- When a port transitions into a *blocking* or *down* state.

The TCN will eventually reach the Root Bridge, which will then inform all other switches of the change by sending a BPDU with the Topology Change (TC) bit set.

In RSTP, only a **non-edge port transitioning to a *forwarding* state** will generate a TCN. The switch recognizing a topology change *does not* have to inform the Root Bridge first. Any switch can generate and forward a TC BPDU, allowing the topology to quickly converge via handshakes.

A switch receiving a TC BPDU will **flush all MAC addresses learned on designated ports**, except for the port that received the TC BPDU.

In the event of a topology change, RSTP will allow alternate or backup ports to *immediately* enter a forwarding state. Additionally, RSTP does not have to wait an arbitrary max age timer to accept an inferior BPDU, if there is an indirect failure in the topology.

Essentially, RSTP inherently supports the functionality of UplinkFast and BackboneFast.

RSTP is compatible with 802.1D STP. If a neighboring switch does not respond to an RSTP handshake, a port reverts back to transitioning through 802.1D states. Note that this means that all RSTP benefits are lost on that port.

Two implementations of RSTP exist:

- **Rapid Per-VLAN Spanning Tree Protocol (RPVST+)**
- **Multiple Spanning Tree (MST)**

RPVST+ is Cisco proprietary, while MST is defined in the IEEE 802.1s standard.

To enable RPVST+ globally on a switch:

```
Switch(config)# spanning-tree mode rapid-pvst
```

Multiple Spanning Tree (MST)

Earlier in this guide, three versions of 802.1D STP were described:

- **CST** utilizes a *single* STP instance for *all* VLANs**.**
- **PVST** and **PVST+** employ a *separate* STP instance for *each* VLAN**.**

PVST and PVST+ are more efficient, and allow STP to load balance VLANs across links. This comes at a cost – maintaining a separate STP instance for each VLAN adds overhead to the CPU and memory on a switch.

Multiple Spanning Tree (MST), defined in **IEEE 802.1s**, allows a *group* of VLANs to be **mapped** to an STP instance.

Each **MST instance (MSTI)** builds its own **RSTP** topology database, including electing its own Root Bridge. A VLAN can only be assigned to *one* instance.

MST further separates the STP topology into **regions.** All switches in a region must be configured with *identical* MST parameters:

- 32-byte **configuration name**
- 16-bit **revision number**
- **VLAN-to-instance mapping database**

If two switches are configured with *different* MST parameters, they belong to *different* MST regions.

For most Cisco platforms, a region can contain a maximum of **16 MST instances**, numbered *0* through *15.* By default, all VLANs belong to **instance 0**.

The **Internal Spanning Tree (IST)** is responsible for maintaining the topology for the *entire* region and all of the MSTIs. Only the IST can send and receive BPDUs, and encapsulates the MSTI information within a BPDU as an **MST record (M-record).**

The IST is always mapped to **instance 0**.

MST is compatible with all other implementations of STP. An MST region is *obfuscated* from non-MST switches, which will see the entire MST region as a **single 802.1D or RSTP switch.**

Multiple Spanning Tree (MST) (continued)

To enable MST globally on a switch:

Switch(config)# *spanning-tree mode mst*

Changes to MST parameters must be made from *MST configuration mode:*

Switch(config)# *spanning-tree mst configuration*
Switch(config-mst)#

To assign the MST configuration name and revision number:

Switch(config-mst)# *name MYMSTNAME*
Switch(config-mst)# *revision 2*

To map VLANs to a specific MST instances:

Switch(config-mst)# *instance 2 vlan 1-100*
Switch(config-mst)# *instance 3 vlan 101-200*

Remember: A maximum of 16 MST instances are supported, numbered *0* to *15*. The MST configuration name, revision number, and VLAN-to-instance mapping must be identical on all switches in the same region.

To view the changes to the configuration:

Switch(config-mst)# *show pending*

```
Pending MST configuration
Name [MYMSTNAME]
Revision 2
Instance   Vlans mapped
--------   ------------------------
0          201-4094
2          1-100
3          101-200
```

All other MST parameters are configured *identically* to 802.1D STP, with two exceptions:

- The *mst* parameter must be used on all commands
- All commands reference the MST **instance** instead of a **VLAN**.

Thus, to configure a switch as the Root Bridge for *MST* instance *2:*

Switch(config)# *spanning-tree mst 2 root primary*