Cisco CCNP Switching Study Guide

v2.01 © 2014

Aaron Balchunas
aaron@routeralley.com
http://www.routeralley.com

Foreword:

This study guide is intended to provide those pursuing the CCNP certification with a framework of what concepts need to be studied. This is ***not*** *a comprehensive document containing all the secrets of the CCNP Switching exam, nor is it a "braindump" of questions and answers.*

This document is freely given, and can be freely distributed. However, the contents of this document ***cannot*** *be altered, without my written consent. Nor can this document be sold or published without my expressed consent.*

I sincerely hope that this document provides some assistance and clarity in your studies.

* * *

Table of Contents

* * *

Part I

General Switching Concepts

* * *

Section 1
- Ethernet Technologies -

What is Ethernet?

Ethernet is a family of technologies that provides data-link and physical specifications for controlling access to a shared network medium. It has emerged as the dominant technology used in LAN networking.

Ethernet was originally developed by Xerox in the 1970s, and operated at 2.94Mbps. The technology was standardized as **Ethernet Version 1** by a consortium of three companies - DEC, Intel, and Xerox, collectively referred to as **DIX -** and further refined as **Ethernet II** in 1982.

In the mid 1980s, the **Institute of Electrical and Electronic Engineers (IEEE)** published a formal standard for Ethernet, defined as the **IEEE 802.3** standard. The original 802.3 Ethernet operated at 10Mbps, and successfully supplanted competing LAN technologies, such as Token Ring.

Ethernet has several benefits over other LAN technologies:

- Simple to install and manage
- Inexpensive
- Flexible and scalable
- Easy to interoperate between vendors

(References: http://docwiki.cisco.com/wiki/Ethernet_Technologies; http://www.techfest.com/networking/lan/ethernet1.htm)

Ethernet Cabling Types

Ethernet can be deployed over three types of cabling:

- **Coaxial** cabling – *almost entirely deprecated in Ethernet networking*
- **Twisted-pair** cabling
- **Fiber optic** cabling

Coaxial cable, often abbreviated as *coax*, consists of a single wire surrounded by insulation, a metallic shield, and a plastic sheath. The shield helps protect against **electromagnetic interference (EMI),** which can cause **attenuation,** a reduction of the strength and quality of a signal. EMI can be generated by a variety of sources, such as florescent light ballasts, microwaves, cell phones, and radio transmitters.

Coax is commonly used to deploy cable television to homes and businesses.

* * *

Ethernet Cabling Types (continued)

Two types of coax were used historically in Ethernet networks:

- **Thinnet**
- **Thicknet**

Thicknet has a wider diameter and more shielding, which supports greater distances. However, it is less flexible than the smaller *thinnet*, and thus more difficult to work with. A **vampire tap** is used to physically connect devices to thicknet, while a **BNC** connector is used for thinnet.

Twisted-pair cable consists of two or four pairs of copper wires in a plastic sheath. Wires in a pair *twist* around each other to reduce **crosstalk**, a form of EMI that occurs when the signal from one wire *bleeds* or *interferes* with a signal on another wire. Twisted-pair is the most common Ethernet cable.

Twisted-pair cabling can be either **shielded** or **unshielded**. Shielded twisted-pair is more resistant to external EMI; however, all forms of twisted-pair suffer from greater signal attenuation than coax cable.

There are several *categories* of twisted-pair cable, identified by the number of *twists per inch* of the copper pairs:

- **Category 3** or **Cat3** - three twists per inch.
- **Cat5** - five twists per inch.
- **Cat5e** - five twists per inch; pairs are also twisted around each other.
- **Cat6 –** six twists per inch, with improved insulation.

An **RJ45** connector is used to connect a device to a twisted-pair cable. The *layout* of the wires in the connector dictates the function of the cable.

While coax and twisted-pair cabling carry *electronic* signals, **fiber optics** uses *light* to transmit a signal. Ethernet supports two fiber specifications:

- **Singlemode fiber** – consists of a very small glass *core*, allowing only a single ray or *mode* of light to travel across it. This greatly reduces the attenuation and dispersion of the light signal, supporting high bandwidth over *very* long distances, often measured in kilometers.
- **Multimode fiber** – consists of a larger core, allowing multiple modes of light to traverse it. Multimode suffers from greater dispersion than singlemode, resulting in shorter supported distances.

Singlemode fiber requires more *precise* electronics than multimode, and thus is significantly more *expensive*. Multimode fiber is often used for high-speed connectivity within a datacenter.

* * *

Network Topologies

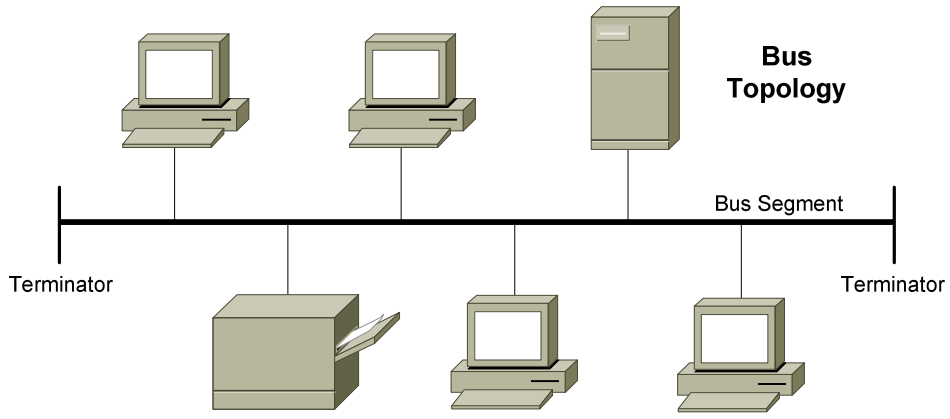
A **topology** defines both the *physical* and *logical* structure of a network. Topologies come in a variety of configurations, including:

- Bus
- Star
- Ring
- Full or partial mesh

Ethernet supports two topology types – **bus** and **star**.

Ethernet Bus Topology

In a **bus topology**, all hosts share a single physical segment (the *bus* or the *backbone*) to communicate:



A frame sent by one host is received by *all other* hosts on the bus. However, a host will only *process* a frame if it matches the destination hardware address in the data-link header.

Bus topologies are inexpensive to implement, but are almost entirely deprecated in Ethernet. There are several disadvantages to the bus topology:

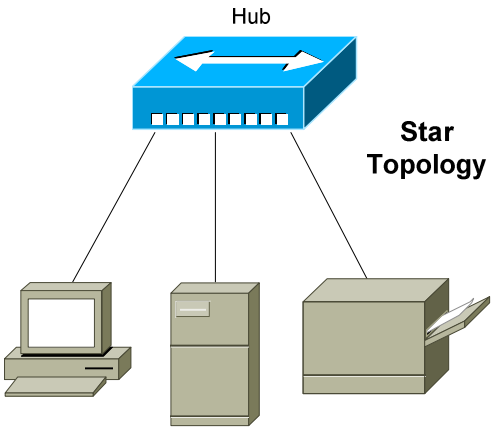
- Both ends of the bus must be **terminated,** otherwise a signal will *reflect* back and cause interference, severely degrading performance.
- Adding or removing hosts to the bus can be difficult.
- The bus represents a single point of failure - a break in the bus will affect *all* hosts on the segment. Such faults are often very difficult to troubleshoot.

A bus topology is implemented using either thinnet or thicknet coax cable.

* * *

Ethernet Star Topology

In a **star topology**, each host has an individual point-to-point connection to a centralized *hub* or *switch*:



A **hub** provides no intelligent forwarding whatsoever, and will always forward every frame out every port, excluding the port originating the frame. As with a bus topology, a host will only *process* a frame if it matches the destination hardware address in the data-link header. Otherwise, it will discard the frame.

A **switch** builds a **hardware address table,** allowing it to make intelligent forwarding decisions based on frame (data-link) headers. A frame can then be forwarded out *only* the appropriate destination port, instead of *all* ports.

Hubs and switches are covered in great detail in another guide.

Adding or removing hosts is very simple in a star topology. Also, a break in a cable will affect *only that one host*, and not the entire network.

There are two disadvantages to the star topology:

- The hub or switch represents a single point of failure.
- Equipment and cabling costs are generally higher than in a bus topology.

However, the star is still the dominant topology in modern Ethernet networks, due to its flexibility and scalability. Both twisted-pair and fiber cabling can be used in a star topology.

* * *

The Ethernet Frame

An Ethernet frame contains the following fields:

Field	***Length***	***Description***
Preamble	7 bytes	Synchronizes communication
Start of Frame	1 byte	Signals the start of a valid frame
MAC Destination	6 bytes	Destination MAC address
MAC Source	6 bytes	Source MAC address
802.1Q tag	4 bytes	Optional VLAN tag
Ethertype or length	2 bytes	Payload type or frame size
Payload	42-1500 bytes	Data payload
CRC	4 bytes	Frame error check
Interframe Gap	12 bytes	Required idle period between frames

The **preamble** is 56 bits of alternating 1s and 0s that synchronizes communication on an Ethernet network. It is followed by an 8-bit **start of frame delimiter** (10101011) that indicates a valid frame is about to begin. The preamble and the start of frame are *not considered* part of the actual frame, or calculated as part of the total frame size.

Ethernet uses the 48-bit **MAC address** for hardware addressing. The first 24-bits of a MAC address determine the manufacturer of the network interface, and the last 24-bits uniquely identify the host.

The *destination* MAC address identifies who is to receive the frame - this can be a single host (a *unicast*), a group of hosts (a *multicast*), or all hosts (a *broadcast*). The *source* MAC address indentifies the host originating the frame.

The **802.1Q tag** is an *optional* field used to identify which **VLAN** the frame belongs to. VLANs are covered in great detail in another guide.

The 16-bit **Ethertype/Length field** provides a different function depending on the standard - Ethernet II or 802.3. With Ethernet II, the field identifies the type of payload in the frame (the *Ethertype).* However, Ethernet II is almost entirely deprecated.

With 802.3, the field identifies the *length* of the payload. The length of a frame is important – there is both a *minimum* and *maximum* frame size.

(Reference: http://www.techfest.com/networking/lan/ethernet2.htm; http://www.dcs.gla.ac.uk/~lewis/networkpages/m04s03EthernetFrame.htm)

* * *

The Ethernet Frame (continued)

Field	***Length***	***Description***
Preamble	7 bytes	Synchronizes communication
Start of Frame	1 byte	Signals the start of a valid frame
MAC Destination	6 bytes	Destination MAC address
MAC Source	6 bytes	Source MAC address
802.1Q tag	4 bytes	Optional VLAN tag
Ethertype or length	2 bytes	Payload type or frame size
Payload	42-1500 bytes	Data payload
CRC	4 bytes	Frame error check
Interframe Gap	12 bytes	Required idle period between frames

The absolute **minimum frame size** for Ethernet is **64 bytes** (or **512 bits**) including headers. A frame that is smaller than 64 bytes will be discarded as a **runt**. The required fields in an Ethernet header add up to 18 bytes – thus, the frame **payload** must be a minimum of 46 bytes, to equal the minimum 64-byte frame size. If the payload *does not* meet this minimum, the payload is **padded** with **0 bits** until the minimum is met.

Note: If the optional 4-byte 802.1Q tag is used, the Ethernet header size will total 22 bytes, requiring a minimum payload of 42 bytes.

By default, the **maximum frame size** for Ethernet is **1518 bytes –** *18 bytes* of header fields, and *1500 bytes* of payload - or **1522 bytes** with the 802.1Q tag. A frame that is larger than the maximum will be discarded as a **giant.** With both runts and giants, the receiving host will *not* notify the sender that the frame was dropped. Ethernet relies on higher-layer protocols, such as TCP, to provide retransmission of discarded frames.

Some Ethernet devices support **jumbo frames** of **9216 bytes**, which provide less overhead due to fewer frames. Jumbo frames must be explicitly enabled on *all* devices in the traffic path to prevent the frames from being dropped.

The 32-bit **Cycle Redundancy Check (CRC)** field is used for error-detection. A frame with an invalid CRC will be discarded by the receiving device. This field is a *trailer,* and not a *header*, as it follows the payload.

The 96-bit **Interframe Gap** is a required idle period between frame transmissions, allowing hosts time to prepare for the next frame.

(Reference: http://www.infocellar.com/networks/ethernet/frame.htm)

* * *

CSMA/CD and Half-Duplex Communication

Ethernet was originally developed to support a **shared media** environment. This allowed two or more hosts to use the same physical network medium.

There are two methods of communication on a shared physical medium:

- **Half-Duplex –** hosts can transmit or receive, but *not simultaneously*
- **Full-Duplex –** hosts can both transmit and receive simultaneously

On a half-duplex connection, Ethernet utilizes **Carrier Sense Multiple Access with Collision Detect (CSMA/CD)** to control media access. *Carrier sense* specifies that a host will monitor the physical link, to determine whether a *carrier* (or *signal)* is currently being transmitted. The host will *only* transmit a frame if the link is **idle**, and the Interframe Gap has expired.

If two hosts transmit a frame simultaneously, a **collision** will occur. This renders the collided frames unreadable. Once a collision is detected, both hosts will send a **32-bit jam sequence** to ensure all transmitting hosts are aware of the collision. The collided frames are also discarded.

Both devices will then wait a *random* amount of time before resending their respective frames, to reduce the likelihood of another collision. This is controlled by a **backoff** timer process.

Hosts *must* detect a collision before a frame is finished transmitting, otherwise CSMA/CD cannot function reliably. This is accomplished using a consistent **slot time,** the time required to send a specific amount of data from one end of the network and then *back*, measured in bits.

A host must continue to transmit a frame for a *minimum* of the slot time. In a properly configured environment, a collision should *always* occur within this slot time, as enough time has elapsed for the frame to have reached the far end of the network and back, and thus all devices should be aware of the transmission. The slot time effectively limits the physical length of the network – if a network segment is too long, a host may not detect a collision within the slot time period. A collision that occurs after the slot time is referred to as a **late collision.**

For 10 and 100Mbps Ethernet, the slot time was defined as **512 bits,** or 64 bytes. Note that this is the equivalent of the *minimum Ethernet frame size* of 64 bytes. The slot time actually defines this minimum. For Gigabit Ethernet, the slot time was defined as **4096 bits.**

(Reference: *http://www.techfest.com/networking/lan/ethernet3.htm*)

* * *

Full-Duplex Communication

Unlike half-duplex, **full-duplex** Ethernet supports simultaneously communication by providing separate transmit and receive paths. This effectively *doubles* the throughput of a network interface.

Full-duplex Ethernet was formalized in IEEE 802.3x, and *does not use CSMA/CD* or slot times. Collisions should *never* occur on a functional full-duplex link. Greater distances are supported when using full-duplex over half-duplex.

Full-duplex is only supported on a point-to-point connection between two devices. Thus, a bus topology using coax cable *does not* support full-duplex.

Only a connection **between two hosts** or between **a host and a *switch*** supports full-duplex. A host connected to a *hub* is limited to half-duplex. Both hubs and half-duplex communication are mostly deprecated in modern networks.

Categories of Ethernet

The original 802.3 Ethernet standard has evolved over time, supporting faster transmission rates, longer distances, and newer hardware technologies. These *revisions* or *amendments* are identified by the letter appended to the standard, such as 802.3*u* or 802.3*z*.

Major categories of Ethernet have also been organized by their speed:

- **Ethernet** (10Mbps)
- **Fast Ethernet** (100Mbps)
- **Gigabit Ethernet**
- **10 Gigabit Ethernet**

The *physical* standards for Ethernet are often labeled by their transmission rate, signaling type, and media type. For example, *100baseT* represents the following:

- The first part (*100*) represents the transmission rate, in Mbps.
- The second part (*base)* indicates that it is a baseband transmission.
- The last part (*T)* represents the physical media type (*twisted-pair)*.

Ethernet communication is **baseband**, which dedicates the entire capacity of the medium to one signal or channel. In **broadband**, multiple signals or channels can share the same link, through the use of *modulation* (usually frequency modulation).

* * *

Ethernet (10 Mbps)

Ethernet is now a somewhat generic term, describing the entire family of technologies. However, Ethernet traditionally referred to the original 802.3 standard, which operated at **10 Mbps**. Ethernet supports coax, twisted-pair, and fiber cabling. Ethernet over twisted-pair uses **two** of the four pairs.

Common Ethernet physical standards include:

IEEE Standard	*Physical Standard*	*Cable Type*	*Maximum Speed*	*Maximum Cable Length*
802.3a	10base2	Coaxial (thinnet)	10 Mbps	185 meters
802.3	10base5	Coaxial (thicknet)	10 Mbps	500 meters
802.3i	10baseT	Twisted-pair	10 Mbps	100 meters
802.3j	10baseF	Fiber	10 Mbps	2000 meters

Both 10baseT and 10baseF support full-duplex operation, effectively doubling the bandwidth to 20 Mbps. Remember, only a connection **between two hosts** or between **a host and a *switch*** support full-duplex. The maximum distance of an Ethernet segment can be extended through the use of a **repeater**. A hub or a switch can also serve as a repeater.

Fast Ethernet (100 Mbps)

In 1995, the IEEE formalized **802.3u**, a **100 Mbps** revision of Ethernet that became known as **Fast Ethernet.** Fast Ethernet supports both twisted-pair copper and fiber cabling, and supports both half-duplex and full-duplex.

Common Fast Ethernet physical standards include:

IEEE Standard	*Physical Standard*	*Cable Type*	*Maximum Speed*	*Maximum Cable Length*
802.3u	100baseTX	Twisted-pair	100 Mbps	100 meters
802.3u	100baseT4	Twisted-pair	100 Mbps	100 meters
802.3u	100baseFX	Multimode fiber	100 Mbps	400-2000 meters
802.3u	100baseSX	Multimode fiber	100 Mbps	500 meters

100baseT4 was never widely implemented, and only supported half-duplex operation. 100baseTX is the dominant Fast Ethernet physical standard. 100baseTX uses **two** of the four pairs in a twisted-pair cable, and requires Category 5 cable for reliable performance.

* * *

Speed and Duplex Autonegotiation

Fast Ethernet is backwards-compatible with the original Ethernet standard. A device that supports both Ethernet and Fast Ethernet is often referred to as a *10/100* device.

Fast Ethernet also introduced the ability to **autonegotiate** both the speed and duplex of an interface. Autonegotiation will attempt to use the *fastest* speed available, and will attempt to use *full-duplex* if both devices support it. Speed and duplex can also be **hardcoded**, preventing negotiation.

The configuration *must* be consistent on both sides of the connection. Either both sides must be configured to autonegotiate, or both sides must be hardcoded with *identical* settings. Otherwise a **duplex mismatch** error can occur.

For example, if a workstation's NIC is configured to autonegotiate, and the switch interface is hardcoded for 100Mbps and full-duplex, then a duplex mismatch will occur. The workstation's NIC will sense the correct speed of 100Mbps, but will not detect the correct duplex and will default to *half-duplex.*

If the duplex is mismatched, collisions will occur. Because the full-duplex side of the connection does not utilize CSMA/CD, performance is *severely* degraded. These issues can be difficult to troubleshoot, as the network connection will still function, but will be excruciatingly slow.

When autonegotiation was first developed, manufacturers did not always adhere to the same standard. This resulted in frequent mismatch issues, and a sentiment of distrust towards autonegotiation.

Though modern network hardware has alleviated most of the incompatibility, many administrators are still skeptical of autonegotiation and choose to hardcode all connections. Another common practice is to hardcode server and datacenter connections, but to allow user devices to autonegotiate.

Gigabit Ethernet, covered in the next section, provided several enhancements to autonegotiation, such as hardware flow control. Most manufacturers **recommend autonegotiation** on Gigabit Ethernet interfaces as a best practice.

* * *

Gigabit Ethernet

Gigabit Ethernet operates at 1000 Mbps, and supports both twisted-pair **(802.3ab)** and fiber cabling **(802.3z)**. Gigabit over twisted-pair uses **all four pairs**, and requires Category 5e cable for reliable performance.

Gigabit Ethernet is backwards-compatible with the original Ethernet and Fast Ethernet. A device that supports all three is often referred to as a *10/100/1000* device. Gigabit Ethernet supports both half-duplex or full-duplex operation. Full-duplex Gigabit Ethernet effectively provides 2000 Mbps of throughput.

Common Gigabit Ethernet physical standards include:

IEEE Standard	***Physical Standard***	***Cable Type***	***Speed***	***Maximum Cable Length***
802.3ab	1000baseT	Twisted-pair	1 Gbps	100 meters
802.3z	1000baseSX	Multimode fiber	1 Gbps	500 meters
802.3z	1000baseLX	Multimode fiber	1 Gbps	500 meters
802.3z	1000baseLX	Singlemode fiber	1 Gbps	Several kilometers

In modern network equipment, Gigabit Ethernet has replaced both Ethernet and Fast Ethernet.

10 Gigabit Ethernet

10 Gigabit Ethernet operates at 10000 Mbps, and supports both twisted-pair **(802.3an)** and fiber cabling **(802.3ae)**. 10 Gigabit over twisted-pair uses **all four pairs**, and requires Category 6 cable for reliable performance.

Common Gigabit Ethernet physical standards include:

IEEE Standard	***Physical Standard***	***Cable Type***	***Speed***	***Maximum Cable Length***
802.3an	10Gbase-T	Twisted-pair	10 Gbps	100 meters
802.3ae	10Gbase-SR	Multimode fiber	10 Gbps	300 meters
802.3ae	10Gbase-LR	Singlemode fiber	10 Gbps	Several kilometers

10 Gigabit Ethernet is usually used for high-speed connectivity within a datacenter, and is predominantly deployed over fiber.

* * *

Twisted-Pair Cabling Overview

A typical twisted-pair cable consists of **four pairs** of copper wires, for a total of **eight wires**. Each side of the cable is terminated using an RJ45 connector, which has eight **pins.** When the connector is *crimped* onto the cable, these pins make contact with each wire.

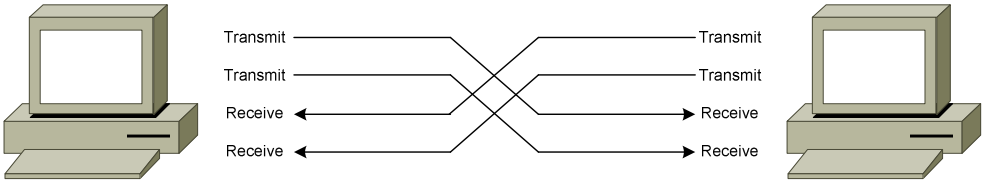
The wires themselves are assigned a *color* to distinguish them. The color is dictated by the cabling standard - **TIA/EIA-568B** is the current standard:

Color	***Pin#***
White Orange	1
Orange	2
White Green	3
Blue	4
White Blue	5
Green	6
White Brown	7
Brown	8

Each wire is assigned a specific purpose. For example, both Ethernet and Fast Ethernet use two wires to transmit, and two wires to receive data, while the other four pins remain unused.

For communication to occur, *transmit* pins must connect to the *receive* pins of the remote host. This does not occur in a **straight-through** configuration:

The pins must be **crossed-over** for communication to be successful:



The *crossover* can be controlled either by the cable, or an intermediary device, such as a hub or switch.

* * *

Twisted-Pair Cabling – Cable and Interface Types

The *layout* or *pinout* of the wires in the RJ45 connector dictates the **function** of the cable. There are three common types of twisted-pair cable:

- **Straight-through** cable
- **Crossover** cable
- **Rollover** cable

The network *interface* type determines when to use each cable:

- **Medium Dependent Interface (MDI)**
- **Medium Dependent Interface with Crossover (MDIX)**

Host interfaces are generally MDI, while hub or switch interfaces are typically MDIX.

Twisted-Pair Cabling – Straight-Through Cable

A **straight-through** cable is used in the following circumstances:

- From a host to a hub – *MDI to MDIX*
- From a host to a switch - *MDI to MDIX*
- From a router to a hub - *MDI to MDIX*
- From a router to a switch - *MDI to MDIX*

Essentially, a straight-through cable is used to connect *any device* to a hub or switch, *except* for another hub or switch. The hub or switch provides the *crossover* (or *MDIX*) function to connect transmit pins to receive pins.

The pinout on each end of a straight-through cable **must be identical**. The TIA/EIA-568B standard for a straight-through cable is as follows:

Pin#	***Connector 1***		***Connector 2***	***Pin#***
1	White Orange	----------	White Orange	1
2	Orange	----------	Orange	2
3	White Green	----------	White Green	3
4	Blue	----------	Blue	4
5	White Blue	----------	White Blue	5
6	Green	----------	Green	6
7	White Brown	----------	White Brown	7
8	Brown	----------	Brown	8

A straight-through cable is often referred to as a **patch cable.**

* * *

Twisted-Pair Cabling – Crossover Cable

A **crossover** cable is used in the following circumstances:

- From a host to a host – *MDI to MDI*
- From a hub to a hub - *MDIX to MDIX*
- From a switch to a switch - *MDIX to MDIX*
- From a hub to a switch - *MDIX to MDIX*
- From a router to a router - *MDI to MDI*

Remember that a hub or a switch will provide the crossover function. However, when connecting a host directly to another host (MDI to MDI), the crossover function must be provided by a crossover cable.

A crossover cable is often required to uplink a hub to another hub, or to uplink a switch to another switch. This is because the crossover is performed *twice,* once on each hub or switch (MDIX to MDIX), negating the crossover.

Modern devices can now **automatically detect** whether the crossover function is required, negating the need for a crossover cable. This functionality is referred to as **Auto-MDIX**, and is now standard with Gigabit Ethernet, which uses all eight wires to both transmit *and* receive. Auto-MDIX requires that autonegotiation be enabled.

To create a crossover cable, the transmit pins must be swapped with the receive pins on **one** end of the cable:

- Pins 1 and 3
- Pins 2 and 6

Pin#	***Connector 1***		***Connector 2***	***Pin#***
1	White Orange	----------	White Green	3
2	Orange	----------	Green	6
3	White Green	----------	White Orange	1
4	Blue	----------	Blue	4
5	White Blue	----------	White Blue	5
6	Green	----------	Orange	2
7	White Brown	----------	White Brown	7
8	Brown	----------	Brown	8

Note that the Orange and Green pins have been swapped on Connector 2. The first connector is using the TIA/EIA-568B standard, while the second connector is using the TIA/EIA-568A standard.

* * *

Twisted-Pair – Rollover Cable

A **rollover** cable is used to connect a workstation or laptop into a Cisco device's **console** or **auxiliary** port, for management purposes. A rollover cable is often referred to as a *console* cable, and its sheathing is usually flat and light-blue in color.

To create a rollover cable, the pins are completely reversed on one end of the cable:

Pin#	***Connector 1***		***Connector 2***	***Pin#***
1	White Orange	----------	Brown	8
2	Orange	----------	White Brown	7
3	White Green	----------	Green	6
4	Blue	----------	White Blue	5
5	White Blue	----------	Blue	4
6	Green	----------	White Green	3
7	White Brown	----------	Orange	2
8	Brown	----------	White Orange	1

Rollover cables can be used to configure Cisco routers, switches, and firewalls.

* * *

Power over Ethernet (PoE)

Power over Ethernet (PoE) allows both data and power to be sent across the same twisted-pair cable, eliminating the need to provide separate power connections. This is especially useful in areas where installing separate power might be expensive or difficult.

PoE can be used to power many devices, including:

- Voice over IP (VoIP) phones
- Security cameras
- Wireless access points
- Thin clients

PoE was originally formalized as **802.3af**, which can provide roughly 13W of power to a device. **802.3at** further enhanced PoE, supporting 25W or more power to a device.

Ethernet, Fast Ethernet, *and* Gigabit Ethernet all support PoE. Power can be sent across either the *unused* pairs in a cable, or the data transmission pairs, which is referred to as **phantom power.** Gigabit Ethernet requires the phantom power method, as it uses all eight wires in a twisted-pair cable.

The device that *provides* power is referred to as the **Power Source Equipment (PSE).** PoE can be supplied using an **external power injector**, though each powered device requires a separate power injector.

More commonly, an **802.3af-compliant network switch** is used to provide power to many devices simultaneously. The power supplies in the switch must be large enough to support both the switch itself, and the devices it is powering.

(Reference: *http://www.belden.com/docs/upload/PoE_Basics_WP.pdf*)

* * *

Section 2
- Hubs vs. Switches vs. Routers -

Layered Communication

Network communication models are generally organized into **layers.** The **OSI model** specifically consists of **seven layers,** with each layer representing a specific networking function. These functions are controlled by **protocols**, which govern end-to-end communication between devices.

As data is passed from the user application down the virtual layers of the OSI model, each of the lower layers adds a **header** (and sometimes a **trailer**) containing protocol information specific to that layer. These headers are called **Protocol Data Units (PDUs)**, and the process of adding these headers is referred to as **encapsulation**.

The PDU of each lower layer is identified with a unique term:

#	*Layer*	*PDU Name*
7	Application	**-**
6	Presentation	**-**
5	Session	**-**
4	Transport	**Segments**
3	Network	**Packets**
2	Data-link	**Frames**
1	Physical	**Bits**

Commonly, network devices are identified by the OSI layer they *operate* at (or, more specifically, what *header* or *PDU* the device processes).

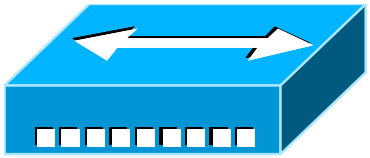
For example, **switches** are generally identified as Layer-2 devices, as switches process information stored in the **Data-Link** header of a frame (such as MAC addresses in Ethernet). Similarly, **routers** are identified as Layer-3 devices, as routers process *logical* addressing information in the **Network** header of a packet (such as IP addresses).

However, the strict definitions of the terms *switch* and *router* have blurred over time, which can result in confusion. For example, the term *switch* can now refer to devices that operate at layers higher than Layer-2. This will be explained in greater detail in this guide.

* * *

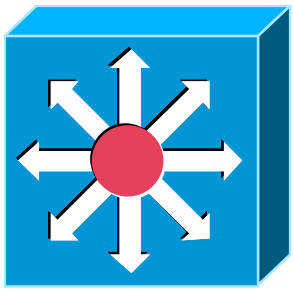
Icons for Network Devices

The following icons will be used to represent network devices for all guides on routeralley.com:

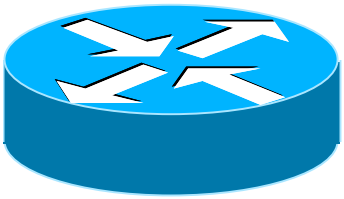


Hub

Switch



Multilayer Switch



Router

Layer-1 Hubs

Hubs are Layer-1 devices that physically connect network devices together for communication. Hubs can also be referred to as **repeaters**.

Hubs provide *no intelligent forwarding* whatsoever. Hubs are incapable of processing either Layer-2 or Layer-3 information, and thus cannot make decisions based on hardware or logical addressing.

Thus, hubs will always forward *every* frame out *every* port, excluding the port originating the frame. Hubs do not differentiate between frame types, and thus will always forward unicasts, multicasts, and broadcasts out *every* port but the originating port.

Ethernet hubs operate at **half-duplex,** which allows a host to either transmit or receive data, but not simultaneously. Half-duplex Ethernet utilizes **Carrier Sense Multiple Access with Collision Detect (CSMA/CD)** to control media access. *Carrier sense* specifies that a host will monitor the physical link, to determine whether a *carrier* (or *signal)* is currently being transmitted. The host will *only* transmit a frame if the link is **idle**.

If two hosts transmit a frame simultaneously, a **collision** will occur. This renders the collided frames unreadable. Once a collision is detected, both hosts will send a **32-bit jam sequence** to ensure all transmitting hosts are aware of the collision. The collided frames are also discarded. Both devices will then wait a *random* amount of time before resending their respective frames, to reduce the likelihood of another collision.

Remember, if *any* two devices connected to a hub send a frame simultaneously, a collision *will* occur. Thus, all ports on a hub belong to the same **collision domain**. A collision domain is simply defined as any physical segment where a collision can occur.

Multiple hubs that are uplinked together still all belong to *one* collision domain. Increasing the number of host devices in a single collision domain will increase the number of collisions, which will degrade performance.

Hubs also belong to only one **broadcast domain –** a hub will forward both broadcasts and multicasts out *every* port but the originating port. A broadcast domain is a logical segmentation of a network, dictating how far a broadcast (or multicast) frame can propagate.

Only a Layer-3 device, such as a router, can separate broadcast domains.

* * *

Layer-2 Switching

Layer-2 devices build **hardware address tables,** which at a minimum contain the following:

- Hardware addresses for hosts
- The port each hardware address is associated with

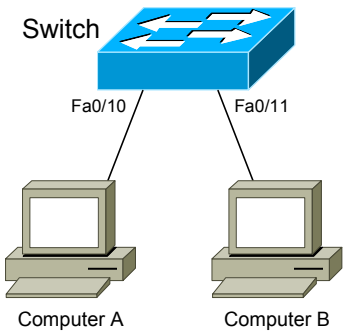
Using this information, Layer-2 devices will make intelligent **forwarding** decisions based on the frame (or data-link) headers. A frame can then be forwarded out *only* the appropriate destination port, instead of *all* ports.

Layer-2 forwarding was originally referred to as **bridging.** Bridging is a largely deprecated term (mostly for marketing purposes), and Layer-2 forwarding is now commonly referred to as **switching.**

There are some subtle technological differences between *bridging* and *switching*. Switches usually have a higher port-density, and can perform forwarding decisions at wire speed, due to specialized hardware circuits called **ASICs (Application-Specific Integrated Circuits).** Otherwise, bridges and switches are nearly identical in function.

Ethernet switches build **MAC address tables** through a dynamic learning process. A switch behaves much like a hub when first powered on. The switch will flood every frame, including unicasts, out *every* port but the originating port.

The switch will then build the MAC-address table by examining the **source MAC address** of each frame. Consider the following diagram:



When ComputerA sends a frame to ComputerB, the switch will add *ComputerA*'s MAC address to its table, associating it with port fa0/10. However, the switch will not learn *ComputerB's* MAC address until ComputerB sends a frame to ComputerA, or to another device connected to the switch. Switches **always learn from the source MAC address in a frame.**

A switch is in a perpetual state of learning. However, as the MAC address table becomes populated, the flooding of frames will decrease, allowing the switch to perform more efficient forwarding decisions.

* * *

Layer-2 Switching (continued)

While hubs were limited to half-duplex communication, switches can operate in **full-duplex.** *Each individual port* on a switch belongs to its *own collision domain*. Thus, switches create **more collision domains**, which results in **fewer collisions.**

Like hubs though, switches belong to only *one* **broadcast domain.** A Layer-2 switch will forward both broadcasts and multicasts out *every* port but the originating port. Only Layer-3 devices separate broadcast domains.

Because of this, Layer-2 switches are poorly suited for large, scalable networks. The Layer-2 header provides no mechanism to differentiate one *network* from another, only one *host* from another.

This poses *significant* difficulties. If *only* hardware addressing existed, all devices would technically be on the *same* network. Modern internetworks like the Internet could not exist, as it would be impossible to separate *my* network from *your* network.

Imagine if the entire Internet existed purely as a Layer-2 switched environment. Switches, as a rule, will forward a broadcast out *every* port. Even with a conservative estimate of a billion devices on the Internet, the resulting broadcast storms would be devastating. The Internet would simply collapse.

Both hubs and switches are susceptible to **switching loops**, which result in destructive broadcast storms. Switches utilize the **Spanning Tree Protocol (STP)** to maintain a loop-free environment. STP is covered in great detail in another guide.

Remember, there are three things that switches do that hubs do not:

- **Hardware address learning**
- **Intelligent forwarding of frames**
- **Loop avoidance**

Hubs are almost entirely deprecated – there is no advantage to using a hub over a switch. At one time, switches were more expensive and introduced more latency (due to processing overhead) than hubs, but this is no longer the case.

* * *

Layer-2 Forwarding Methods

Switches support three **methods** of forwarding frames. Each method copies all or part of the frame into memory, providing different levels of latency and reliability. **Latency** is *delay* - less latency results in quicker forwarding.

The **Store-and-Forward** method copies the *entire* frame into memory, and performs a Cycle Redundancy Check (CRC) to completely ensure the integrity of the frame. However, this level of error-checking introduces the highest latency of any of the switching methods.

The **Cut-Through (Real Time)** method copies only enough of a frame's header to determine its destination address. This is generally the *first 6 bytes* following the preamble. This method allows frames to be transferred at *wire speed*, and has the least latency of any of the three methods. No error checking is attempted when using the cut-through method.

The **Fragment-Free (Modified Cut-Through)** method copies only the *first 64 bytes* of a frame for error-checking purposes. Most collisions or corruption occur in the first 64 bytes of a frame. Fragment-Free represents a compromise between reliability (store-and-forward) and speed (cut-through).

Layer-3 Routing

Layer-3 **routing** is the process of forwarding a packet from one *network* to another *network*, based on the Network-layer header. Routers build **routing tables** to perform forwarding decisions, which contain the following:

- The destination network and subnet mask
- The **next hop** router to get to the destination network
- Routing *metrics* and Administrative Distance

Note that Layer-3 forwarding is based on the destination *network*, and not the destination *host*. It is possible to have *host routes*, but this is less common.

The routing table is concerned with two types of Layer-3 protocols:

- **Routed protocols** - assigns logical addressing to devices, and routes packets between networks. Examples include IP and IPX.
- **Routing protocols** - dynamically builds the information in routing tables. Examples include RIP, EIGRP, and OSPF.

Each individual interface on a router belongs to its *own collision domain*. Thus, like switches, routers create **more collision domains**, which results in **fewer collisions.**

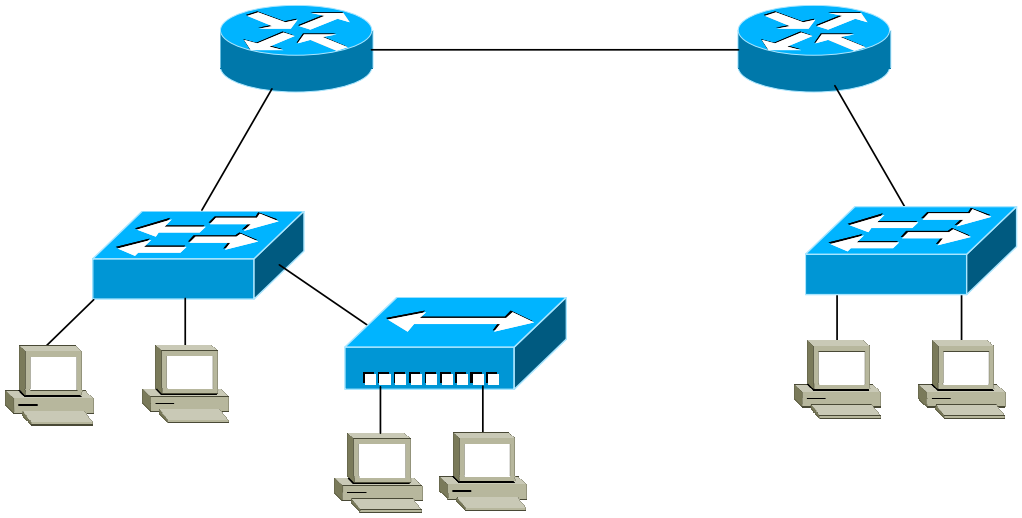
Unlike Layer-2 switches, Layer-3 routers also **separate broadcast domains**. As a rule, a router **will never forward broadcasts** from one network to another network (unless, of course, you explicitly configure it to). ☺

Routers will not forward multicasts either, unless configured to participate in a multicast tree. Multicast is covered in great detail in another guide.

Traditionally, a router was required to copy each individual packet to its buffers, and perform a route-table lookup. Each packet consumed CPU cycles as it was forwarded by the router, resulting in latency. Thus, routing was generally considered **slower** than switching.

It is now possible for routers to *cache* network-layer flows in hardware, greatly reducing latency. This has blurred the line between *routing* and *switching,* from both a technological and marketing standpoint. Caching network flows is covered in greater detail shortly.

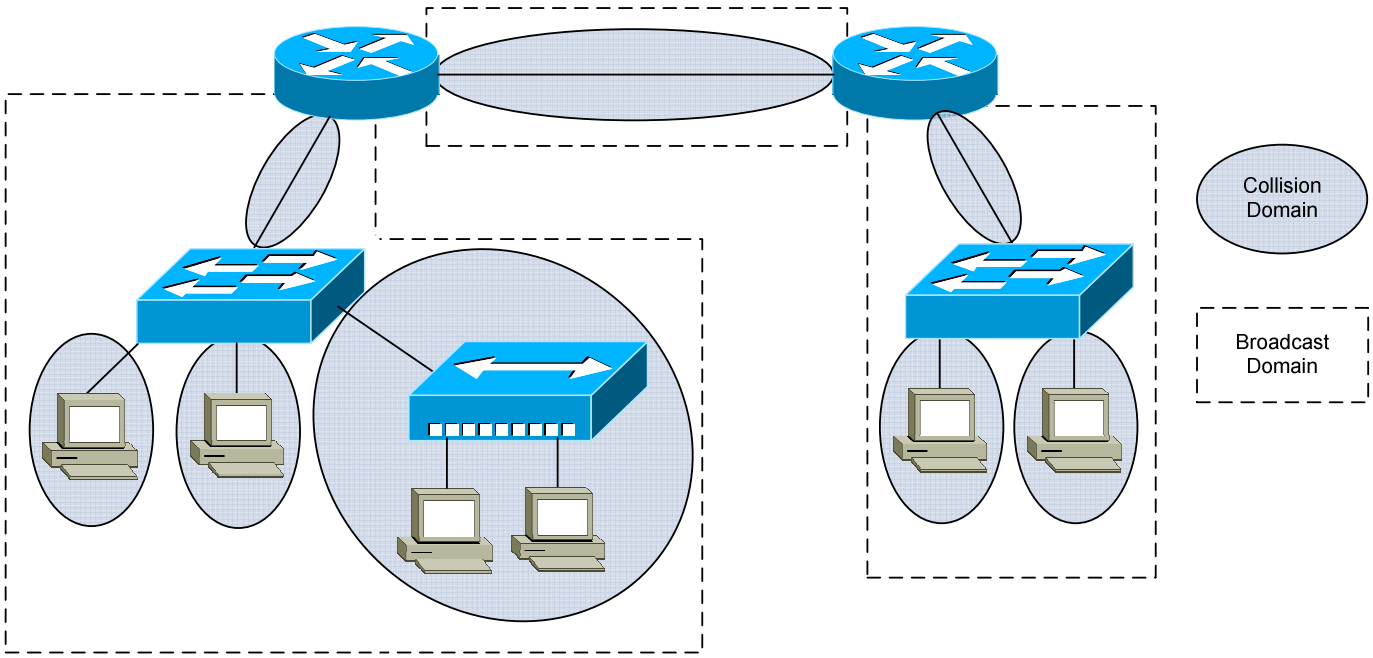
Collision vs. Broadcast Domain Example



Consider the above diagram. Remember that:

- Routers separate *broadcast* and *collision* domains.
- Switches separate *collision* domains.
- Hubs belong to only one *collision* domain.
- Switches and hubs both only belong to one *broadcast* domain.

In the above example, there are **THREE** broadcast domains, and **EIGHT** collision domains:



* * *

VLANs – A Layer-2 or Layer-3 Function?

By default, a switch will forward both broadcasts and multicasts out *every* port but the originating port.

However, a switch can be logically segmented into multiple broadcast domains**,** using **Virtual LANs** (or **VLANs**). VLANs are covered in extensive detail in another guide.

Each VLAN represents a unique broadcast domain:

- Traffic between devices within the *same* VLAN is switched (forwarded at Layer-2).
- Traffic between devices in *different* VLANs requires a Layer-3 device to communicate.

Broadcasts from one VLAN will not be forwarded to another VLAN. The logical separation provided by VLANs is **not a Layer-3 function.** VLAN tags are inserted into the **Layer-2 header**.

Thus, a switch that supports VLANs is not necessarily a Layer-3 switch. However, a purely Layer-2 switch cannot route between VLANs.

Remember, though VLANs provide separation for *Layer-3* broadcast domains, they are still a *Layer-2* function. A VLAN often has a one-to-one relationship with an IP subnet, though this is not a requirement.

* * *

Layer-3 Switching

In addition to performing Layer-2 switching functions, a **Layer-3 switch** must also meet the following criteria:

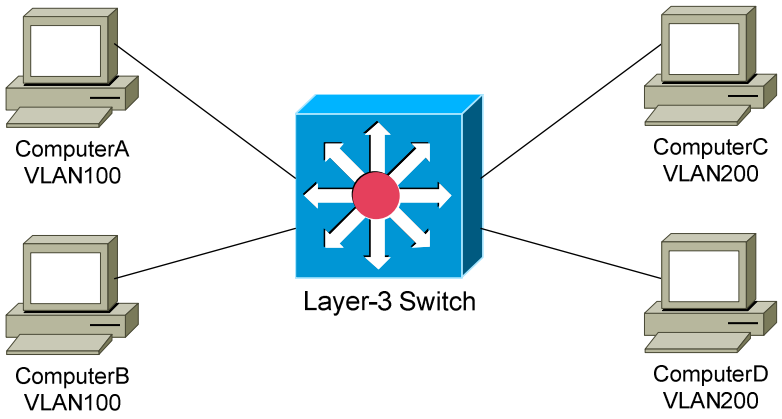
- The switch must be capable of making Layer-3 forwarding decisions (traditionally referred to as routing).
- The switch must cache network traffic flows, so that Layer-3 forwarding can occur in hardware.

Many older modular switches support Layer-3 route processors – this alone does not qualify as Layer-3 switching. Layer-2 and Layer-3 processors can act independently within a single switch chassis, with each packet requiring a route-table lookup on the route processor.

Layer-3 switches leverage ASICs to perform Layer-3 forwarding in hardware. For the first packet of a particular traffic flow, the Layer-3 switch will perform a standard route-table lookup. This flow is then *cached* in hardware – which preserves required routing information, such as the destination network and the MAC address of the corresponding next-hop.

Subsequent packets of that flow will bypass the route-table lookup, and will be forwarded based on the cached information, reducing latency. This concept is known as *route once, switch many.*

Layer-3 switches are predominantly used to route between VLANs:



Traffic between devices within the same VLAN, such as ComputerA and ComputerB, is *switched* at Layer-2 as normal. The first packet between devices in different VLANs, such as ComputerA and ComputerD, is *routed*. The switch will then cache that IP traffic flow, and subsequent packets in that flow will be *switched* in hardware.

* * *

Layer-3 Switching vs. Routing – End the Confusion!

The evolution of network technologies has led to considerable confusion over the terms *switch* and *router.* Remember the following:

- The traditional definition of a *switch* is a device that performs Layer-2 forwarding decisions.
- The traditional definition of a *router* is a device that performs Layer-3 forwarding decisions.

Remember also that, switching functions were typically performed in *hardware,* and routing functions were typically performed in *software.* This resulted in a widespread perception that switching was *fast,* and routing was *slow* (and *expensive*).

Once Layer-3 forwarding became available in hardware, marketing gurus muddied the waters by distancing themselves from the term *router*. Though Layer-3 forwarding in hardware is still *routing* in every technical sense, such devices were rebranded as Layer-3 switches.

Ignore the marketing noise. **A Layer-3 switch** is **still a router.**

Compounding matters further, most devices still currently referred to as *routers* can perform Layer-3 forwarding in hardware as well. Thus, both Layer-3 switches *and* Layer-3 routers perform nearly identical functions at the same performance.

There are some differences in *implementation* between Layer-3 switches and routers, including (but not limited to):

- Layer-3 switches are optimized for Ethernet, and are predominantly used for inter-VLAN routing. Layer-3 switches can also provide Layer-2 functionality for intra-VLAN traffic.
- Switches generally have higher port densities than routers, and are considerably cheaper per port than routers (for Ethernet, at least).
- Routers support a large number of WAN technologies, while Layer-3 switches generally do not.
- Routers generally support more advanced feature sets.

Layer-3 switches are often deployed as the backbone of LAN or campus networks. Routers are predominantly used on network perimeters, connecting to WAN environments.

(Fantastic Reference: http://blog.ioshints.info/2011/02/how-did-we-ever-get-into-this-switching.html)

* * *

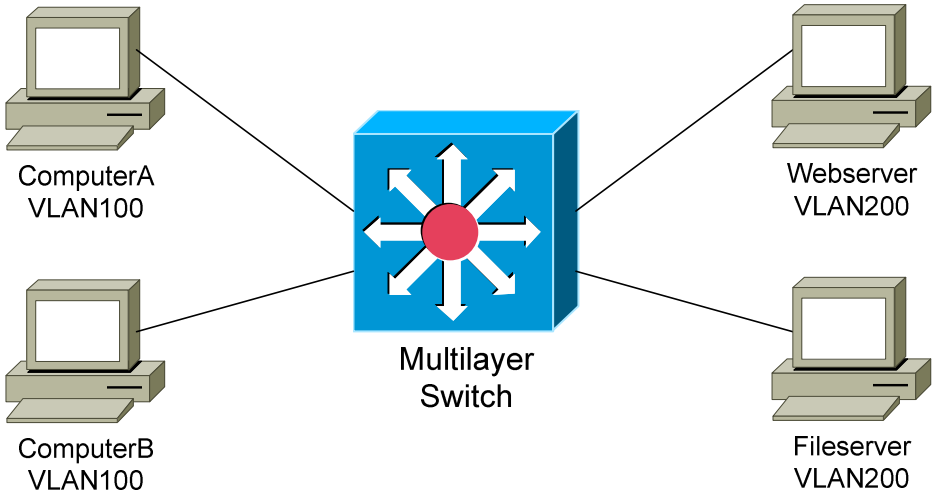
Multilayer Switching

Multilayer switching is a generic term, referring to any switch that forwards traffic at layers higher than Layer-2. Thus, a Layer-3 switch is considered a multilayer switch, as it forwards frames at Layer-2 and packets at Layer-3.

A **Layer-4 switch** provides the same functionality as a Layer-3 switch, but will additionally examine and cache **Transport-layer application flow** information, such as the TCP or UDP port.

By caching application flows, **QoS (Quality of Service)** functions can be applied to preferred applications.

Consider the following example:



Network and application traffic flows from ComputerA to the Webserver and Fileserver will be cached. If the traffic to the Webserver is preferred, then a higher QoS priority can be assigned to that application flow.

Some advanced multilayer switches can provide load balancing, content management, and other application-level services. These switches are sometimes referred to as **Layer-7 switches.**

Section 3
- Switching Architectures -

Network Traffic Models

Traffic flow is an important consideration when designing scalable, efficient networks. Fundamentally, this involves understanding two things:

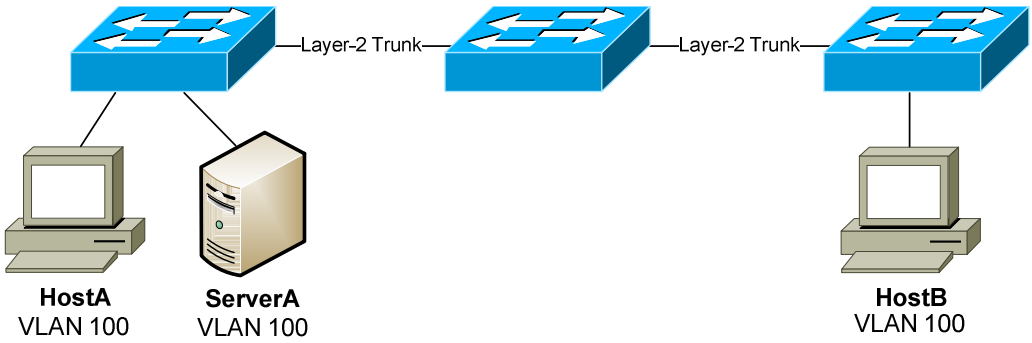
- Where do *resources* reside?
- Where do the *users* reside that access those resources?

Legacy networks adhered to the **80/20 design**, which dictated that**:**

- 80 percent of traffic should remain on the local network.
- 20 percent of traffic should be routed to a remote network.

To accommodate this design practice, resources were placed as close as possible to the users that required them. This allowed the majority of traffic to be *switched*, instead of *routed*, which reduced latency in legacy networks.

The 80/20 design allowed VLANs to be trunked across the entire campus network, a concept known as **end-to-end VLANs**:



End-to-end VLANs allow a host to exist *anywhere* on the campus network, while maintaining Layer-2 connectivity to its resources.

However, this *flat* design poses numerous challenges for scalability and performance:

- STP domains are very large, which may result in instability or convergence issues.
- Broadcasts proliferate throughout the entire campus network.
- Maintaining end-to-end VLANs adds administrative overhead.
- Troubleshooting issues can be difficult.

* * *

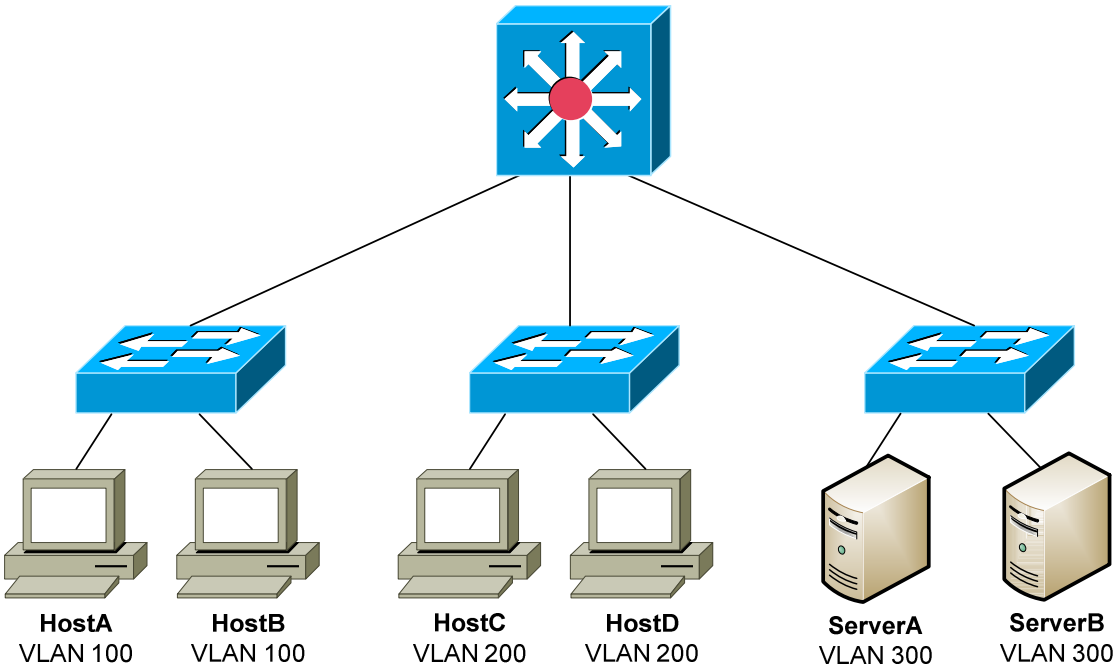
Network Traffic Models (continued)

As network technology improved, centralization of resources became the dominant trend. Modern networks adhere to the **20/80 design**:

- 20 percent of traffic should remain on the local network.
- 80 percent of traffic should be routed to a remote network.

Instead of placing *workgroup* resources in every local network, most organizations centralize resources into a datacenter environment. Layer-3 switching allows users to access these resources with minimal latency.

The 20/80 design encourages a **local VLAN** approach. VLANs should stay **localized** to a single switch or switch block:



This design provides several benefits:

- STP domains are limited, reducing the risk of convergence issues.
- Broadcast traffic is isolated within smaller broadcast domains.
- Simpler, *hierarchical* design improves scalability and performance.
- Troubleshooting issues is typically easier.

There are nearly no drawbacks to this design, outside of a legacy application requiring Layer-2 connectivity between users and resources. In that scenario, it's time to invest in a better application. ☺

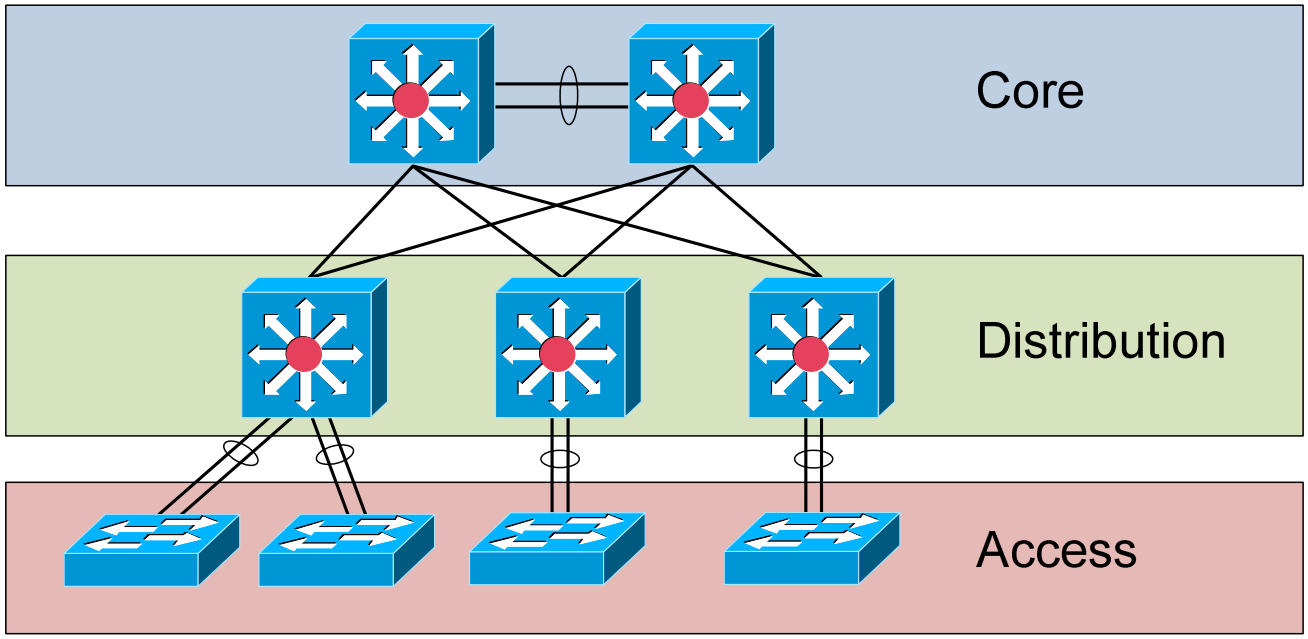
* * *

The Cisco Hierarchical Network Model

To aid in designing scalable networks, Cisco developed a **hierarchical network model,** which consists of three layers:

- **Access** layer
- **Distribution** layer
- **Core** layer

Cisco Hierarchical Model – Access Layer



The **access layer** is where users and hosts connect into the network. Switches at the access layer typically have the following characteristics:

- High port density
- Low cost per port
- Scalable, redundant uplinks to higher layers
- Host-level functions such as VLANs, traffic filtering, and QoS

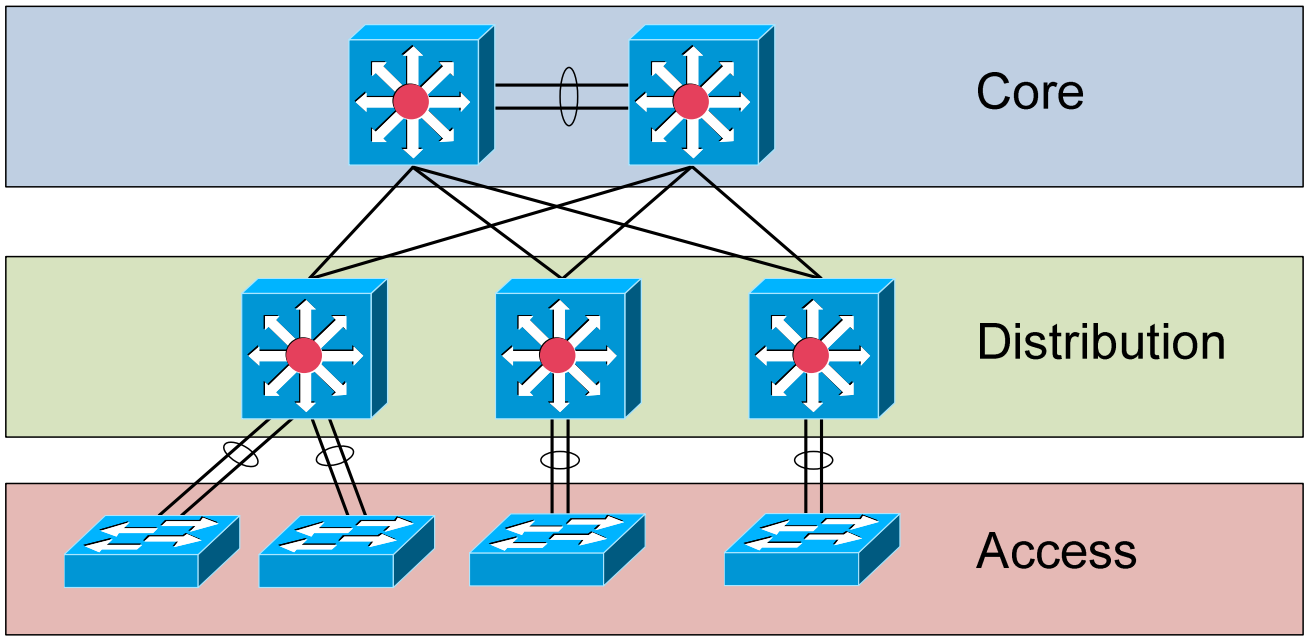
In an *80/20* design, resources are placed as close as possible to the users that require them. Thus, most traffic will never need to leave the access layer.

In a *20/80* design, traffic must be forwarded through higher layers to reach centralized resources.

(Reference: CCNP Switch 642-813 Official Certification Guide by David Hucaby. Cisco Press)

* * *

Cisco Hierarchical Model – Distribution Layer



The **distribution layer** is responsible for aggregating access layer switches, and connecting the access layer to the core layer. Switches at the distribution layer typically have the following characteristics:

- Layer-3 or multilayer forwarding
- Traffic filtering and QoS
- Scalable, redundant links to the core and access layers

Historically, the distribution layer was the Layer-3 boundary in a hierarchical network design:

- The connection between access and distribution layers was Layer-2.
- The distribution switches are configured with VLAN SVIs.
- Hosts in the access layer use the SVIs as their default gateway.

This remains a common design today.

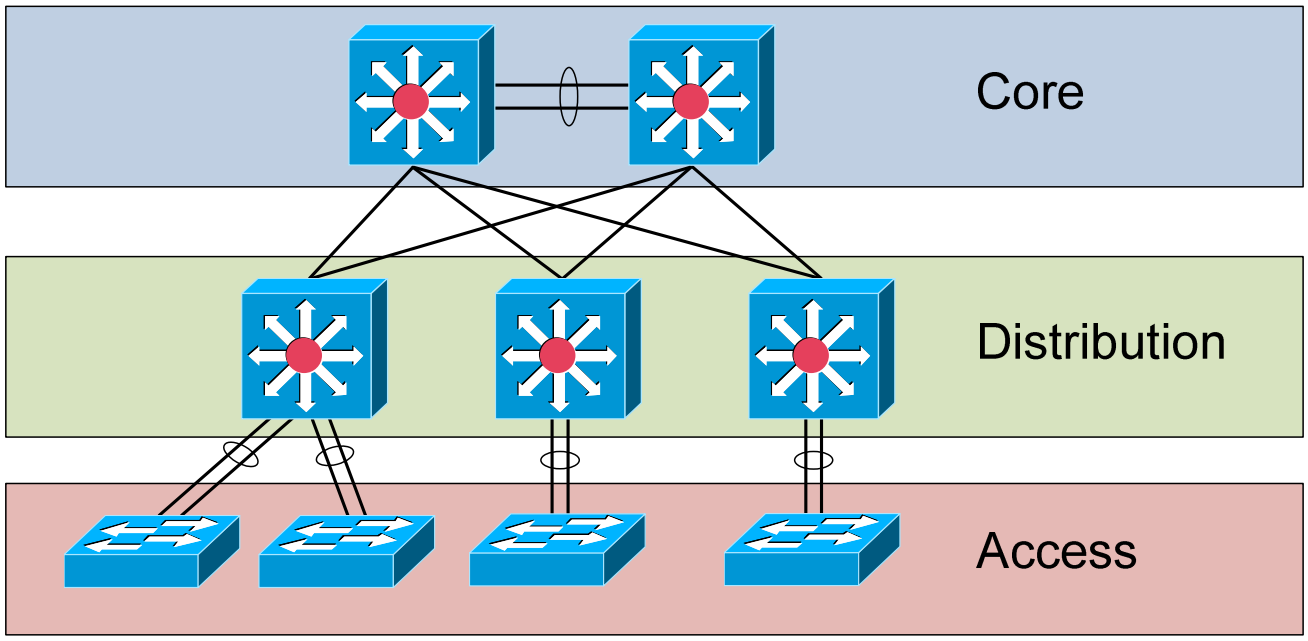
However, **pushing Layer-3 to the access-layer** has become increasingly prevalent. VLAN SVIs are configured on the *access layer* switch, which hosts will use as their default gateway.

A routed connection is then used between access and distribution layers, further minimizing STP convergence issues and limiting broadcast traffic.

(Reference: CCNP Switch 642-813 Official Certification Guide by David Hucaby. Cisco Press)

* * *

Cisco Hierarchical Model – Core Layer



The **core** layer is responsible for connecting all distribution layer switches. The core is often referred to as the **network backbone**, as it forwards traffic from to every end of the network.

Switches at the core layer typically have the following characteristics:

- High-throughput Layer-3 or multilayer forwarding
- Absence of traffic filtering, to limit latency
- Scalable, redundant links to the distribution layer and other core switches
- Advanced QoS functions

Proper core layer design is focused on **speed and efficiency**. In a 20/80 design, most traffic will traverse the core layer. Thus, core switches are often the highest-capacity switches in the campus environment.

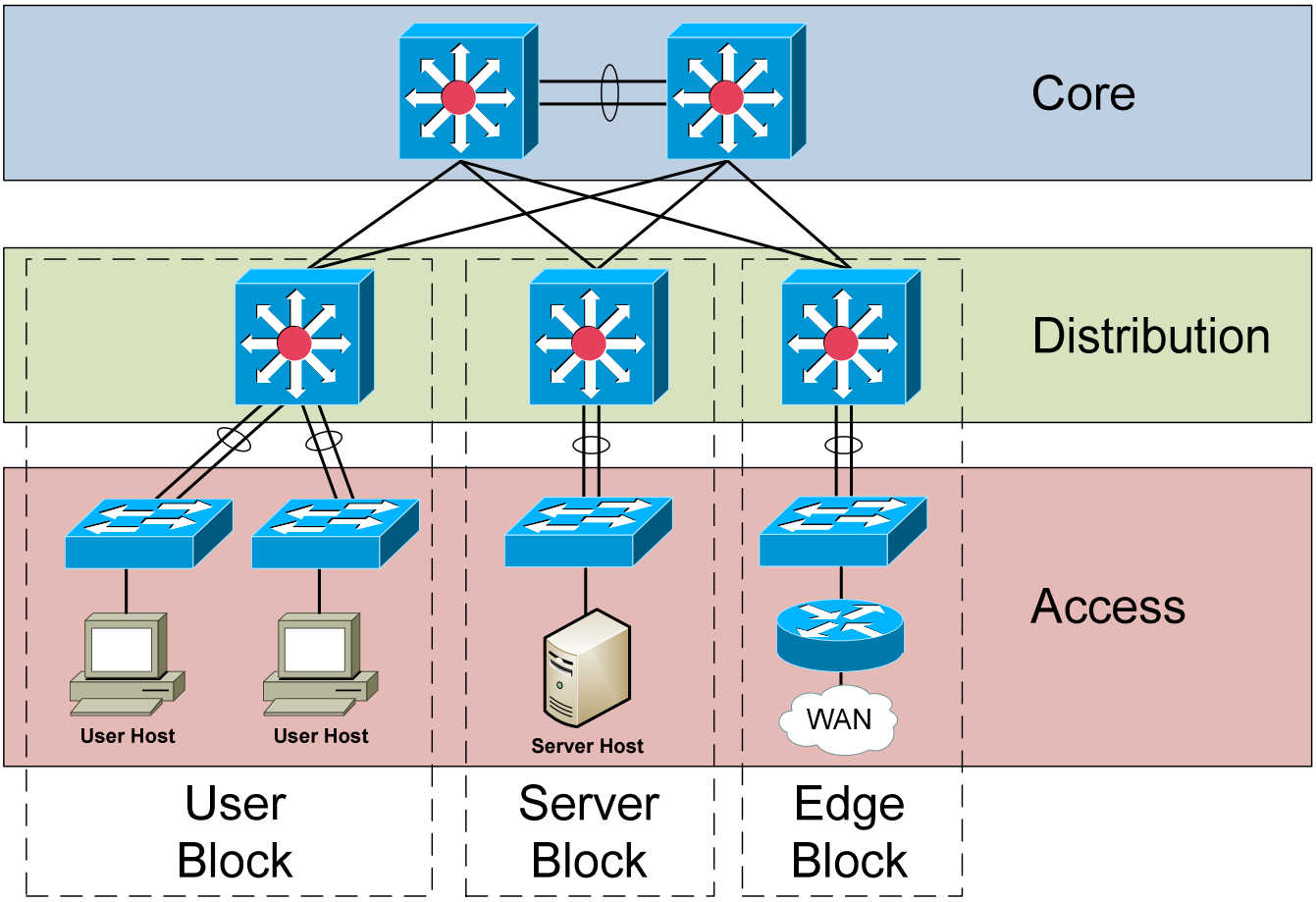
Smaller campus environments may not require a clearly defined core layer separated from the distribution layer. Often, the functions of the core and distribution layers are combined into a single layer. This is referred to as a **collapsed core** design.

(Reference: CCNP Switch 642-813 Official Certification Guide by David Hucaby. Cisco Press)

* * *

Cisco Hierarchical Model – Practical Application

A hierarchical approach to network design enforces scalability and manageability. Within this framework, the network can be compartmentalized into modular **blocks**, based on function.



The above example illustrates common block types:

- **User block** – containing end users
- **Server block** – containing the resources accessed by users
- **Edge block** – containing the routers and firewalls that connect users to the WAN or Internet

Each block connects to each other through the core layer, which is often referred to as the **core block.** Connections from one layer to another should always be redundant.

A large campus environment may contain *multiple* user, server, or edge blocks. Limiting bottlenecks and broadcasts are key considerations when determining the size of a block.

* * *

Section 4
- Switching Tables -

Layer-2 Forwarding Overview

Layer-2 devices build **hardware address tables,** which at a minimum contain the following:

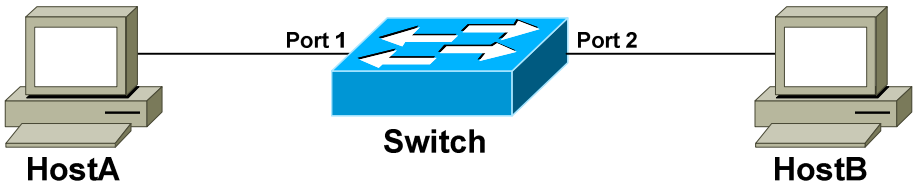
- Hardware addresses for hosts (such as Ethernet MAC addresses)
- The port each hardware address is associated with

Using this information, Layer-2 devices will make intelligent **forwarding** decisions based on the frame (or data-link) header. A frame can then be forwarded out *only* the appropriate destination port, instead of *all* ports.

Layer-2 forwarding was originally referred to as **bridging.** Bridging is a largely deprecated term (mostly for marketing purposes), and Layer-2 forwarding is now commonly referred to as **switching.**

Switching Queues

Layer-2 switches utilize **queues** to store incoming and outgoing frames. Consider the following diagram:



1. The switch receives a frame on Port 1, from HostA destined for HostB.
2. The frame is placed in Port 1's **ingress queue.**
3. The switch performs a lookup on the destination hardware address - HostB in this example.
4. The switch determines that the appropriate destination port for HostB is Port 2.
5. The frame is placed in Port 2's **egress queue.**

If the switch had no knowledge of HostB's hardware address, the frame would be placed in the egress queue **of all ports** except for the originating port**,** and thus flooded to the entire network.

A port can contain multiple ingress or egress queues. This allows critical traffic to be prioritized over less important traffic.

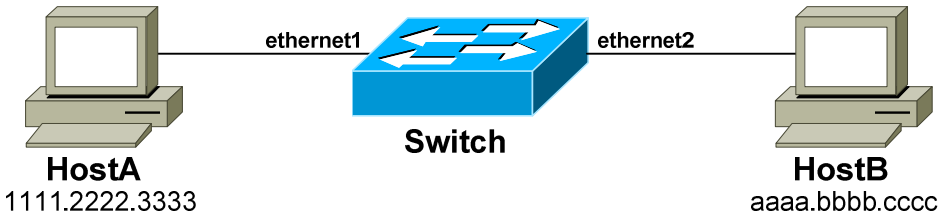
* * *

MAC Address Table

In the previous example, the switch made a **forwarding decision** based on the destination host's hardware address. The switch essentially forwarded the frame from one port's *ingress* queue to another port's *egress* queue.

To perform this forwarding decision, a switch consults its hardware address table. For Ethernet switches, this is referred to as the **MAC address table**, or the Layer-2 forwarding table.

When a switch is first powered on, the MAC address table will be empty. The switch will build the table through a dynamic learning process, by observing the **source MAC address** of frames:



1. Initially, the switch will have no knowledge of the MAC addresses of HostA and HostB.
2. When HostA sends a frame to HostB, the switch will add *HostA's* MAC address to its table, associating it with port e*thernet1*.
3. The switch will not learn HostB's MAC address until HostB sends a frame back to HostA, or to any other host connected to the switch.
4. HostB's MAC address will then be associated with port *ethernet2*.

Remember: a switch will only add MAC address table entries based on the **source** MAC address in a frame.

The MAC address table is stored in fast volatile memory**,** allowing lookups to be performed very quickly. However, this also results in dynamically-learned MAC addresses being lost if the switch is rebooted or powered off.

Stale (or *idle*) entries in the table will be **aged out.** By default on Cisco switches, idle entries will be purged after 300 seconds.

Most switches support **statically** configuring MAC addresses into the table, which will survive a reboot or power failure, and never be purged. Statically configuring entries in the table is only required in limited circumstances.

* * *

CAM and TCAM Tables

On Cisco switches, the MAC address table is stored in **Content Addressable Memory (CAM).**

CAM differs from the more prevalent Random Access Memory (RAM):

- RAM queries a specific *memory address***,** and then returns the *data* or *content* stored at that address location.
- CAM operates essentially in the reverse, and does not require that a memory address be provided. Instead, CAM queries for the desired *content,* and then returns all matching results, including any *associated* content.

CAM is *significantly* faster than RAM, as it searches the entire memory content in one cycle, instead of a single address at a time. However, CAM is more expensive than RAM.

When performing a MAC address table lookup, the MAC address itself is the *content* being queried. For any matching results, CAM will return the destination port (the *associated content).*

Cisco uses the terms *MAC address table* and *CAM table* interchangeably. This guide will use the term **CAM table** moving forward.

Idle entries in the CAM are purged after **300 seconds**, by default. This timer is reset every time a frame is received with the associated MAC address on the correct port.

If a host moves to a different port on a switch, the CAM table entry for the previous port will be **purged immediately**. This is desirable behavior - a MAC address is unique, and should never exist on more than one switch port unless a switching loop or other issue exists.

Ternary Content Addressable Memory (TCAM) tables provide high-speed lookups for two additional functions:

- Filtering traffic using access-lists
- Prioritizing traffic using QoS

TCAM tables are covered in greater detail later in this guide.

Multilayer switches utilize the **Forwarding Information Base (FIB)** table for L3 forwarding decisions. Multilayer switching is covered extensively in a different guide.

* * *

Managing the CAM Table

Each entry in the CAM table contains the following information:

- The MAC address
- The switch port the MAC address was learned on
- The VLAN of the switch port
- A time stamp, for the aging timer

To view the entire CAM table:

Switch# *show mac address-table*

```
 vlan    mac address     type      port
-------+---------------+---------+--------------------
   9    000c.291e.96f0   dynamic  GigabitEthernet1/1
   9    000c.293c.7cac   dynamic  GigabitEthernet1/1
   9    000c.2950.e3e9   dynamic  GigabitEthernet1/1
   9    000c.29ba.fe28   dynamic  GigabitEthernet1/2
   9    842b.2ba6.3a7d   dynamic  GigabitEthernet1/3
   9    d067.e50b.1975   dynamic  GigabitEthernet1/5
   9    d067.e51e.e35a   dynamic  GigabitEthernet2/1
   9    f04d.a2f6.d37b   dynamic  GigabitEthernet2/2
```

A single switch port can learn many addresses. In the above output, *GigabitEthernet1/1* has multiple MAC addresses associated with it. This usually indicates this is an uplink to another switch.

To view the CAM table entries for a specific port or MAC address:

Switch# *show mac address-table interface GigabitEthernet 1/5*

```
 vlan    mac address     type      port
-------+---------------+---------+--------------------
   9    d067.e50b.1975   dynamic  GigabitEthernet1/5
```

Switch# *show mac address-table address f04d.a2f6.d37b*

```
 vlan    mac address     type      port
-------+---------------+---------+--------------------
   9    f04d.a2f6.d37b   dynamic  GigabitEthernet2/2
```

The output of a command can be filtered using the **pipe** command. For example, to search for any entry that contains *3a7d* in the MAC address:

Switch# *show mac address-table | include 3a7d*

```
 vlan    mac address     type      port
-------+---------------+---------+--------------------
   9    842b.2ba6.3a7d   dynamic  GigabitEthernet1/3
```

* * *

Managing the CAM Table (continued)

To specifically display only *dynamic* or *static* CAM entries:

Switch# *show mac address-table dynamic*
Switch# *show mac address-table static*

To view the total number of entries in the CAM table:

Switch# *show mac address-table count*

```
MAC Entries for all vlans:
Dynamic Unicast Address Count:                  234
Static Unicast Address (User-defined) Count:    0
Static Unicast Address (System-defined) Count:  6
Total Unicast MAC Addresses In Use:             240
Total Unicast MAC Addresses Available:          55000
Multicast MAC Address Count:                    9
Total Multicast MAC Addresses Available:        32768
```

The CAM aging timer can be changed from its **default of 300**, though this is needed only in rare circumstances:

Switch# *config t*
Switch(config)# *mac address-table aging-time 360*

To add a static entry into the CAM table:

Switch(config)# *mac address-table static 0011.2233.4455 vlan 9 interface GigabitEthernet 2/7*

To clear all dynamic entries in the CAM table:

Switch# *clear mac address-table dynamic all*

To clear a single entry in the CAM, either by MAC address or interface:

Switch# *clear mac address-table dynamic address d067.e51e.e35a*
Switch# *clear mac address-table dynamic interface GigabitEthernet 2/1*

Note: In Cisco IOS versions prior to 12.1, the syntax for all CAM table commands contained an additional hyphen between *mac* and *address:*

Switch# *show mac-address-table*

This additional hyphen is **no longer required** on modern versions of the IOS. Some IOS versions may support both syntaxes.

* * *

Ternary Content Addressable Memory (TCAM)

Recall that switches utilize **TCAM tables** for two purposes:

- Filtering traffic using access-lists
- Prioritizing traffic using QoS

Some Layer-3 devices store the routing table in TCAM as well. Most Layer-3 switches support multiple TCAM tables, to separately manage the access-lists for inbound and outbound traffic, and for QoS.

The TCAM consists of two components:

- **Feature Manager (FM) –** automatically integrates access-lists into the TCAM.
- **Switching Database Manager (SDM) –** supports partitioning the TCAM for separate functions (supported on only some Cisco models).

Each entry in the TCAM table contains three components, defined by access-list entries:

- **Values –** defines the addresses or ports that must be matched
- **Masks –** defines how much of each address to match
- **Result –** defines the action to take when a match occurs

Consider the following access-list:

Switch(config)# *access-list WEB permit tcp 10.1.1.0 0.0.0.255 host 10.2.1.1 eq 443*
Switch(config)# *access-list WEB deny tcp 10.1.0.0 0.0.0.255 host 10.2.1.1 eq 80*

- The **values** are the source (*10.1.1.0*) and destination (*10.2.1.1)* addresses, and the TCP ports (*443* and *80*, respectively).
- The **masks** are *0.0.0.255* for the source, and *0.0.0.0* for the destination. This indicates that the first three octets must match for the source, and the destination much match exactly.
- The **results** are *permit* for the first entry, and *deny* for the second. Other results are possible - such as when using QoS which is more concerned with *prioritizing* traffic than *filtering* it.

The Feature Manager (FM) will automatically integrate the access-list named *WEB* into the TCAM. Configuring the TCAM consists *solely* of creating the necessary access-lists. However, the access-list will not take effect until it's applied to an interface or VLAN.

* * *

Part II

Switch Configuration

Section 5
- The Cisco IOS -

Cisco IOS

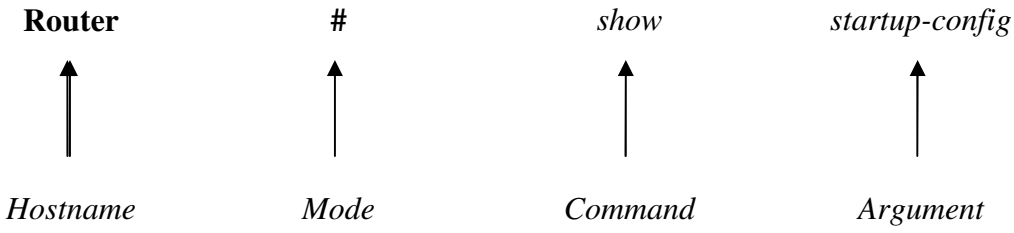
The **Cisco IOS (Internetwork Operating System)** is a command-line interface used by nearly all current Cisco routers and Catalyst switches. The IOS provides the mechanism to configure all Layer 2 and Layer 3 functions on Cisco devices.

The IOS is structured into several **modes,** which contain sets of commands specific to the function of that mode. Access to a specific mode (and specific commands) is governed by **privilege levels**. (Both modes and privilege levels are covered in great detail in this guide).

The following is a representation of the IOS command-line interface, with an example command:

Router# *show startup-config*

All commands throughout all guides on this site will be represented like the above. The following is an explanation of each component of the above command:



Hitting the "enter" key after a command will usually yield output specific to your command:

```
Router# show startup-config
!
version 12.2
service timestamps log uptime
service password-encryption
!
hostname Router
!

<snip>
```

(Note: The above output was truncated to save space.)

* * *

IOS Version Numbers

IOS version numbers are formatted as follows:

x.y(z)t

- The "x" designates a **major revision number.**
- The "y" designates a **minor revision number.**
- The "z" designates an **individual release number**
- The "t" designates a **train identifier.**

Thus, the third release of IOS version 12.4 would be identified as 12.4(3). The major and minor revision numbers combined is often called the **Maintenance Release** number (for example, "12.4").

Trains identify IOS releases to specific markets, and are represented by a single letter:

- The "T" or **Technology** train is continuously updated with new features and security fixes.
- The "E" or **Enterprise** train contains features and commands for enterprise-level equipment.
- The "S" or **Service Provider** train contains features and a command-set for specific ISP equipment

The absence of a train identifier denotes a **Mainline** release. Security updates are released for the mainline train, but new functionality is never added to the feature set.

The latest version of the IOS (as of this writing) is 12.4(11)T. To view the IOS version of your Cisco device:

Router# *show version*

The Cisco IOS is stored in **Flash** on Cisco routers and Catalyst switches, in a **.BIN** file format. It can be upgraded using one of several methods:

- Replacing the existing Flash stick
- Via a TFTP server
- Via Xmodem
- Via a PCMCIA slot (not supported by all Cisco devices)

(Reference: *http://en.wikipedia.org/wiki/Cisco_IOS*)

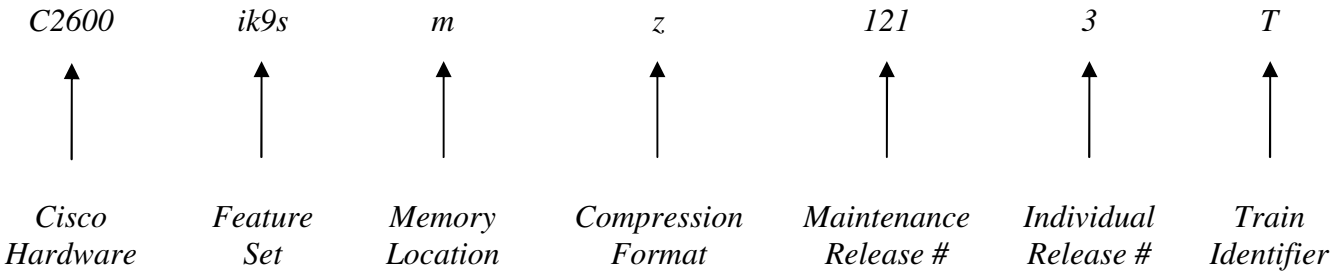
* * *

IOS Version Numbers (continued)

The IOS .bin file stored in flash follows a specific naming convention. Observe the following IOS image:

c2600-ik9s-mz.121-3.T.bin

The following is an explanation of each component of the above file name:



(Reference: *http://www.cisco.com/en/US/products/sw/iosswrel/ps1828/products_white_paper09186a008018305e.shtml*)

The IOS supports a wide variety of feature sets. The following is a list of common feature sets (and is by no means comprehensive):

- **is**
- **ipbase**
- **ipvoice**
- **advsecurityk9**
- **advipservicesk9**
- **ik9s**
- **jk9s**
- **io3**
- **bin**

(Reference: ***http://www.cisco.com/en/US/products/hw/routers/ps259/prod_bulletin09186a0080161082.html***)

* * *

IOS Modes on Cisco Devices

As stated earlier in this guide, the Cisco IOS is comprised of several modes, each of which contains a set of commands specific to the function of that mode.

By default, the first mode you enter when logging into a Cisco device is **User EXEC mode.** User mode appends a ">" after the device hostname:

Router>

No configuration can be changed or viewed from User mode. Only basic status information can be viewed from this mode.

Privileged EXEC mode allows all configuration files, settings, and status information to be viewed. Privileged mode appends a "#" after the device hostname:

Router#

To enter Privileged mode, type *enable* from User mode:

Router> *enable*
Router#

To return back to User mode from Privileged mode, type *disable*:

Router# *disable*
Router>

Very little configuration can be *changed* directly from Privileged mode. Instead, to actually configure the Cisco device, one must enter **Global Configuration mode:**

Router(config)#

To enter Global Configuration mode, type *configure terminal* from Privileged Mode:

Router# *configure terminal*
Router(config)#

To return back to Privileged mode, type *exit*:

Router(config)# *exit*
Router#

* * *

IOS Modes on Cisco Devices (continued)

As its name implies, Global Configuration mode allows parameters that *globally* affect the device to be changed. Additionally, Global Configuration mode is sectioned into several sub-modes dedicated for specific functions. Among the most common sub-modes are the following:

- **Interface Configuration mode - Router(config-if)#**
- **Line Configuration mode - Router(config-line)#**
- **Router Configuration mode - Router(config-router)#**

Recall the difference between *interfaces* and *lines*. **Interfaces** connect routers and switches to each other. In other words, traffic is actually routed or switched across interfaces. Examples of interfaces include Serial, ATM, Ethernet, Fast Ethernet, and Token Ring.

To configure an interface, one must specify both the *type* of interface, and the interface *number* (which always begins at "0"). Thus, to configure the first Ethernet interface on a router:

Router(config)# *interface ethernet 0*
Router(config-if)#

Lines identify ports that allow us to connect into, and then configure, Cisco devices. Examples would include console ports, auxiliary ports, and VTY (or telnet) ports.

Just like interfaces, to configure a line, one must specify both the *type* of line, and the line *number* (again, always begins at "0"). Thus, to configure the first console line on a router:

Router(config)# *line console 0*
Router(config-line)#

Multiple telnet lines can be configured simultaneously. To configure the first five telnet (or VTY) lines on a router:

Router(config)# *line vty 0 4*
Router(config-line)#

Remember that the numbering for both interfaces and lines begins with "0."

Router Configuration mode is used to configure dynamic routing protocols, such as RIP. This mode is covered in great detail in other guides.

* * *

IOS Command Shortcuts

Shortcuts are allowed on the IOS command-line, as long as the truncated command is not ambiguous. For example, observe the following commands:

Router# *clear*
Router# *clock*
Router# *configure*
Router# *connect*
Router# *copy*
Router# *debug*

We could use *de* as a shortcut for the *debug* command, as no other command here begins with *de*. We could not, however, use *co* as a shortcut, as three commands begin with those letters. The following error would be displayed:

Router# *co*

% Ambiguous command: "co"

If you type a command incorrectly, the IOS will point out your error:

Router# *clcok*
^
% Invalid input detected at "^" marker

Keyboard Shortcuts

Several hotkeys exist to simplify using the IOS interface:

Keyboard Shortcut	***Result***
CTRL-B (or Left-Arrow)	*Moves cursor back one character*
CTRL-F (or Right-Arrow)	*Moves cursor forward one character*
CTRL-A	*Moves cursor to beginning of a line*
CTRL-E	*Moves cursor to end of a line*
ESC-B	*Moves cursor back one word*
ESC-F	*Moves cursor forward one word*
CTRL-P (or Up-Arrow)	*Returns previous command(s) from history buffer*
CTRL-N (or Down-Arrow)	*Returns next command from history buffer*
CTRL-Z	*Exits out of the current mode*
TAB	*Finishes an incomplete command (assuming it is not ambiguous)*

(Reference: http://www.cisco.com/en/US/products/hw/switches/ps708/products_configuration_guide_chapter09186a008007e6d5.html#wp1028871)

* * *

Terminal History Buffer

As implied in the previous section, the Cisco IOS keeps a **history** of previously entered commands. By default, this history buffer stores the previous **10** commands entered. To view the terminal history buffer:

RouterA# *show history*

```
enable
config t
hostname RouterA
exit
show history
```

The **Up-Arrow** key (or **CTRL-P)** allows you to scroll through previously entered commands. To scroll back down the list, use the **Down-Arrow** key (or **CTRL-N**).

To adjust the number of commands the history buffer stores (range 0-256):

RouterA# *terminal history size 30*

* * *

IOS Context-Sensitive Help

The **question mark** *(?)* is one of the most powerful tools in the Cisco IOS, as it provides **context-sensitive help** for each IOS mode.

Typing *?* at the command prompt displays a list of all commands available at that mode, with explanations:

Router# *?*

```
access-enable    Create a temporary Access-List entry
access-profile   Apply user-profile to interface
access-template  Create a temporary Access-List entry
alps             ALPS exec commands
archive          manage archive files
audio-prompt     load ivr prompt
bfe              For manual emergency modes setting
call             Load IVR call application
cd               Change current directory
clear            Reset functions
clock            Manage the system clock
configure        Enter configuration mode
connect          Open a terminal connection
copy             Copy from one file to another
debug            Debugging functions (see also 'undebug')

<snip>
```

Typing in part of a command with a *?* displays a list of all commands that begin with those characters:

Router# *co?*

```
configure   connect    copy
```

Typing in a full command followed by a *?* displays the available options and arguments for that command:

Router# *clock ?*

```
set    Set the time and date
```

Notice the space between the command *clock* and the *?*.

* * *

The "Show" Command

The *show* command provides the ability to view a wide variety of configuration and status information on your router. The command is executed from Privileged mode, and the syntax is simple:

Router# *show* [argument]

There are literally dozens of arguments for the *show* command, and each provides information on a specific aspect of the router. Numerous *show* commands will be described throughout this and most other guides.

One common *show* command displays the IOS version, configuration-register settings, router uptime, and basic router hardware information:

Router# *show version*

```
Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-IS-L), Version 12.3(1a), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-2003 by cisco Systems, Inc.
Compiled Fri 09-Jan-03 11:23 by xxxxx
Image text-base: 0x0307F6E8, data-base: 0x00001000

ROM: System Bootstrap, Version 11.0(10c)XB2, PLATFORM SPECIFIC RELEASE SOFTWARE
(fc1)
BOOTLDR: 3000 Bootstrap Software (IGS-BOOT-R), Version 11.0(10c)XB2, PLATFORM
SPECIFIC RELEASE SOFTWARE (fc1)

Router uptime is 2 minutes
System returned to ROM by reload
System image file is "flash:c2500-is-l.123-1a.bin"

cisco 2500 (68030) processor (revision L) with 14336K/2048K bytes of memory.
Processor board ID 13587050, with hardware revision 00000000
Bridging software.
X.25 software, Version 3.0.0.
2 Ethernet/IEEE 802.3 interface(s)
2 Serial network interface(s)
32K bytes of non-volatile configuration memory.
16384K bytes of processor board System flash (Read ONLY)

Configuration register is 0x2102
```

(Example *show version* output from: http://www.cisco.com/en/US/products/hw/routers/ps233/products_tech_note09186a008009464c.shtml)

The following command provides output similar to *show version*:

Router# *show hardware*

Other common *show* commands will be described shortly.

* * *

Enable Passwords

The *enable* password protects a router's Privileged mode. This password can be set or changed from Global Configuration mode:

Router(config)# *enable password MYPASSWORD*
Router(config)# *enable secret MYPASSWORD2*

The *enable password* command sets an unencrypted password intended for legacy systems that do not support encryption. It is no longer widely used.

The *enable secret* command sets an MD5-hashed password, and thus is far more secure. The *enable password* and *enable secret* passwords **cannot be identical**. The router will not accept identical passwords for these two commands.

Line Passwords and Configuration

Passwords can additionally be configured on router **lines**, such as telnet (vty), console, and auxiliary ports. To change the password for a console port and all telnet ports:

Router(config)# *line console 0*
Router(config-line)# *login*
Router(config-line)# *password cisco1234*

Router(config-line)# *exec-timeout 0 0*
Router(config-line)# *logging synchronous*

Router(config)# *line vty 0 4*
Router(config-line)# *login*
Router(config-line)# *password cisco1234*

Router(config-line)# *exec-timeout 0 0*
Router(config-line)# *logging synchronous*

The *exec-timeout 0 0* command is optional, and disables the automatic timeout of your connection. The two zeroes represent the timeout value in minutes and seconds, respectively. Thus, to set a timeout for 2 minutes and 30 seconds:

Router(config-line)# *exec-timeout 2 30*

The *logging synchronous* command is also optional, and prevents system messages from interrupting your command prompt.

By default, line passwords are stored in clear-text in configuration files. To ensure these passwords are encrypted in all configuration files:

Router(config)# *service password–encryption*

* * *

Router Interfaces

Recall that, to configure an interface, one must specify both the *type* of interface, and the interface *number* (which always begins at "0"). Thus, to configure the first Ethernet interface on a router:

Router(config)# *interface ethernet 0*
Router(config-if)#

Certain router families (such as the 3600 series) are modular, and have multiple "slots" for interfaces. All commands must reflect both the **module number** *and* the interface number, formatted as: *module/interface*

Thus, to configure the third Fast Ethernet interface off of the first module:

Router(config)# *interface fastethernet 0/2*
Router(config-if)#

By default, all router interfaces are **administratively shutdown**. To take an interface out of an administratively shutdown state:

Router(config)# *interface fa 0/0*
Router(config-if)# *no shutdown*

Notice the use of *fa* as a shortcut for *fastethernet* in the above example. To manually force an interface into a shutdown state:

Router(config-if)# *shutdown*

To assign an IP address to an interface:

Router(config-if)# *ip address 192.168.1.1 255.255.255.0*

An additional *secondary* IP Address can be assigned to an interface:

Router(config-if)# *ip address 192.168.1.1 255.255.255.0*
Router(config-if)# *ip address 192.168.1.2 255.255.255.0 secondary*

Serial interfaces require special consideration. The **DCE** (**Data Communication Equipment)** side of a serial connection must set the speed, or *clock rate*, for the **DTE (Data Terminal Equipment)** side**.** *Clock rate* is measured in BPS (bits-per-second).

To set the *clock rate,* if you are the DCE side of a serial connection:

Router(config)# *interface serial 0*
Router(config-if)# *clock rate 64000*

* * *

Status of Router Interfaces

To view the current status and configuration of all interfaces:

Router# *show interfaces*

```
Ethernet 0 is up, line protocol is up
  Hardware is Ethernet, address is 5520.abcd.1111
  Internet address is 192.168.1.1, subnet mask is 255.255.255.0
  MTU 1500 bytes, BW 10000 Kbit, DLY 100000 usec, rely 255/255, load 1/255
  Encapsulation ARPA, loopback not set, keepalive set (10 sec)
  ARP type: ARPA, ARP Timeout 4:00:00
  Last input 0:00:00, output 0:00:00, output hang never
  Last clearing of "show interface" counters 0:00:00
  Output queue 0/40, 0 drops; input queue 0/75, 0 drops
  Five minute input rate 0 bits/sec, 0 packets/sec
  Five minute output rate 2000 bits/sec, 4 packets/sec
     53352 packets input, 351251 bytes, 0 no buffer
     Received 4125 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     12142 packets output, 16039 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets, 0 restarts

Ethernet 1 is up, line protocol is up
  Hardware is Ethernet, address is 5520.abcd.1112

  <snip>
```

The *show interfaces* command displays a plethora of information, including:

- Current interface status *(ethernet 0 is up, line protocol is up)*
- MAC address (*5520.abcd.1111)*
- IP address (*192.168.1.1*)
- MTU (*1500 bytes)*
- Bandwidth *(10 Mbps)*
- Output and input queue status
- Traffic statistics (*packets input, packets output, collisions, etc.)*

To view the current status of a *specific* interface:

Router# *show interfaces ethernet 0*

To view only IP information for all interfaces:

Router# *show ip interface brief*

```
Interface     IP Address    OK?  Method Status                Protocol

Ethernet0     192.168.1.1   YES  NVRAM  up                    up
Ethernet1     192.168.2.1   YES  NVRAM  up                    up
Serial0       unassigned    YES  unset  administratively down down
```

* * *

Status of Router Interfaces (continued)

Traffic can only be routed across an interface if its status is as follows:

Serial 0 is up, line protocol is up

The first part of this status (*Serial0 is up)* refers to the **physical layer** status of the interface. The second part (*line protocol is up)* refers to the **data-link layer** status of the interface. A status of *up/up* indicates that the physical interface is active, and both sending and receiving keepalives.

An interface that is physically down will display the following status:

Serial 0 is down, line protocol is down

The mostly likely cause of the above status is a defective (or unplugged) cable or interface.

There are several potential causes of the following status:

Serial 0 is up, line protocol is down

Recall that *line protocol* refers to data-link layer functions. Potential causes of the above status could include:

- Absence of keepalives being sent or received
- Clock rate not set on the DCE side of a serial connection
- Different encapsulation types set on either side of the link

An interface that has been administratively shutdown will display the following status:

Serial 0 is administratively down, line protocol is down

* * *

Managing Configuration Files

Cisco IOS devices employ two distinct configuration files

- **running-config** – stored in RAM, contains the *active* configuration
- **startup-config** – stored in NVRAM (or flash), contains the *saved* configuration

Any configuration change made to an IOS device is made to the running-config. Because the running-config file is stored in RAM, the contents of this file will be lost during a power-cycle. Thus, we must save the contents of the running-config to the startup-config file. We accomplish this by using the *copy* command from Privileged mode:

Router# *copy running-config startup-config*

The copy command follows a very specific logic: *copy [from] [to]*. Thus, if we wanted to copy the contents of the startup-config file to running-config:

Router# *copy startup-config running-config*

We can use shortcuts to simplify the above commands:

Router# *copy run start*
Router# *copy start run*

To view the contents of the running-config and startup-config files:

Router# *show run*
Router# *show start*

To delete the contents of the startup-config file:

Router# *erase start*

If the router is power-cycled after erasing the startup-config file, the router will enter **Initial Configuration Mode (**sometimes called **Setup Mode**). This mode is a series of interactive questions intended for quick reconfiguration of the router.

Initial Configuration Mode can be exited by typing CTRL-C.

* * *

"Piping" Commands

In newer versions of the Cisco IOS, the output of *show* commands can be filtered to remove irrelevant lines, through the use of the **pipe " | "** character.

The following command will display the contents of the startup-config, **begin**ning with the first line containing the text *ethernet*:

Router# *show startup | begin ethernet*

The following command will **exclude** all lines containing the text *ethernet*:

Router# *show startup | exclude ethernet*

The following command will **include** all lines containing the text *ethernet*:

Router# *show startup | include ethernet*

Miscellaneous Commands

To change the hostname of your router:

Router(config)# *hostname MyRouter*
MyRouter(config)# *hostname MyRouter*

To assign a description to an interface for documentation purposes:

Router(config)# *interface serial 0*
Router(config-if)# *description SBC T1 connection to Chicago*

Router# *show interfaces*

```
Serial 0 is up, line protocol is up
  Hardware is Serial
  Internet address is 70.22.3.1, subnet mask is 255.255.255.0
  Description: SBC T1 connection to Chicago
```

To create a *banner* message which users will see when logging into an IOS device:

Router(config)# *banner motd #*

```
Logging into this router without authorization is illegal
and will be prosecuted!
#
```

The # symbol is used as a delimiter to indicate the beginning and end of the banner. Any character can be used as a delimiter.

* * *

IOS Troubleshooting Commands

The *show tech-support* command prints to screen every configuration file, and the output of several important *show* commands. This can be redirected to a file and either viewed or sent to Cisco for troubleshooting purposes:

Router# *show tech-support*

The *debug* command is a powerful tool to view real-time information and events occurring on an IOS device. As with the *show* command, there are a multitude of arguments for the *debug* command. An example *debug* command is as follows:

Router# *debug ip rip events*

To disable a specific debugging command, simply preprend the word *no* in front of the command:

Router# *no debug ip rip events*

To enable all possible debugging options on an IOS device:

Router# *debug all*

Using the *debug all* command is not recommended, as it will critically impair router performance.

To disable all possible debugging options on an IOS device:

Router# *no debug all*

Section 6
- Switch Port Configuration -

Cisco Operating Systems

Cisco offers two brands of network switches:

- **Catalyst –** Cisco's flagship switching platform, with a large selection of models spanning access, distribution, and core layers.
- **Nexus –** high-end switches focused at datacenter environments.

Depending on the brand and model, Cisco supports one of three switch operating systems:

- **Catalyst OS (CatOS) -** interface based on *set* commands, that is almost entirely deprecated. CatOS will not be covered in this guide.
- **IOS –** interface that is nearly identical to the Cisco router IOS, except for switching-specific commands.
- **NX-OS –** interface supported exclusively on Nexus brand switches.

The following details the supported operating system for various Cisco platforms:

CatOS	IOS	NX-OS
• Catalyst 1200	• Catalyst 29xx	• Nexus 1000V
• Catalyst 2948	• Catalyst 35xx	• Nexus 3000
• Catalyst 4000	• Catalyst 36xx	• Nexus 4000
• Catalyst 4500	• Catalyst 37xx	• Nexus 5000
• Catalyst 5000	• Catalyst 38xx	• Nexus 7000
• Catalyst 5500	• Catalyst 4500	• Nexus 9000
• Catalyst 6000	• Catalyst 4900	
• Catalyst 6500	• Catalyst 6500	
	• Catalyst 6800	

The basic IOS interface is nearly identical between switches and routers, and is covered in *great* detail in other guides on this site:

- Introduction to the Cisco IOS
- Advanced IOS Functions

This guide will focus on switch-specific IOS functions. For a more comprehensive IOS review, consult the guides listed above.

* * *

Switch Port Configuration

Traditionally, network connections on switches have been referred to as **ports,** while on routers they are referred to as **interfaces.** However, in the Cisco IOS, switch ports are referred to as interfaces as well.

Additionally, most Cisco switches are **modular**. Thus, configuration commands must reflect both the *module* and *interface* number, in the following format: *module/interface*

Some catalyst switches support being **stacked** – essentially, multiple physical switches connected together to form one logical switch. Configuration commands must reflect the *stack, module,* and *interface* number*,* in the following format: *stack/module/interface*

To enter interface configuration mode for the *third* Fast Ethernet interface off of the *second* module:

Switch(config)# *interface FastEthernet 2/3*
Switch(config-if)#

Note that most switches will number their modules and interfaces starting at *1,* while most routers will number their modules/interfaces starting at *0.*

The above command can be *shortened*, as long as the truncated command is not ambiguous:

Switch(config)# *interface fa 2/3*

If the interface above was Gigabit Ethernet instead of Fast Ethernet:

Switch(config)# *interface GigabitEthernet 2/3*
Switch(config)# *interface gi 2/3*

Note: On Nexus switches using the NX-OS, the speed of an Ethernet interface is not used to identify it – all interfaces are simply *Ethernet:*

NexusSwitch(config)# *interface Ethernet 2/3*

Multiple interfaces can be configured simultaneously:

Switch(config)# *interface range gi2/3 , gi2/5 , gi2/7*
Switch(config-if-range)#

The above command selects interfaces *gi2/3, gi2/5, gi2/7*. Please note that there is a space on either side of each comma.

* * *

Switch Port Configuration (continued)

A contiguous range of interfaces can be specified:

Switch(config)# *interface range gi3/10 - 15*

The above command selects interfaces *gi3/10* through *gi3/15*. Please note the space on either side of the dash.

Macros can be created for groups of interfaces that are configured often:

Switch(config)# *define interface-range MACRONAME gi3/10 – 15*
Switch(config)# *interface range macro MACRONAME*
Switch(config-if-range)#

The first command creates a macro (or group) of interfaces called *MACRONAME*. The second command actually selects those interfaces for configuration.

Descriptions can be applied to an interface for documentation purposes:

Switch(config)# *interface gi3/10*
Switch(config-if)# *description WEBSERVER*

Spaces are allowed in a description:

Switch(config)# *interface gi2/10*
Switch(config-if)# *description CHECK PRINTER IN PAYROLL*

Descriptions can greatly assist in troubleshooting issues, as long as they are accurate. Descriptions on access-layer switches can be difficult to manage, as hosts may change interfaces often.

Scenarios where adding descriptions to interfaces is valuable:

- Switch or router uplinks
- WAN connections
- Datacenter hosts – such as servers or firewalls

Descriptions should be as detailed as possible. For example, including the circuit number on a WAN connection can accelerate resolution of an outage.

If an interface *does not* have an accurate description, there are two methods of determining what is connected to it:

- Trace the physical cable to the host (always fun)
- Leverage the CAM table to identify a host by its MAC address

* * *

Switch Port Configuration – Speed and Duplex

Some switch interfaces are **fixed** at a single speed. Often though, a single switch interface can support multiple speeds.

For example, a Gigabit Ethernet interface is often backwards-compatible with original and Fast Ethernet, and is referred to as a *10/100/1000* interface.

To manually specify the **speed** of an interface:

```
Switch(config)# interface gi3/10
Switch(config-if)#  speed 10
Switch(config-if)#  speed 100
Switch(config-if)#  speed 1000
```

The **duplex** of an interface can be manually specified as well:

```
Switch(config)#  interface gi3/10
Switch(config-if)#  duplex half
Switch(config-if)#  duplex full
```

Ethernet also has the ability to **autonegotiate** both the speed and duplex of an interface. Autonegotiation will attempt to use the *fastest* speed available, and will attempt to use *full-duplex* if both devices support it.

Only one command is required to force an interface to autonegotiate its speed and duplex:

```
Switch(config)# interface gi3/10
Switch(config-if)#  speed auto
```

The configuration *must* be consistent on both sides of the connection. Both sides must be configured to autonegotiate, or both sides must be hardcoded with *identical* settings. Otherwise a **duplex mismatch** error can occur.

For example, if a workstation's interface is configured to autonegotiate, and the switch interface is hardcoded for 100Mbps and full-duplex, then a duplex mismatch will occur. The workstation's interface will sense the correct speed of 100Mbps, but will not detect the correct duplex and will default to *half-duplex.*

If the duplex is mismatched, collisions will occur. Because the full-duplex side of the connection does not utilize CSMA/CD, performance is *severely* degraded. These issues can be difficult to troubleshoot, as the network connection will still function, but will be excruciatingly slow.

* * *

Switch Port Configuration – Speed and Duplex (continued)

When autonegotiation was first developed, manufacturers did not always adhere to the same standard. This resulted in frequent mismatch issues, and a sentiment of distrust towards autonegotiation.

Though modern network hardware has alleviated most of the incompatibility, many administrators are still skeptical of autonegotiation and choose to hardcode all connections. Another common practice is to hardcode server and datacenter connections, but to allow user devices to autonegotiate.

Gigabit Ethernet provided several enhancements to autonegotiation, such as hardware flow control. Most manufacturers **recommend autonegotiation** on Gigabit Ethernet interfaces as a best practice.

Switch Port Configuration - Shutdown

By default on most Cisco switches, all interfaces will be in an **administratively shutdown** state. An interface that is shutdown will not send or receive traffic.

To take an interface out of a shutdown state:

Switch(config)# *interface gi3/10*
Switch(config-if)# *no shutdown*

Intuitively, to place the interface into a shutdown state again:

Switch(config)# *interface gi3/10*
Switch(config-if)# *shutdown*

It is possible to shut or no shut multiple interface simultaneously, using the range command:

Switch(config)# *interface range gi3/10 – 15*
Switch(config-if-range)# *no shut*

Be careful – do not unintentionally shutdown the wrong interfaces when using the range command.

It is possible to apply any configuration to an interface that is shutdown. However, the interface (and corresponding configuration) will not become active until the interface is *no shut*.

* * *

Switch Port Error Conditions

Cisco switches can detect **error conditions** on an interface. If the error is severe enough, the switch can automatically disable that interface. An interface that is disabled due to an error is considered to be in an **errdisable state.**

The following events can put a port into errdisable state:

arp-inspection	Error with dynamic ARP inspection, used to prevent ARP poisoning.
bpduguard	BPDU is received on an interface configured for STP Portfast and BPDU guard.
channel-misconfig	Error with an Etherchannel configuration
dhcp-rate-limit	Error with DHCP Snooping
dtp-flap	*Flapping* between trunk encapsulations (ISL or 802.1Q)
gbic-invalid	Invalid SFP or GBIC module
ilpower	Error with inline power
loopback	Interface is *looped* (received a keepalive packet that it sent out).
l2tpguard	Error with L2 tunneling
link-flap	Interface is flapping between an *up* or *down* state
mac-limit	Violation of MAC address limit
pagp-flap	Flapping of an Etherchannel during PaGP negotiation, usually due to misconfiguration
psecure-violation	Violation of port-security
rootguard	BPDU from a root bridge received on a non-designated port
security-violation	Violation of 802.1x security
storm-control	Broadcast storm detected
udld	Unidirectional link detected
unicast-flood	Violation of unicast flooding limit

(Reference: http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/69980-errdisable-recovery.html)

* * *

Switch Port Error Conditions (continued)

All errdisable conditions are enabled by default. To disable a specific errdisable condition:

Switch(config)# *no errdisable detect cause link-flap*

It is also possible to disable *all* errdisable conditions:

Switch(config)# *no errdisable detect cause all*

To enable errdisable conditions again, either individually or collectively:

Switch(config)# *errdisable detect cause udld*
Switch(config)# *errdisable detect cause all*

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

Switch(config)# *interface gi3/10*
Switch(config-if)# *shut*
Switch(config-if)# *no shut*

However, if the error condition still exists, the interface will be placed back into an errdisable state again. Thus, an interface should not be recovered until the root cause is resolved.

Interfaces can also *automatically* recover from an error condition. First, the errdisable conditions that should auto-recover must be individually or collectively identified:

Switch(config)# *errdisable recovery cause udld*
Switch(config)# *errdisable recovery cause all*

By default, an interface will recover from an errdisable state in **300 seconds**. This timer can be adjusted:

Switch(config)# *errdisable recovery interval 250*

To view which errdisable conditions are currently enabled for recovery:

Switch# *show errdisable recovery*

By default, automatic errdisable recovery is **disabled for all conditions.**

Note that all errdisable commands are applied *globally*, and thus apply to all interfaces on the switch.

* * *

Troubleshooting Switch Port s

Cisco switches offer several *show* commands for monitoring and troubleshooting switch ports.

The *show interface* is invaluable, and provides the following information:

- Current interface status – *up, down, administratively down, errdisable*
- Speed and duplex
- Current input/output rates
- Historical packet input/output and error statistics

To view the details of a specific interface, such as GigabitEthernet 3/10:

Switch# *show interface gi3/10*

```
GigabitEthernet3/10 is up, line protocol is up (connected)
  Hardware is Gigabit Ethernet Port, address is 2c54.2db2.dab1
  MTU 1500 bytes, BW 100000 Kbit/sec, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 100Mb/s, link type is auto, media type is 10/100/1000-TX
  input flow-control is on, output flow-control is on
  Auto-MDIX on (operational: on)
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:25, output never, output hang never
  Last clearing of "show interface" counters 1y9w
  Input queue: 0/2000/0/0 (size/max/drops/flush); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 6000 bits/sec, 8 packets/sec
     190197453 packets input, 22130686309 bytes, 0 no buffer
     Received 1660069 broadcasts (1295453 multicasts)
     0 runts, 110243 giants, 0 throttles
     110243 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 input packets with dribble condition detected
     518261888 packets output, 352807215573 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
```

To view details of *all* interfaces:

Switch# *show interface*

* * *

Troubleshooting Switch Port s (continued)

The *show interface* command provides an occasionally overwhelming amount of detailed information.

The *show interface status* command provides a high-level summary of all switch interfaces:

Switch# *show interface status*

```
Port      Name     Status       Vlan       Duplex  Speed Type
Gi2/1              connected    15         a-full  a-100 10/100/1000-TX
Gi2/2              notconnect   15           auto   auto 10/100/1000-TX
Gi2/3              connected    15         a-full  a-100 10/100/1000-TX
Gi2/4              connected    15         a-full a-1000 10/100/1000-TX
Gi2/5              connected    15         a-half   a-10 10/100/1000-TX
Gi2/6              notconnect   15           auto   auto 10/100/1000-TX
Gi2/7              connected    15         a-full  a-100 10/100/1000-TX
Gi2/8              connected    15         a-full  a-100 10/100/1000-TX
Gi2/9              connected    15         a-full  a-100 10/100/1000-TX
Gi2/10             notconnect   15           auto   auto 10/100/1000-TX
Gi2/11             notconnect   9            auto   auto 10/100/1000-TX

<snipped for brevity>
```

To view which interfaces are currently in an errdisable state, including the reason for the errdisable condition:

Switch# *show interface status err-disabled*

```
Port      Name     Status             Reason
Gi3/1              err-disabled       rootguard
Gi3/3              err-disabled       ilpower
Gi3/8              err-disabled       security-violation
Gi3/9              err-disabled       grue-eats-your-face
```

Watch out for the grue.

* * *

Part III

Switching Protocols and Functions

---

* * *

Section 7
- Virtual LANs (VLANs) and VTP -

Collision vs. Broadcast Domains

A **collision domain** is simply defined as any physical segment where a **collision** can occur. Hubs can only operate at half-duplex, and thus all ports on a hub belong to the same collision domain.

Layer-2 switches can operate at full duplex. *Each individual port* on a switch belongs to its *own collision domain*. Thus, Layer-2 switches create **more collision domains**, which results in **fewer collisions.**

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

Only Layer-3 devices separate broadcast domains. Because of this, Layer-2 switches are poorly suited for large, scalable networks. The Layer-2 header provides no mechanism to differentiate one *network* from another, only one *host* from another.

Virtual LANs (VLANs)

By default, a switch will forward both broadcasts and multicasts out *every* port but the originating port. However, a switch can be *logically* segmented into separate broadcast domains, using **Virtual LANs** (or **VLANs**).

Each VLAN represents a unique broadcast domain:

- Traffic between devices within the *same* VLAN is switched.
- Traffic between devices in *different* VLANs requires a Layer-3 device to communicate.

Broadcasts from one VLAN will not be forwarded to another VLAN. The logical separation provided by VLANs is **not a Layer-3 function.** VLAN tags are inserted into the **Layer-2 header**.

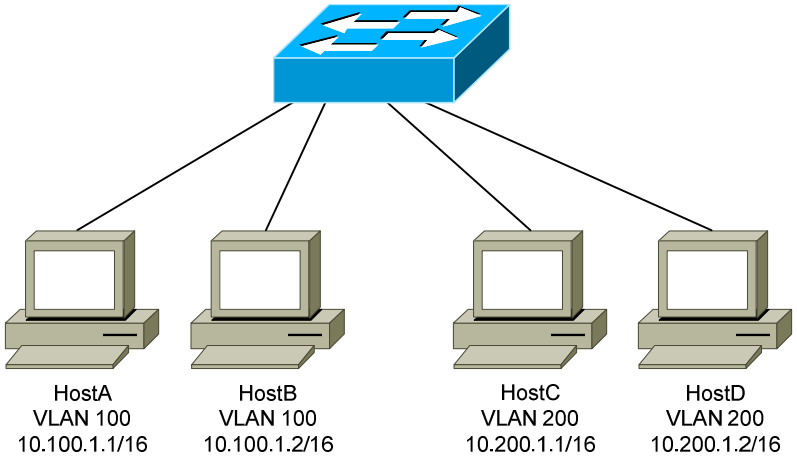
Thus, a switch that supports VLANs is not necessarily a Layer-3 switch. However, a purely Layer-2 switch cannot route between VLANs.

Remember, though VLANs provide separation for *Layer-3* broadcast domains, they are still a *Layer-2* function. A VLAN often has a direct relationship with an IP subnet, though this is not a requirement.

* * *

VLAN Example

Consider the following example:



Four hosts are connected to a Layer-2 switch that supports VLANs:

- HostA and HostB belong to VLAN 100
- HostC and HostD belong to VLAN 200

Because HostA and HostB belong to the *same* VLAN, they belong to the same broadcast domain as well. The switch will be able to forward frames between the two hosts without the need of a Layer-3 device, such as a router.

Likewise, HostC and HostD belong to the same VLAN, and thus the same broadcast domain. They also will be able to communicate without an intervening Layer-3 device.

However, HostA and HostB *will not* be able to communicate with HostC and HostD. They are members of separate VLANs, and belong in *different* broadcast domains. Thus, a Layer-3 device is required for those hosts to communicate.

A broadcast sent from a host in VLAN 100 will be received by all other hosts in that same VLAN. However, that broadcast will not be forwarded to any other VLAN, such as VLAN 200.

On Cisco switches, all interfaces belong to **VLAN 1** by default**.** VLAN 1 is also considered the **Management VLAN,** and should be dedicated for system traffic such as CDP, STP, VTP, and DTP.

* * *

Advantages of VLANs

VLANs provide the several benefits:

- **Broadcast Control –** eliminates unnecessary broadcast traffic, improving network performance and scalability.
- **Security –** logically separates users and departments, allowing administrators to implement access-lists to control traffic between VLANs.
- **Flexibility –** removes the physical boundaries of a network, allowing a user or device to exist anywhere.

VLANs are very common in LAN and campus networks. For example, user networks are often separated from server networks using VLANs.

VLANs can span across WANs as well, though there are only limited scenarios where this is necessary or recommended.

VLAN Membership

VLAN membership can be configured one of two ways:

- **Statically**
- **Dynamically**

Statically assigning a VLAN involves manually assigning an individual or group of ports to a VLAN. Any host connected to that port (or ports) immediately becomes a member of that VLAN. This is *transparent* to the host - it is unaware that it belongs to a VLAN.

VLANs can be assigned **dynamically** based on the MAC address of the host. This allows a host to remain in the same VLAN, regardless of which switch port it is connected to.

Dynamic VLAN assignment requires a separate database to maintain the MAC-address-to-VLAN relationship. Cisco developed the **VLAN Membership Policy Server (VMPS)** to provide this functionality.

In more sophisticated systems, a user's network account can be used to determine VLAN membership, instead of a host's MAC address.

Static VLAN assignment is far more common than dynamic, and will be the focus of this guide.

* * *

Creating VLANs

By default, all interfaces belong to **VLAN 1.** To assign an interface to a different VLAN, that VLAN must first be *created*:

Switch(config)# *vlan 100*
Switch(config-vlan)# *name SERVERS*

The first command creates VLAN 100, and enters VLAN configuration mode. The second command assigns the name *SERVERS* to this VLAN.

Note that naming a VLAN is *not* required.

The standard range of VLAN numbers is **1 – 1005,** with VLANs 1002-1005 reserved for legacy Token Ring and FDDI purposes.

A switch operating in VTP **transparent mode** can *additionally* use the VLAN range of **1006 – 4094.** These are known as extended-range VLANs. VTP is covered in great detail later in this guide.

To remove an individual VLAN:

Switch(config)# *no vlan 100*

Note that VLAN 1 cannot be removed. To remove a group of VLANs:

Switch(config)# *no vlan 150-200*

To view all created VLANs, including the interfaces assigned to each VLAN:

Switch# *show vlan*

```
VLAN Name                             Status    Ports
---- -------------------------------- --------- -----------
1    default                          active    gi1/1-24
100  SERVERS                          active
1002 fddi-default                     suspended
1003 token-ring-default               suspended
1004 fddinet-default                  suspended
1005 trnet-default                    suspended
```

Note that no interfaces have been assigned to the newly created VLAN 100 yet.

* * *

Statically Assigning VLANs

To statically assign an interface into a specific VLAN:

Switch(config)# *interface gi1/10*
Switch(config-if)# *switchport mode access*
Switch(config-if)# *switchport access vlan 100*

The first command enters interface configuration mode. The second command indicates that this is an *access* port, as opposed to a *trunk* port. This will be explained in detail shortly.

The third command assigns this access port to VLAN *100*. Note that the VLAN *number* is specified, and not the VLAN *name*.

The *show vlan* command should now reflect the new VLAN assignment:

Switch# *show vlan*

```
VLAN Name                             Status    Ports
---- -------------------------------- --------- -----------
1    default                          active    gi1/1-9,11-24
100  SERVERS                          active    gi1/10
1002 fddi-default                     suspended
1003 token-ring-default               suspended
1004 fddinet-default                  suspended
1005 trnet-default                    suspended
```

For switches running in VTP **server** or **client mode**, the *list* of VLANs are stored in a database file named **vlan.dat.** The vlan.dat file is usually stored in **flash**, though on some switch models it is stored in NVRAM. The VLAN database will be maintained even if the switch is rebooted.

For switches running in VTP **transparent mode**, the list of VLANs is stored in the startup-config file in NVRAM. VTP is covered extensively later in this guide.

Regardless of VTP mode, the VLAN *assignment* for every switch interface is stored in the switch's **startup-config**.

VLAN Port Types

A VLAN-enabled switch supports two types of ports:

- **Access ports**
- **Trunk ports**

An **access port** is a member of only a *single* VLAN. Access ports are most often used to connect host devices, such as computers and printers. By default on Cisco switches, *all* switch ports are access ports.

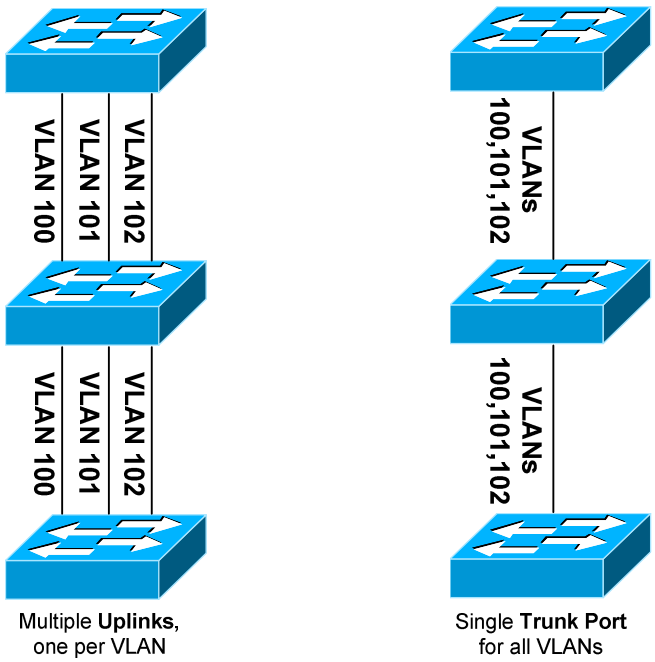
Any host connected to an access port immediately becomes a member of the VLAN configured on that port. This is *transparent* to the host - it is unaware that it belongs to a VLAN.

It is possible for a VLAN to span more than one switch. There are two methods of connecting a VLAN across multiple switches:

- Create *uplink* access ports between the switches, one for each VLAN.
- Create a **trunk connection** between the switches.

A **trunk port** is not a member of a single VLAN. The traffic from *any or all* VLANs can traverse trunk links to reach other switches.

Uplinking access ports quickly becomes unfeasible in large switching environments. The following illustrates the advantage of using trunk ports:

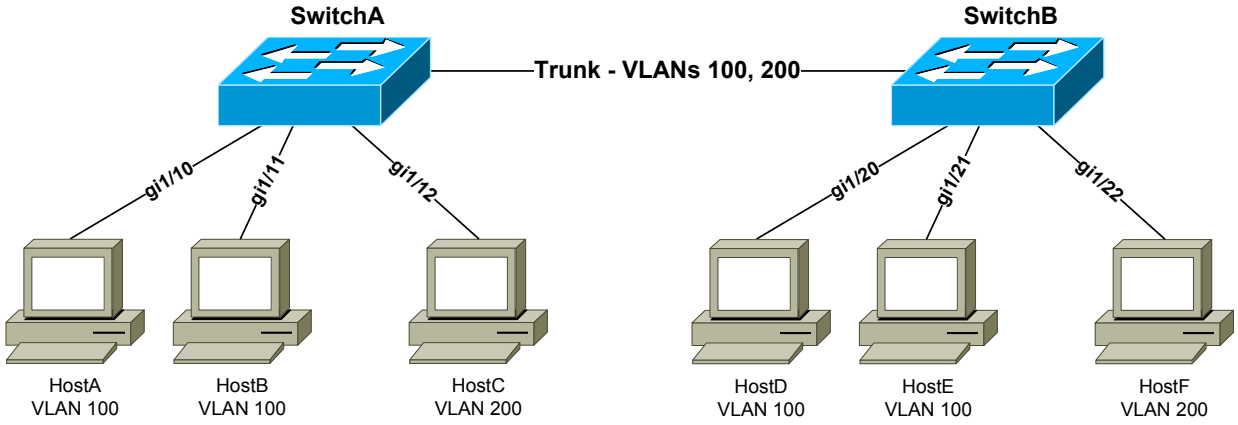


* * *

VLAN Frame-Tagging

When VLANs span multiple switches, a mechanism is required to identify which VLAN a frame belongs to. This is accomplished through **frame tagging,** which places a VLAN ID in each frame.

Tagging *only* occurs when a frame is **sent out a trunk port**. Traffic sent out access ports is never tagged. Consider the following example:



If HostA sends a frame to HostB, no frame tagging will occur:

- The frame never leaves the SwitchA.
- The frame stays within its own VLAN.
- The frame is simply **switched** to HostB.

If HostA sends a frame to HostC, which is in a separate VLAN:

- The frame *again* never leaves the switch.
- Frame tagging will *still* not occur.
- Because HostC is in a different VLAN, the frame must be **routed**.

If HostA sends a frame to HostD, which is on a separate switch:

- The frame is sent out the trunk port to SwitchB.
- The frame *must* be **tagged** as it is sent out the trunk port.
- The frame is tagged with its VLAN ID - VLAN 100 in this example.
- When SwitchB receives the frame, it will only forward it out ports belonging to VLAN 100 – *gi1/20* and *gi1/21*.
- If SwitchB has HostD's MAC address in its table, it will forward the frame only out the appropriate port – *gi1/20*.
- The VLAN tag is stripped from the frame before being forwarded to the host.

* * *

Frame Tagging Protocols

Cisco switches support two frame tagging protocols:

- **Inter-Switch Link (ISL)**
- **IEEE 802.1Q**

The tagging protocol can be manually specified on a trunk port, or dynamically negotiated using Cisco's proprietary **Dynamic Trunking Protocol (DTP)**.

Inter-Switch Link (ISL)

Inter-Switch Link (ISL) is Cisco's proprietary frame tagging protocol. ISL supports several technologies:

- Ethernet
- Token Ring
- FDDI
- ATM

ISL *encapsulates* a frame with an additional header (**26 bytes**) and trailer (**4 bytes**). Thus, ISL increases the size of a frame by **30 bytes**.

The header contains several fields, including a 15-bit VLAN ID. The trailer contains an additional 4-byte CRC to verify data integrity.

Normally, the maximum possible size of an Ethernet frame is **1518 bytes.** This is known as the Maximum Transmission Unit (**MTU**). Most Ethernet devices use a *default* MTU of **1514 bytes.**

ISL increases the frame size by another 30 bytes. Thus, most switches will disregard ISL-tagged frames as being **oversized,** and drop the frame. An oversized frame is usually referred to as a **giant**. Somewhat endearingly, a *slightly* oversized frame is known as a **baby giant.**

Cisco switches are specifically engineered to support these giant ISL – tagged frames. Note that this is a key reason why ISL is Cisco-proprietary.

ISL supports a maximum of **1000 VLANs** on a trunk port. ISL is also almost entirely deprecated - most modern Cisco switches no longer support it.

(Reference: http://www.cisco.com/c/en/us/support/docs/lan-switching/8021q/17056-741-4.html)

* * *

IEEE 802.1Q

IEEE 802.1Q, otherwise referred to as **dot1Q**, is an industry-standard frame-tagging protocol.

802.1Q is supported by nearly all switch manufacturers, including Cisco. Because 802.1Q is an open standard, switches from different vendors can be trunked together.

Recall that ISL encapsulates a frame with an additional header and trailer. In contrast, 802.1Q *embeds* a **4-byte VLAN tag** directly into the Layer-2 frame header. Because the Layer-2 header is modified, 802.1Q must recalculate the frame's CRC value.

The VLAN tag includes a 12-bit VLAN ID. This tag increases the size of an Ethernet frame, from its default of 1514 bytes to 1518 bytes. Nearly all modern switches support the 802.1Q tag and the slight increase in frame size.

802.1Q supports a maximum of **4096 VLANs** on a trunk port.

Configuring Trunk Links

To manually configure an interface as a trunk port:

```
Switch(config)#  interface gi2/24
Switch(config-if)#  switchport mode trunk
```

For a switch that supports both ISL and 802.1Q, the tagging or *encapsulation* protocol must be configured first:

```
Switch(config)#  interface gi2/24
Switch(config-if)#  switchport trunk encapsulation isl
Switch(config-if)#  switchport mode trunk

Switch(config)#  interface gi2/24
Switch(config-if)#  switchport trunk encapsulation dot1q
Switch(config-if)#  switchport mode trunk
```

Important note: Both sides of the trunk must be configured with the *same* tagging protocol. Otherwise, a trunk connection will not form.

If the switch only supports 802.1Q, the *switchport trunk encapsulation* command will not be available.

* * *

Configuring Trunk Links (continued)

The switch can *negotiate* the tagging protocol, using DTP:

```
Switch(config)#  interface gi2/24
Switch(config-if)#  switchport trunk encapsulation negotiate
Switch(config-if)#  switchport mode trunk
```

The tagging protocol that is supported by *both* switches will be used. If the switches support both ISL and 802.1Q, **ISL** will be the preferred protocol.

By default, **all active VLANs** are allowed to traverse a trunk link. While this is convenient, a good security practice is to allow only necessary VLANs over a trunk.

To explicitly *allow* a subset of VLANs on a trunk port:

```
Switch(config)#  interface gi2/24
Switch(config-if)#  switchport trunk allowed vlan 3,9,11-15
```

The above command will force the trunk link to only forward traffic from VLANS *3, 9,* and *11 – 15.* To *remove* a VLAN from the allowed list:

```
Switch(config)#  interface gi2/24
Switch(config-if)#  switchport trunk allowed vlan remove 12
```

To *add* a specific VLAN back into the allowed list:

```
Switch(config)#  interface gi2/24
Switch(config-if)#  switchport trunk allowed vlan add 25
```

Important Note: It is common to restrict the allowed VLANs on a trunk link, and then add to the allowed list as new VLANs are created. **However**, don't forget to use the *add* parameter. If *add* is omitted, the command will *replace* the list of allowed VLANs on the trunk link, to the great distress of network admins everywhere (sorry Karl). Always remember to *add*! ☺

To allow all VLANs *except* for a specific range:

```
Switch(config-if)#  switchport trunk allowed vlan except 50-99
```

To allow *all* VLANs again:

```
Switch(config-if)#  switchport trunk allowed vlan all
```

* * *

Native VLANs

Recall that a trunk port *tags* frames with a VLAN ID. But what happens if a trunk port receives an *untagged* frame?

The **native VLAN** determines the VLAN that untagged traffic belongs to. By default on all trunking ports, the native VLAN is **VLAN 1**. The native VLAN can be changed on a per trunk port basis:

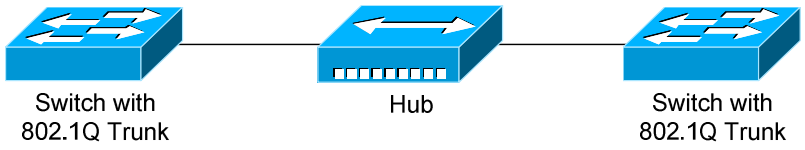
```
Switch(config)#  interface gi2/24
Switch(config-if)#  switchport mode trunk
Switch(config-if)#  switchport trunk native vlan 42
```

Only one native VLAN can be assigned to a trunk port. All untagged traffic *received* on this port will become a member of the native VLAN. Additionally, frames belonging to the native VLAN are not tagged when being *sent* out a trunk port.

Native VLANS are only supported on **802.1Q** trunk ports**.** ISL does not support untagged frames, and will *always* tag frames from *all* VLANs.

The native VLAN must be configured **identically on both sides of the 802.1Q trunk,** otherwise the switches will not form a trunk connection.

The original intent of native VLANs was for legacy compatibility with hubs. Consider the following deprecated example:



The hub has no knowledge of VLANs or 802.1Q. Traffic from hosts connected to the hub will be forwarded to the switches untagged, which in turn will place the untagged traffic into the native VLAN.

Native VLANs pose a **security risk**, allowing an attacker to *hop* to another VLAN by *double-tagging* a frame. This can be mitigated by changing the native VLAN to an unused or disabled VLAN. A better solution is to force trunk ports to tag native VLAN traffic - globally or on a per-trunk basis:

```
Switch(config)#  vlan dot1q tag native

Switch(config)#  interface gi2/24
Switch(config-if)#  switchport trunk native vlan tag
```

* * *

Dynamic Trunking Protocol (DTP)

Recall that a trunk's frame tagging protocol can be autonegotiated, through the use of the **Dynamic Trunking Protocol (DTP)**.

DTP can also negotiate **whether a port becomes a trunk at all**. Previous examples demonstrated how to manually configure a port to trunk:

```
Switch(config)# interface gi2/24
Switch(config-if)# switchport mode trunk
```

DTP has two modes to *dynamically* decide whether a port becomes a trunk:

- **Desirable** – the port will *actively* attempt to form a trunk with the remote switch. This is the default setting.
- **Auto** – the port will *passively* wait for the remote switch to initiate the trunk.

To configure the DTP mode on an interface:

```
Switch(config)# interface gi2/24
Switch(config-if)# switchport mode dynamic desirable
Switch(config-if)# switchport mode dynamic auto
```

Trunk ports send out DTP frames **every 30 seconds** to indicate their configured mode.

A trunk ***will form*** in the following configurations:

- manual trunk ←→ manual trunk
- manual trunk ← → dynamic desirable
- manual trunk ← → dynamic auto
- dynamic desirable ← → dynamic desirable
- dynamic desirable ← → dynamic auto

A trunk ***will never form*** if the two sides of the trunk are set to dynamic auto, as both ports are waiting for the other to initialize the trunk.

It is best practice to manually configure trunk ports, to avoid DTP negotiation errors. DTP is also vulnerable to VLAN spoofing attacks.

To explicitly disable DTP:

```
Switch(config)# interface gi2/24
Switch(config-if)# switchport mode trunk
Switch(config-if)# switchport nonegotiate
```

* * *

Troubleshooting Trunk Connections

A trunk connection requires several parameters to be configured identically on both sides of the trunk:

- **Trunk Mode**
- **Frame-tagging protocol**
- **Native VLAN**
- **Allowed VLANs**
- **VTP Domain** – only when using DTP to negotiate a trunk

If there is a mismatch in configuration, the trunk connection will never become active.

To determine whether an interface is an access or trunk port:

Switch# *show interface gi2/24 switchport*

```
Name: Gi2/24
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
Administrative Trunking Encapsulation: dot1q
Operational Trunking Encapsulation: dot1q
Negotiation of Trunking: Off
Access Mode VLAN: 1 (default)
Trunking Native Mode VLAN: 42

<snip>
```

To view the status of all trunk links:

Switch# *show interface trunk*

```
Port        Mode         Encapsulation      Status       Native VLAN
Fa0/24      on           802.1q             trunking     42

Port        Vlans allowed on trunk
Fa0/24      3,9,11-15

Port        Vlans allowed and active in management domain
Fa0/24      3,9

Port        Vlans in spanning tree forwarding state and not pruned
Fa0/24      3,9
```

Note that VLANs *11-15* are not active. Most likely, no interfaces have been assigned to those VLANs.

If there are no interfaces in an active trunking state, the *show interface trunk* command will return no output.

* * *

VLAN Trunking Protocol (VTP)

Maintaining a consistent VLAN database can be difficult in a large switching environment.

Cisco's proprietary **VLAN Trunking Protocol (VTP)** simplifies this management - updates to the VLAN database are propagated to all switches using **VTP advertisements.**

VTP requires that all participating switches join a **VTP domain**. Switches must belong to the same domain to share VLAN information, and a switch can only belong to a single domain.

VTP Versions

There are *three* versions of VTP. **VTP version 1** supports the standard 1 – 1005 VLAN range. VTP version 1 is also default on Catalyst switches.

VTP version 2 introduces some additional features:

- Token Ring support
- VLAN consistency checks
- Domain-independent transparent pass through

VTPv1 and v2 are **not compatible.** The VTP version is dictated by the **VTP server**, discussed in detail shortly. If the VTP server is configured for VTPv2, all other switches in the VTP domain will change to v2 as well.

Until recently, **VTP Version 3** was supported on only limited Cisco switch platforms. VTPv3 was built to be flexible, and can forward both VLAN and other database information, such as Multiple Spanning Tree (MST) protocol.

Other enhancements provided by VTPv3 include:

- Support for the extended 1006-4094 VLAN range.
- Support for private VLANs.
- Improved VTP authentication.
- Protection from accidental database overwrites, by using VTP primary and secondary servers.
- Ability to enable VTP on a per-port basis.

(Reference: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3560/software/release/12-2_52_se/configuration/guide/3560scg/swvtp.html)

* * *

VTP Modes

A switch using VTP must operate in one of three **modes**:

- **Server**
- **Client**
- **Transparent**

VTP servers are responsible for creating, deleting, or modifying entries in the VLAN database. Each VTP domain must have at least one VTP server, and this is the **default mode** for Cisco switches.

Servers advertise the VLAN database to all other switches in the VTP domain, including other servers. VTP servers can only advertise the standard 1-1005 VLAN range, and advertisements are only sent out **trunk** ports.

VTP clients cannot modify the VLAN database, and rely on advertisements from other switches to update VLAN information. A client will also *forward* VTP advertisements out every trunk port.

Remember: switches must be in the **same VTP Domain** to share and accept updates to the VLAN database. Only servers can change the VLAN database.

A **VTP transparent** switch maintains its own local VLAN database, and does not directly participate in the VTP domain. A transparent switch will never accept VLAN database information from another switch, even a server. Also, a transparent switch will never advertise its local VLAN database to another switch.

Transparent switches will **pass through** advertisements from other switches in the VTP domain. The VTP version dictates how the pass through is handled:

- **VTP version 1** – the transparent switch will only pass through advertisements from the *same* VTP domain.
- **VTP version 2** – the transparent switch will pass through advertisements from *any* VTP domain.

* * *

VTP Advertisements – Revision Number

Recall that updates to the VLAN database are propagated using **VTP advertisements.** VTP advertisements are always sent out **trunk ports**, on VLAN 1.

VTP advertisements are marked with a 32-bit **configuration revision number**, to identify the most current VLAN database revision. Any change to the VLAN database increments the configuration revision number by 1. Thus, a *higher* number represents a *newer* database revision.

A switch will only accept an advertisement if the revision number is *higher* than the current VLAN database. Advertisements with a *lower* revision number are ignored.

Important note: While only VTP servers can *change* the VLAN database, VTP clients can *advertise* updates, to other clients and even to a server! As long as the revision number is higher, the switch will accept the update.

This can result in a newly-introduced switch advertising a *blank* or *incorrect* VLAN database to all other switches in the domain. Switch ports would then lose their VLAN memberships, resulting in a significant network outage.

This can be avoided when implementing a new switch into the VTP domain. Best practice is to configure a new switch as a VTP client, and reset its revision number to **zero** before deploying into a production network.

There are two methods of resetting the revision number to zero on a switch:

1. Change the VTP domain name, and then change it back to the original name.
2. Change the VTP mode to transparent, and then change it back to either server or client. Transparent switches always a revision number of 0.

VTP has fallen out of favor, due to the risk of an unintentional overwrite of the VLAN database. Until very recently, Cisco did not support VTP on the Nexus platform of switches.

VTPv3 directly addresses this risk through the use of VTP **primary** and **secondary** servers. Only the primary server is allowed to update the VLAN database on other switches. Only one primary server is allowed per domain.

(Reference: http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/solution_guide_c78_508010.html)

* * *

VTP Advertisements – Message Types

Three **message types** exist for VTP advertisements:

- **Summary Advertisement**
- **Subset Advertisement**
- **Advertisement Request**

Both VTP servers and clients will send out a **summary advertisement** every **300 seconds**. Summary advertisements contain the following information about the VTP domain:

- VTP version
- Domain name
- Configuration revision number
- Time stamp
- MD5 digest

Summary advertisements are also sent when a **change occurs** to the VLAN database. The summary is then followed with a **subset advertisement**, which actually contains the full, updated VLAN database.

A subset advertisement will contain the following information:

- VTP version
- Domain name
- Configuration revision number
- VLAN IDs for each VLAN in the database
- VLAN-specific information, such as the VLAN name and MTU

Important note: Switches will only accept summary and subset advertisements if the *domain name* and *MD5 digest* match. Otherwise, the advertisements are ignored.

If a switch receives a summary advertisement with a revision number *higher* than its own, it will send out an **advertisement request.** VTP servers will then respond with an updated summary and subset advertisement so that the switch can synchronize to the most current VLAN database.

A switch that is reset or newly joined to the VTP domain will also send out an advertisement request.

(Reference: http://www.cisco.com/c/en/us/support/docs/lan-switching/vtp/10558-21.html#vtp_msg)

* * *

Configuring VTP

By default, a switch is in VTP *server* mode, and joined to a *blank* domain labeled *NULL.*

To change the VTP *domain* name:

Switch(config)# *vtp domain MYDOMAIN*

Note that the domain name is case sensitive. To configure the VTP *mode*:

Switch(config)# *vtp mode server*
Switch(config)# *vtp mode client*
Switch(config)# *vtp mode transparent*

The VTP domain can be secured using a password:

Switch(config)# *vtp password P@SSWORD!*

The password is also case sensitive. All switches participating in the VTP domain must be configured with the same password. The password is hashed into a 16-byte MD5 digest.

Cisco switches use **VTP version 1** by default, which is not compatible with VTPv2**.** The VTP version is dictated by the **VTP server**, and if the server is configured for VTPv2, all other switches in the VTP domain will change to v2 as well.

Switch(config)# *vtp version 2*

To view status information about VTP:

Switch# *show vtp status*

```
VTP Version                     : 2
Configuration Revision          : 42
Maximum VLANs supported locally : 1005
Number of existing VLANs        : 7
VTP Operating Mode              : Server
VTP Domain Name                 : MYDOMAIN
VTP Pruning Mode                : Disabled
VTP V2 Mode                     : Enabled
VTP Traps Generation            : Disabled
MD5 digest                      : 0x42 0x51 0x69 0xBA 0xBE 0xFA 0xCE 0x34
Configuration last modified by 0.0.0.0 at 6-22-14 4:07:52
```

To view VTP statistical information and error counters:

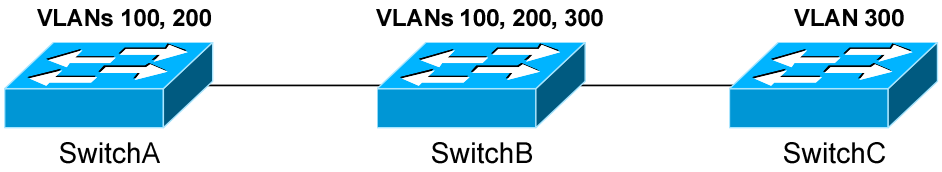
Switch# *show vtp counters*

* * *

VTP Pruning

Recall that Layer-2 switches belong to only *one* **broadcast domain.** A Layer-2 switch will thus forward both broadcasts and multicasts out *every* port in the same VLAN but the originating port. This includes sending out broadcasts out trunk ports to *other* switches, which will in turn flood that broadcast out all ports in the same VLAN.

VTP pruning eliminates unnecessary broadcast or multicast traffic throughout the switching infrastructure. Consider the following example:



Assume that a host is connected to SwitchB, in VLAN 300. If the host sends out a broadcast, SwitchB will forward the broadcast out every port in VLAN 300, including the trunk ports to SwitchA and SwitchC. Both SwitchA and SwitchC will then forward that broadcast out every port in VLAN 300.

However, SwitchA does not have any ports in VLAN 300, and will drop the broadcast. Thus, sending the broadcast to SwitchA is a waste of bandwidth.

VTP pruning allows a switch to learn which VLANs are *active* on its neighbors. Thus, broadcasts are only sent out the necessary trunk ports where those VLANs exist. In the preceding example, pruning would prevent VLAN 300 broadcasts from being sent to SwitchA, and would prevent VLAN 100 and 200 broadcasts from being sent to SwitchC.

VTP pruning is **disabled by default** on IOS switches. VTP pruning must be enabled on a server, and will be applied globally to the entire VTP domain:

```
Switch(config)# vtp pruning
```

Both VLAN 1 and the system VLANs 1002-1005 are never eligible for pruning. To manually specify which VLANs are pruning eligible on a trunk:

```
Switch(config)# interface gi2/24
Switch(config-if)# switchport trunk pruning vlan 2-10
Switch(config-if)# switchport trunk pruning vlan add 42
Switch(config-if)# switchport trunk pruning vlan remove 5
Switch(config-if)# switchport trunk pruning vlan except 100-200
Switch(config-if)# switchport trunk pruning vlan none
```

* * *

Section 8
- EtherChannel -

Port Aggregation

A network will often span across multiple switches. Trunk ports are usually used to connect switches together.

There are two issues with using only a *single* physical port for the trunk connection:

- The port represents a **single point of failure.** If the port goes down, the trunk connection is lost.
- The port represents a traffic **bottleneck**. All other ports on the switch will use that one port to communicate across the trunk connection.

Thus, the obvious benefits of adding redundancy to the trunk connection are **fault tolerance** and **increased bandwidth**, via load balancing.

However, simply trunking two or more ports between the switches will not work, as this creates a **switching loop**. One of two things will occur:

- Spanning Tree Protocol (STP) will disable one or more ports to eliminate the loop.
- If STP is disabled, the switching loop will result in an almost instantaneous broadcast storm, crippling the network.

Port aggregation allows multiple *physical* ports to be bundled together to form a single *logical* port. The switch and STP will treat the bundled ports as a single interface, eliminating the possibility of a switching loop.

Cisco's implementation of port aggregation is called **EtherChannel.** EtherChannel supports **Fast**, **Gigabit**, and **10 Gigabit** Ethernet ports.

A maximum of **8 active ports** are supported in a single EtherChannel. If the ports are operating in full duplex, the maximum *theoretical* bandwidth supported is as follows:

- Fast Ethernet – **1600 Mbps**
- Gigabit Ethernet – **16 Gbps**
- 10 Gigabit Ethernet – **160 Gbps**

The maximum number of supported EtherChannels on a single switch is platform-dependent, though most support up to **64** or **128** EtherChannels.

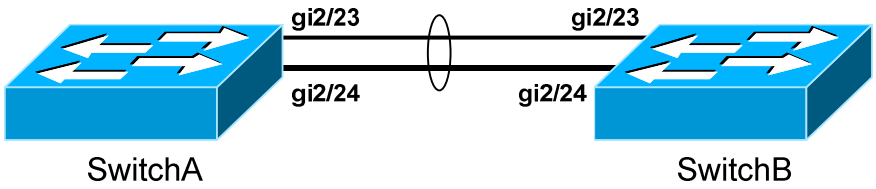
* * *

EtherChannel Requirements

The previous section described the benefits of port aggregation for a trunk connection. However, EtherChannels can be formed with *either* access or trunk ports. EtherChannels are also supported on Layer-3 interfaces.

Implementing an EtherChannel for access ports provides increased bandwidth and redundancy to a host device, such as a server. However, the host device must support a **port aggregation protocol**, such as LACP. Port aggregation protocols are covered in great detail later in this guide.

Similarly, implementing EtherChannel for trunk connections provides increased bandwidth and redundancy to other switches.



If a port in an EtherChannel bundle fails, traffic will be redistributed across the remaining ports in the bundle. This happens nearly *instantaneously*.

For an EtherChannel to become active, all ports in the bundle **must be configured identically**, regardless if the EtherChannel is being used with access or trunk ports. Port settings that must be identical include the following:

- **Speed** settings
- **Duplex** settings
- **STP** settings
- **VLAN membership** (for access ports)
- **Native VLAN** (for trunk ports)
- **Allowed VLANs** (for trunk ports)
- **Trunking encapsulation protocol** (for trunk ports)

On trunk connections, the above settings must be configured identically across all participating ports on *both* switches.

Historically, **port-security** has not been supported on an EtherChannel. Newer platforms may provide support as long as port-security is enabled on both the physical interfaces and the EtherChannel itself.

* * *

EtherChannel Load-Balancing

Traffic sent across an EtherChannel is *not* evenly distributed across all ports in the bundle. Instead, EtherChannel utilizes a load-balancing algorithm to determine the port to send the traffic out, based on one of several criteria:

- Source IP address - **src-ip**
- Destination IP address - **dst-ip**
- Source *and* destination IP address - **src-dst-ip**
- Source MAC address - **src-mac**
- Destination MAC address - **dst-mac**
- Source *and* Destination MAC address - **src-dst-mac**
- Source TCP/UDP port number - **src-port**
- Destination TCP/UDP port number - **dst-port**
- Source *and* destination port number - **src-dst-port**

Using a deterministic algorithm prevents perfect load-balancing. However, a particular traffic flow is forced to always use the same port in the bundle, preventing out-of-order delivery.

The default load-balancing method for a Layer-2 EtherChannel is either **src-mac** or **src-dst-mac**, depending on the platform. The default method for a Layer-3 EtherChannel is **src-dst-ip**.

The load-balancing method must be configured *globally* on the switch:

Switch(config)# *port-channel load-balance src-dst-mac*

To display the currently configured load-balancing method:

Switch# *show etherchannel load-balance*

```
EtherChannel Load-balancing Configuration:
       src-dst-mac
```

To view the load on each port in an EtherChannel (output abbreviated):

Switch# *show etherchannel 1 port-channel*

```
Index   Load   Port     EC state
------+------+--------+------------
  0     55     Gi2/23   active
  1     3A     Gi2/24   active
```

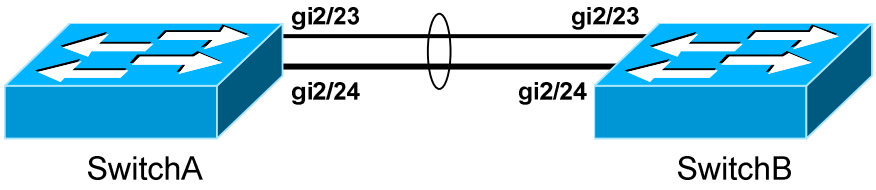
The load is rather cryptically represented in a hexadecimal value.

(Reference: http://www.cisco.com/c/en/us/support/docs/lan-switching/etherchannel/12023-4.html)

* * *

EtherChannel Load-Balancing Example

Consider the following example, where ports *gi2/23* and *gi2/24* on both switches are members of an EtherChannel:



Assume that the load-balancing method is **src-ip.** The first port in the EtherChannel will become Link0; the second will become Link1.

The two links in the EtherChannel can be represented in one binary **bit**. The load-balancing algorithm will create an index that associates Link0 with a binary bit of *0,* and Link1 with a bit of *1*.

As traffic passes through the EtherChannel, the algorithm will convert the source IP addresses into a binary hash, to compare against the index. For example, consider the following IP addresses and their binary equivalents:

10.1.1.2	00001010.00000001.00000001.00000010
10.1.1.3	00001010.00000001.00000001.00000011

Because there are only two ports in the EtherChannel, only one bit needs to be considered in the IP address – **the last bit.** The first address ends with a *0* bit, and thus would be sent out *Link0*. The second address ends with a *1* bit, and thus would be sent down *Link1*. Simple, right?

An EtherChannel that contained four ports would require a 2-bit index, requiring that the *last two* bits of the IP address be considered:

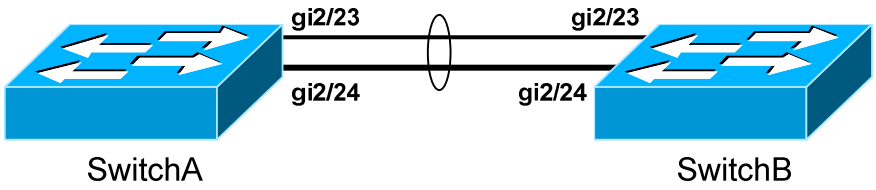
Link0	00
Link1	01
Link2	10
Link3	11

Best practice dictates that EtherChannels should contain an **even** number of ports, to promote uniform load-balancing. An EtherChannel can support an odd number of ports; however, this may result in one of the ports being severely overburdened due to uneven load-balancing.

* * *

EtherChannel Load-Balancing Example (continued)

Consider again the following example:



This time, assume that the load-balancing method is **src-dst-ip.** Again, the first port in the EtherChannel will become Link0; the second will become Link1.

Both the source *and* destination IP must be considered when choosing a link. This requires performing an **exclusive OR (XOR)** operation. In an XOR operation, if the two values being compared are *equal*, the result is *0.* If the two values are *not equal*, the result is *1*.

The following illustrates all possible results with a 1-bit index:

Source	0	1	0	1
Destination	0	0	1	1
Result	**0**	**1**	**1**	**0**

Consider the following source and destination IP address pair:

Source	192.168.1.10	11000000.10101000.00000001.00001010
Destination	192.168.2.25	11000000.10101000.00000010.00011001

The XOR result of the above address pair would be *1*. Thus, the traffic would be sent out *Link1*.

The following illustrates all possible results with a 2-bit index, representing four ports:

Source	00	00	00	00	01	01	01	01	10	10	10	10	11	11	11	11
Destination	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
Result	**00**	**01**	**10**	**11**	**01**	**00**	**11**	**10**	**10**	**11**	**00**	**01**	**11**	**10**	**01**	**00**

There are four possible results (*00, 01, 10, 11*), corresponding to the four ports in the EtherChannel.

Regardless of the load-balancing method used, **STP** will always send its packets through the **first operational port** in an EtherChannel bundle.

* * *

EtherChannel – Manual Configuration

There are two methods of configuring an EtherChannel:

- **Manually**
- **Dynamically**, using an aggregation protocol

To *manually* configure two ports to join an EtherChannel:

Switch(config)# *interface range gi2/23 - 24*
Switch(config-if)# *channel-group 1 mode on*

The remote switch must also have the EtherChannel manually configured as ***on***. Remember that speed, duplex, VLAN, and STP configuration must be configured identically across all participating ports on *both* switches.

The *channel-group* number identifies the EtherChannel on the local switch. This number does not need to match on both switches, though for documentation purposes it *should*.

Adding switch ports to a channel-group creates a logical **port-channel interface**. This interface can be configured by referencing the *channel-group* number:

Switch(config)# *interface port-channel 1*
Switch(config-if)#

Changes made to the logical port-channel interface are applied to *all* physical switch ports in the channel-group:

Switch(config)# *interface port-channel 1*
Switch(config-if)# *switchport mode trunk*
Switch(config-if)# *switchport trunk allowed vlan 50-100*

To configure a port-channel as a Layer-3 interface:

Switch(config)# *interface port-channel 1*
Switch(config-if)# *no switchport*
Switch(config-if)# *ip address 192.168.10.1 255.255.255.0*

By default, a port-channel interface is *administratively shutdown*. To bring the port-channel online:

Switch(config)# *interface port-channel 1*
Switch(config-if)# *no shut*

Physical port properties, such as speed and duplex, must be configured on the physical interface, and not on the port-channel interface.

* * *

EtherChannel – Dynamic Configuration

Cisco switches support two dynamic aggregation protocols:

- **PAgP (Port Aggregation Protocol) –** Cisco proprietary aggregating protocol.
- **LACP (Link Aggregation Control Protocol)** – IEEE standardized aggregation protocol, originally defined in 802.3ad.

Both PAgP and LACP exchange **negotiation packets** to form the EtherChannel. When an EtherChannel is configured *manually*, no negotiation packets are exchanged.

Thus, an EtherChannel **will never** form if one switch manually configured the EtherChannel, and the other switch is using a dynamic aggregation protocol.

PAgP and LACP are **not compatible –** both sides of an EtherChannel must use the *same* aggregation protocol.

EthernChannel - PAgP

PAgP is a Cisco-proprietary aggregation protocol, and supports two modes:

- **Desirable** – actively attempts to form a channel
- **Auto** – waits for the remote switch to initiate the channel

A PAgP channel will form in the following configurations:

- desirable ←→ desirable
- desirable ←→ auto

A channel **will not form** if both sides are set to **auto.** Also, PAgP will not form a channel if the remote side is running LACP, or manually configured.

To create an EtherChannel using PAgP negotiation:

```
Switch(config)#  interface range gi2/23 – 24
Switch(config-if)#  channel-protocol pagp
Switch(config-if)#  channel-group 1 mode desirable

Switch(config-if)#  channel-group 1 mode auto
```

PAgP requires that speed, duplex, VLAN, and STP configuration be configured identically across all participating ports.

* * *

EtherChannel - LACP

LACP is an IEEE standard aggregation protocol, and supports two modes:

- **Active** – actively attempts to form a channel
- **Passive** – waits for the remote switch to initiate the channel

An LACP channel will form in the following configurations:

- active ←→ active
- active ←→ passive

A channel **will not form** if both sides are set to **passive.** Also, LACP will not form a channel if the remote side is running PAgP, or manually configured.

To create an EtherChannel using LACP negotiation:

```
Switch(config)#  interface range gi2/23 – 24
Switch(config-if)#  channel-protocol lacp
Switch(config-if)#  channel-group 1 mode active

Switch(config-if)#  channel-group 1 mode passive
```

LACP requires that speed, duplex, VLAN, and STP configuration be configured identically across all participating ports.

Recall that a maximum of **8 active ports** are supported in a single EtherChannel. LACP supports adding an *additional* 8 ports into the bundle in a **standby** state, to replace an active port if it goes down.

LACP assigns a numerical *port-priority* to each port, to determine which ports become active in the EtherChannel. By default, the priority is set to **32768**, and a **lower** priority is preferred. If there is a tie in *port-priority*, the lowest port number is preferred.

To change the LACP *port-priority* to something other than default:

```
Switch(config)#  interface range gi2/23 – 24
Switch(config-if)#  lacp port-priority 100
```

LACP also assigns a *system-priority* to each switch, dictated which switch becomes the decision-maker if there is a conflict about active ports. The default system-priority is **32768,** and a **lower** priority is again preferred. If there is a tie in *system-priority,* the lowest switch MAC address is preferred.

To globally change the *system-priority* on a switch:

```
Switch(config)#  lacp system-priority 500
```

* * *

Troubleshooting EtherChannel

To view status information on all configured EtherChannels:

Switch# *show etherchannel summary*

```
Flags:      D - down        P - in port-channel
            I - stand-alone s - suspended
            R - Layer3      S - Layer2
            U - port-channel in use

Group       Port-channel      Ports
----------- --------------- ---------------
1           Po1(SU)           Gi2/23(P) Gi2/24(P)
```

Note that both ports have a status of *P*, which indicates that they are up and active in the EtherChannel.

On Cisco Nexus switches, the syntax for this command is slightly different:

NexusSwitch# *show port-channel summary*

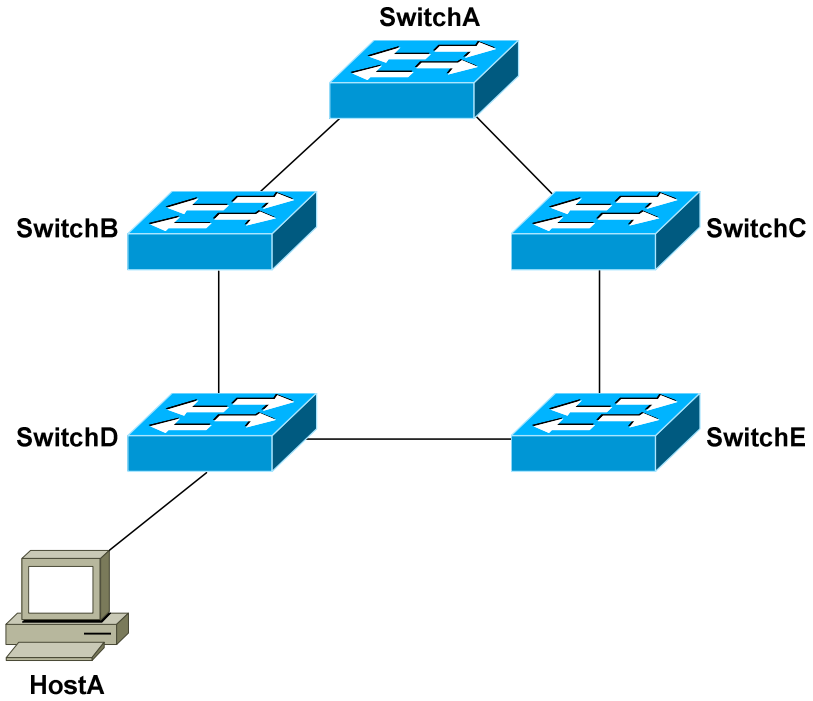
Section 9
- Spanning Tree Protocol -

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

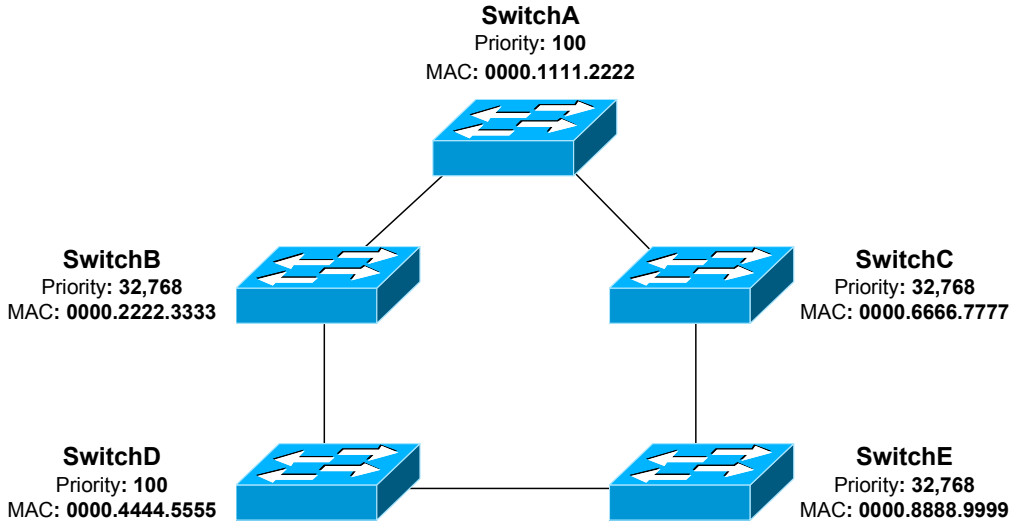
The first step in the STP convergence process is electing a **Root Bridge**, which is the central reference point for the STP topology. As a best practice, the Root Bridge should be the most centralized switch in the STP topology.

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

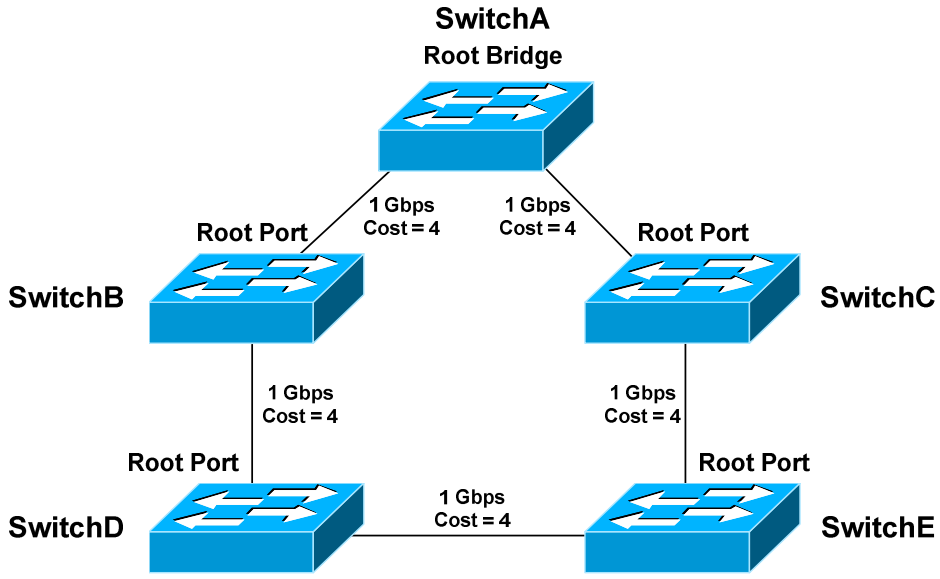
The second step in the STP convergence process is to identify **root ports.** The root port of each switch has the lowest **root path cost** to get to the Root Bridge.

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:



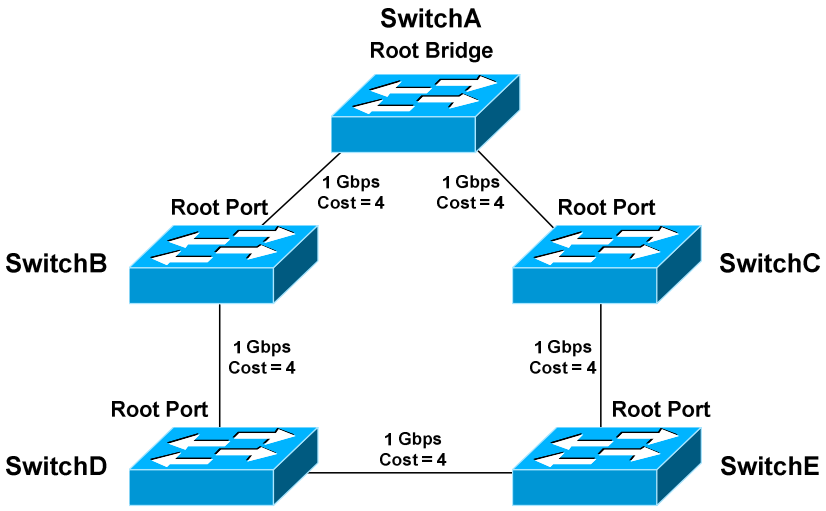
Each 1Gbps link has a path cost of *4*. SwitchA has a cumulative path cost of *0*, because it is the Root Bridge. Thus, when SwitchA sends out BPDU's, it advertises a root path cost of 0.

* * *

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

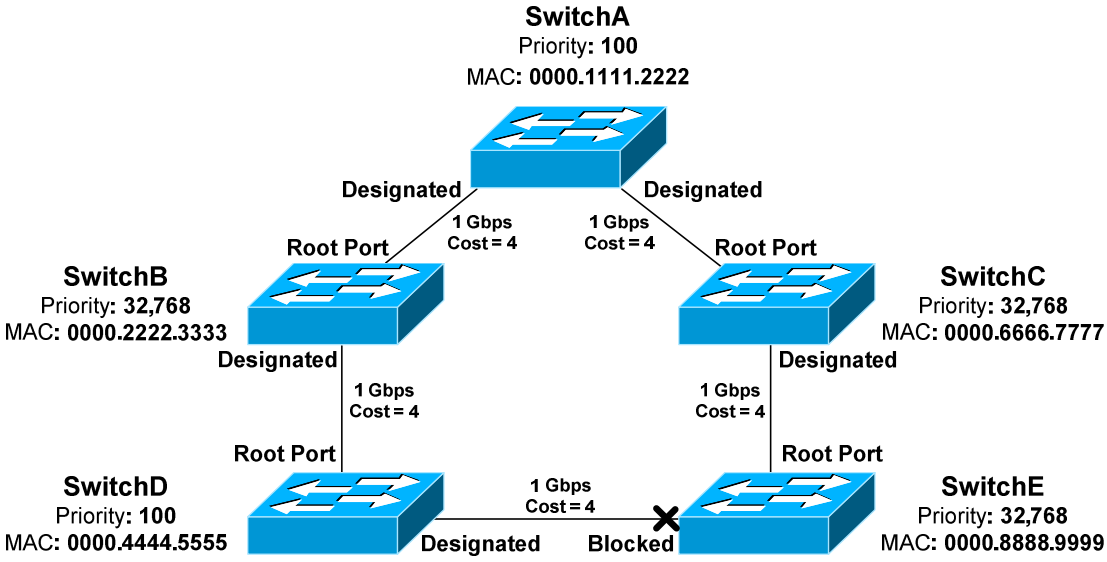
The third step in the STP convergence process is to identify **designated ports.** A single designated port is identified for each network *segment.* This port is responsible for forwarding BPDUs and frames to that segment.

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.

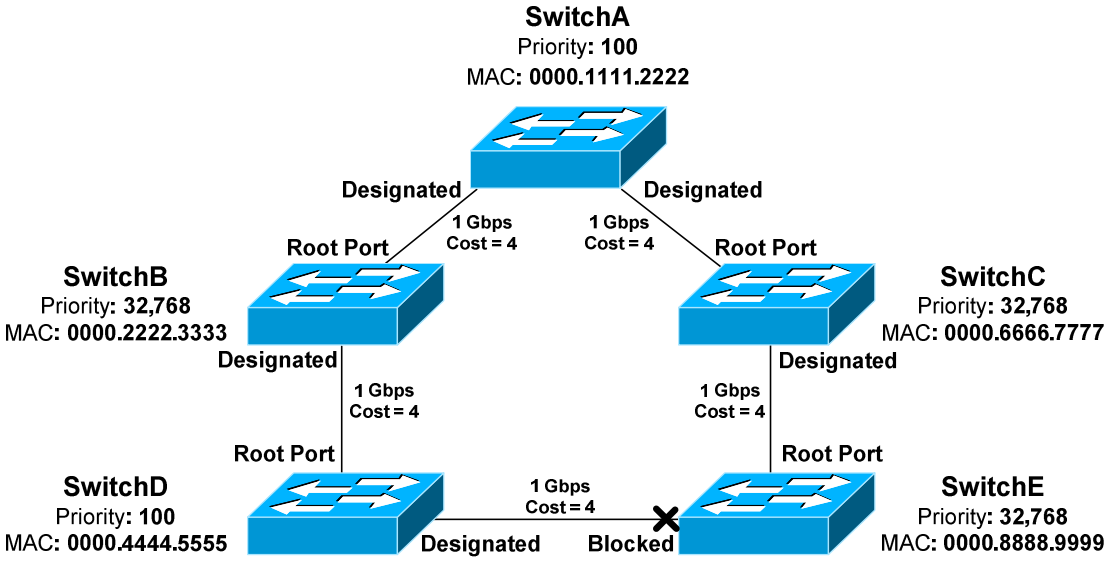
Remember, every network segment **must have one designated port**, regardless if a root port already exists on that segment.

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.

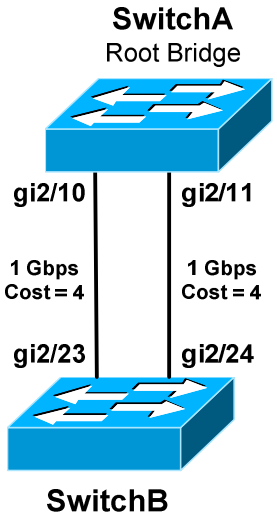
As with electing the Root Bridge, if there is a tie in priority, the **lowest MAC address** is used as the tie breaker.

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
```

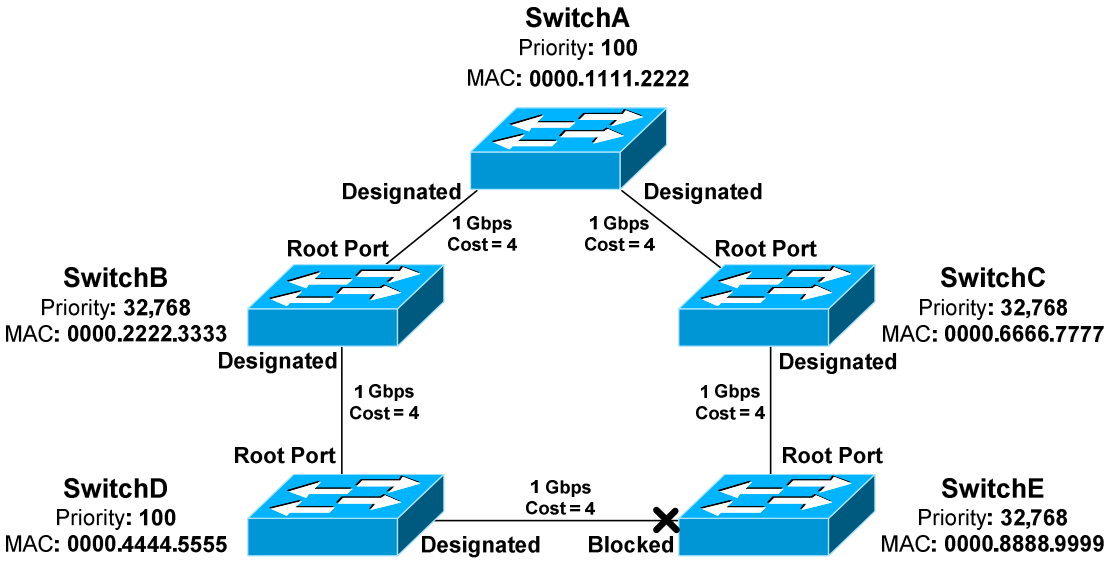
* * *

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.

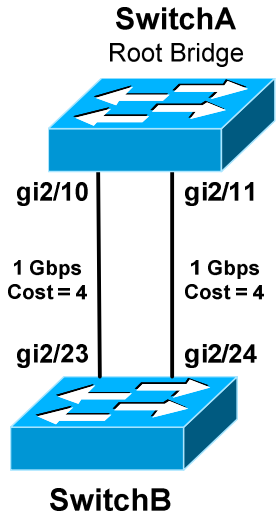
* * *

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*

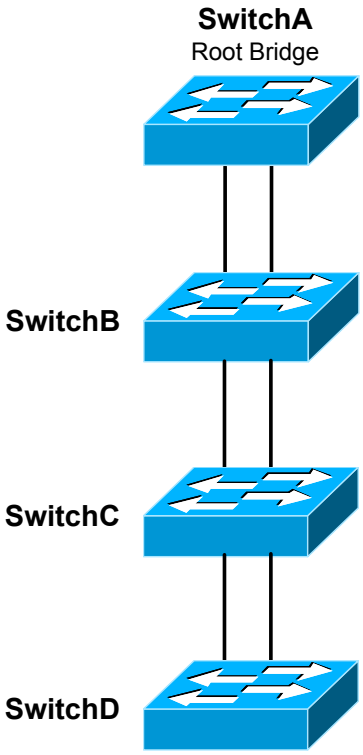
The timer values are measured in seconds, and the above represents the *maximum* possible value for each timer.

* * *

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**

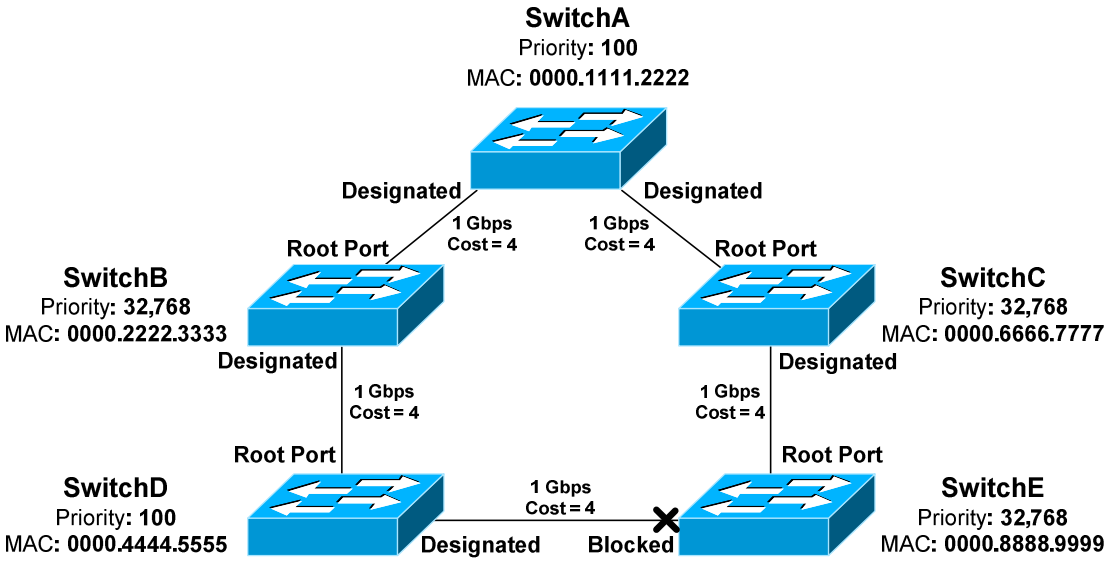
Configuration BPDUs are used to elect Root Bridges, root ports, and designated ports.

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:



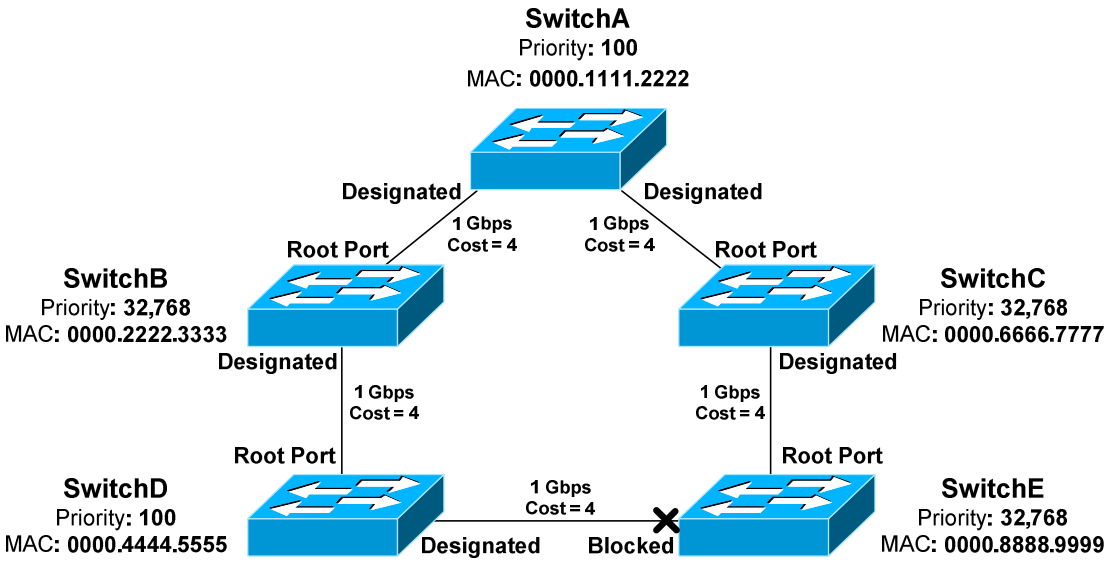
If the port on SwitchD connecting to SwitchE went down:

- 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

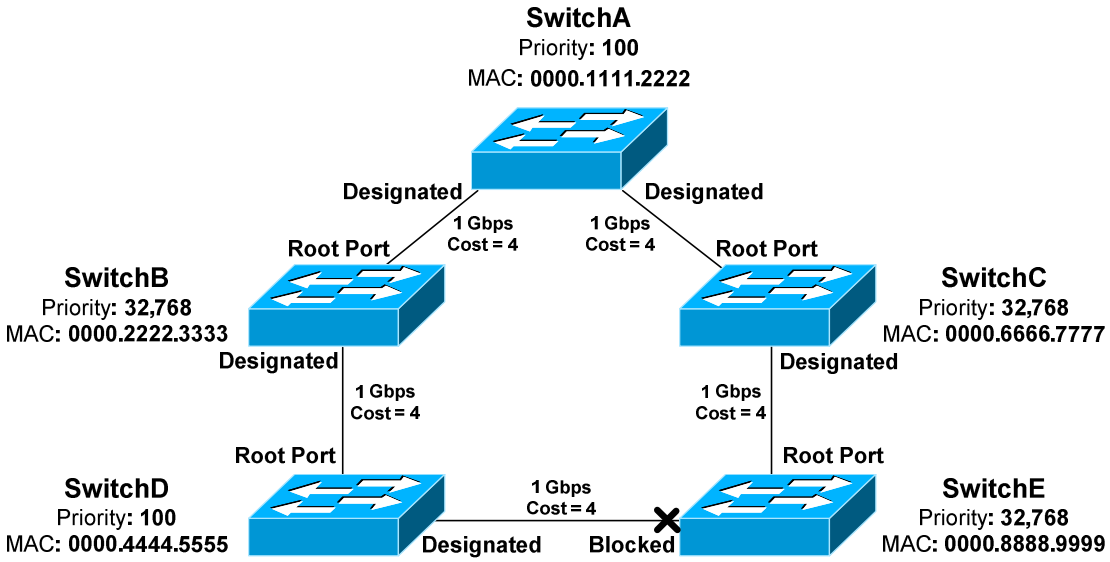
For example, if the root port on SwitchB fails, SwitchB would consider this a **direct failure.** SwitchB will detect immediately that the physical port is down, and STP will react accordingly.

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.

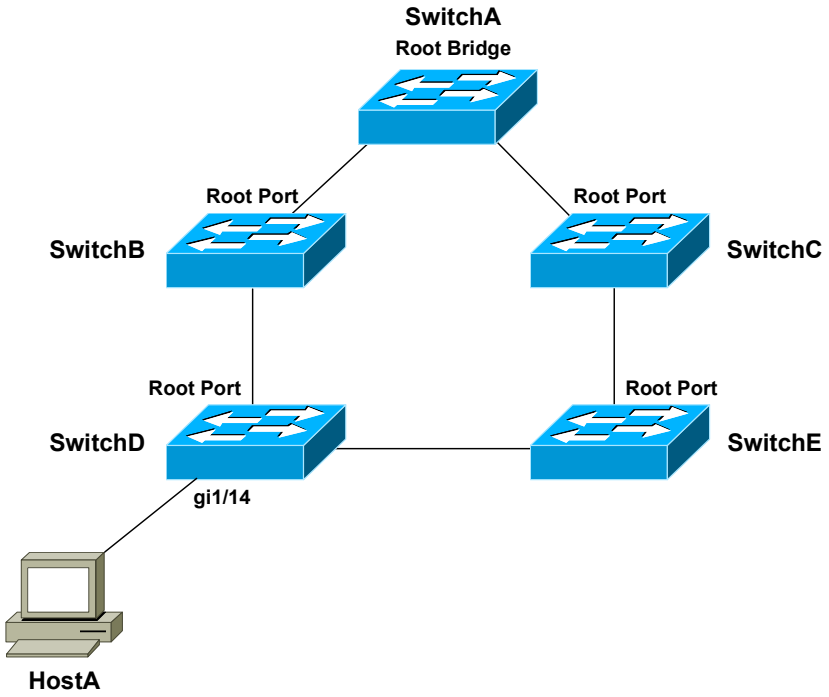
PortFast allows a switch port to bypass the usual progression of STP states. The port will instead transition from a *blocking* to a *forwarding* state **immediately**, eliminating the typical 30 second delay.

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*

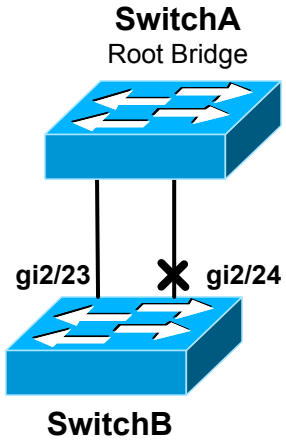
PortFast can also be globally enabled for all interfaces:

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*

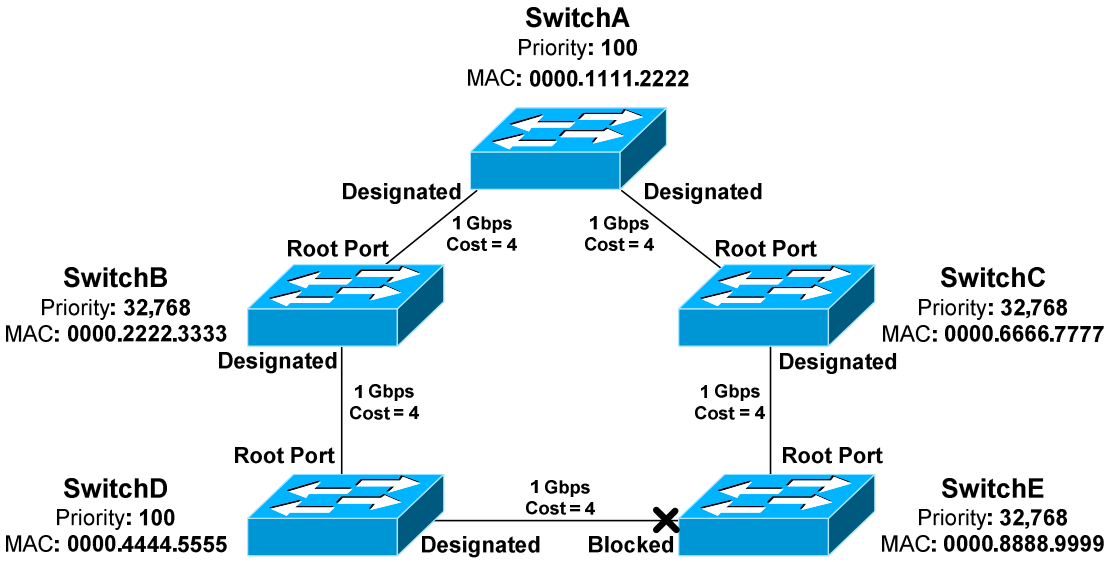
UplinkFast functions by tracking all possible links *to* the Root Bridge. Thus, UplinkFast is **not supported** on the Root Bridge. In fact, enabling this feature will automatically increase a switch's bridge priority to 49,152.

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*

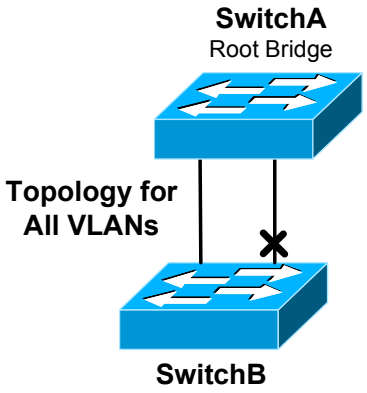
```
Vlan                 Role Sts Cost      Prio.Nbr Type
-------------------- ---- --- --------- -------- -----------------
VLAN0101             Root FWD 4         128.48   P2p
VLAN0102             Altn BLK 4         128.48   P2p
```

* * *

Per-VLAN Spanning Tree (PVST) Load Balancing

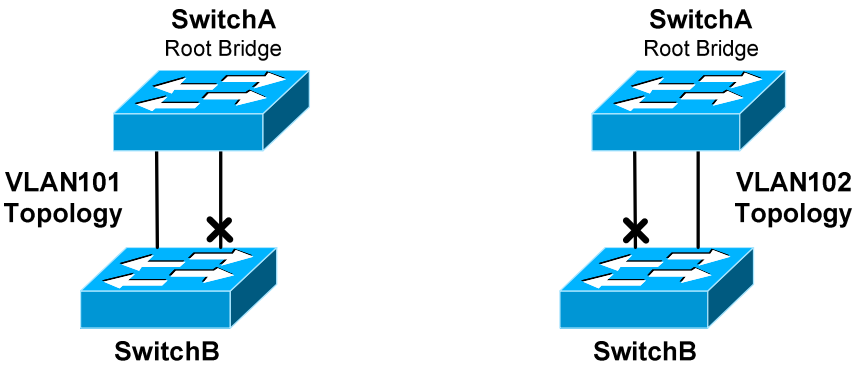
PVST and **PVST+** employ a *separate* STP instance for *each* VLAN. This provides superior flexibility over CST, which only supports a single STP instance for *all* VLANs.

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*

* * *

Section 10
- Multilayer Switching -

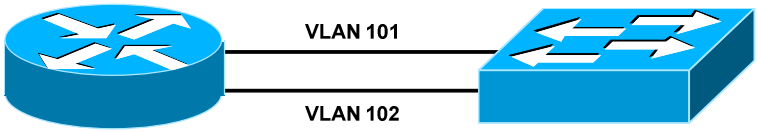
Routing Between VLANs

By default, a switch will forward both broadcasts and multicasts out *every* port but the originating port. However, a switch can be *logically* segmented into separate broadcast domains, using **Virtual LANs** (or **VLANs**).

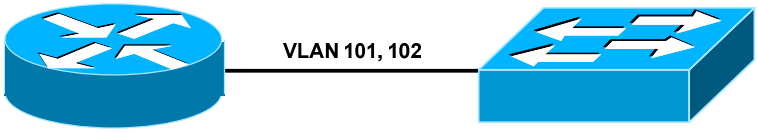
Each VLAN represents a unique broadcast domain:

- Traffic between devices within the *same* VLAN is switched.
- Traffic between devices in *different* VLANs requires a Layer-3 device to communicate.

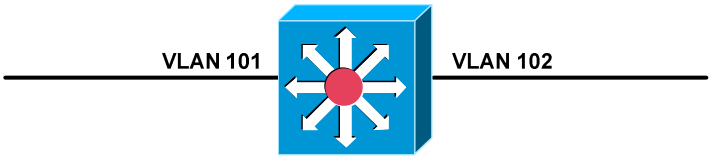
There are *three* methods of routing between VLANs. The first method involves using an **external router** with a separate physical interface **in each VLAN.** This is the *least scalable* solution, and impractical for environments with a large number of VLANs:



The second method involves using an **external router** with a single **trunk link** to the switch, over which all VLANs can be routed. The router must support either 802.1Q or ISL encapsulation. This method is known as **router-on-a-stick:**



The final method involves using a **multilayer switch,** which supports both Layer-2 and Layer-3 forwarding:

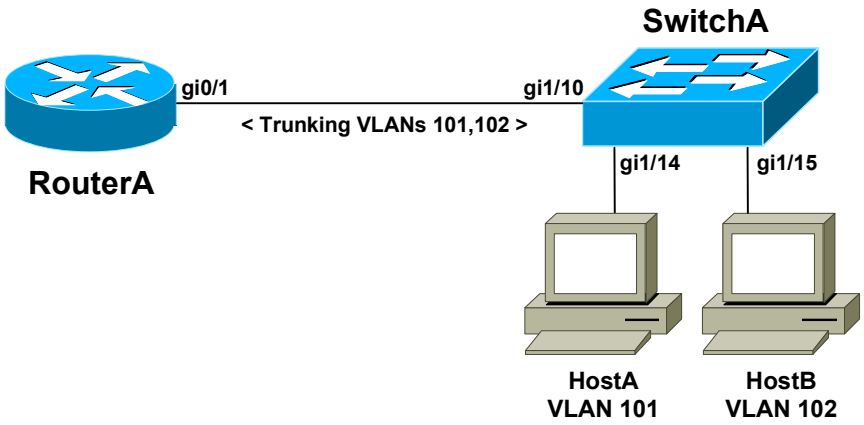


Multilayer switching is a generic term, encompassing any switch that can forward traffic at layers higher than Layer-2.

* * *

Configuring Router-on-a-Stick

Consider the following router-on-a-stick example:



Four elements must be considered in this scenario:

- Interface gi1/10 on SwitchA must be configured as a **trunk port**.
- Interfaces gi1/14 and gi1/15 on SwitchB must be assigned to their specified VLANs.
- Interface gi0/1 on RouterA must be split into **subinterfaces,** one for each VLAN.
- Each subinterface must support the **encapsulation protocol** used by the trunk port on SwitchA.

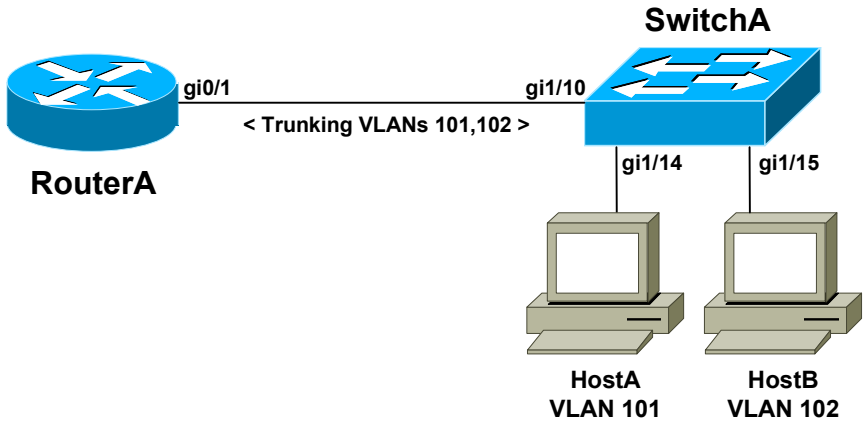
Configuration on **SwitchA** would be as follows:

```
SwitchA(config)# interface gi1/10
SwitchA(config-if)# switchport mode trunk
SwitchA(config-if)# switchport trunk encapsulation dot1q
SwitchA(config-if)# no shut

SwitchA(config)# interface gi1/14
SwitchA(config-if)# switchport mode access
SwitchA(config-if)# switchport access vlan 101
SwitchA(config-if)# no shut

SwitchA(config)# interface gi1/15
SwitchA(config-if)# switchport mode access
SwitchA(config-if)# switchport access vlan 102
SwitchA(config-if)# no shut
```

Note that no Layer-3 information was configured on SwitchA.

* * *

Configuring Router-on-a-Stick (continued)



Configuration on RouterA would be as follows:

```
RouterA(config)# interface gi0/1
RouterA(config-if)# no shut

RouterA(config)# interface gi0/1.101
RouterA(config-subif)# encapsulation dot1q 101
RouterA(config-subif)# ip address 10.101.101.1 255.255.255.0

RouterA(config)# interface gi0/1.102
RouterA(config-subif)# encapsulation dot1q 102
RouterA(config-subif)# ip address 10.102.102.1 255.255.255.0
```

Each subinterface was configured with *dot1q* encapsulation, to match the configuration of SwitchA's trunk port.

Frames sent across the trunk port will be **tagged** with the VLAN ID. The number after each *encapsulation dot1q* command represents this VLAN ID. Otherwise, the router could not interpret the VLAN tag.

The IP address on each subinterface represents the **gateway** for each VLAN:

- HostA is in VLAN 101, and will use 10.101.101.1 as its gateway.
- HostB is in VLAN 102, and will use 10.102.102.1 as its gateway.
- HostA and HostB should now have full Layer-3 connectivity.

There are inherent disadvantages to router-on-a-stick:

- All routed traffic will share the same physical router interface, which represents a bottleneck.
- ISL and DOT1Q encapsulation puts an increased load on the router processor.

* * *

Multilayer Switch Port Types

Multilayer switches support both Layer-2 and Layer-3 forwarding.

Layer-2 forwarding, usually referred to as *switching*, involves decisions based on frame or *data-link* headers. Switches will build hardware address tables to intelligently forward frames.

Layer-3 forwarding, usually referred to as *routing*, involves decisions based on packet or *network* headers. Routers build routing tables to forward packets from one network to another.

A multilayer switch supports three **port types:**

- **Layer-2** or **switchports**
- **Layer-3** or **routed ports**
- **Switched Virtual Interfaces (SVIs)**

A **switchport** can either be an *access* or *trunk* port. By default on Cisco switches, all interfaces are switchports. To *manually* configure an interface as a switchport:

```
Switch(config)#  interface gi1/10
Switch(config-if)#  switchport
```

A **routed port** behaves exactly like a physical router interface, and is not associated with a VLAN. The *no switchport* command configures an interface as a routed port, allowing an IP address to be assigned:

```
Switch(config)#  interface gi1/20
Switch(config-if)#  no switchport
Switch(config-if)#  ip address 10.101.101.1 255.255.255.0
```

Multilayer switches support configuring a VLAN as a *logical* routed interface, known as a **Switched Virtual Interface (SVI).** The SVI is referenced by the VLAN number:

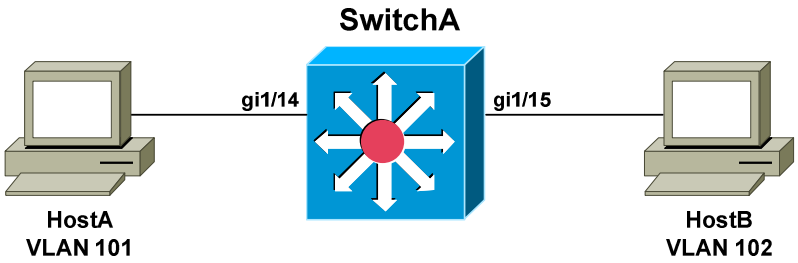
```
Switch(config)#  interface vlan 101
Switch(config-if)#  ip address 10.101.101.1 255.255.255.0
Switch(config-if)#  no shut
```

SVIs are the most common method of configuring inter-VLAN routing. The logical VLAN interface will not become online unless:

- The VLAN is **created.**
- At least **one port is active** in the VLAN.

* * *

Configuring Inter-VLAN Routing Using SVIs



Configuring inter-VLAN routing using SVIs is very simple. First, the VLANs must be created:

```
SwitchA(config)# vlan 101
SwitchA(config-vlan)# name VLAN101

SwitchA(config)# vlan 102
SwitchA(config-vlan)# name VLAN102
```

Layer-3 forwarding must then be enabled globally on the multilayer switch:

```
SwitchA(config)# ip routing
```

Finally, each VLAN SVI must be assigned an IP address:

```
SwitchA(config)# interface vlan 101
SwitchA(config-if)# ip address 10.101.101.1 255.255.255.0
SwitchA(config-if)# no shut

SwitchA(config)# interface vlan 102
SwitchA(config-if)# ip address 10.102.102.1 255.255.255.0
SwitchA(config-if)# no shut
```

The IP address on each SVI represents the **gateway** for hosts on each VLAN. The two networks will be added to the routing table as **directly connected routes.**

Remember: an SVI requires at least one port to be active in the VLAN:

```
SwitchA(config)# interface gi1/14
SwitchA(config-if)# switchport mode access
SwitchA(config-if)# switchport access vlan 101
SwitchA(config-if)# no shut

SwitchA(config)# interface gi1/15
SwitchA(config-if)# switchport mode access
SwitchA(config-if)# switchport access vlan 102
SwitchA(config-if)# no shut
```

* * *

Multilayer Switching – Route Once, Switch Many

Originally, multilayer switches consisted of two independent components:

- Routing engine
- Switching engine

The first packet in an IP traffic flow must be sent to the routing engine to be *routed*. The switching engine could then **cache** this traffic flow. Subsequent packets destined for that flow could then be *switched* instead of routed. This greatly reduced forwarding latency.

This concept is often referred to as **route *once*, switch *many*.**

Just like a router, a multilayer switch must update the following header information:

- **Layer 2 destination address**
- **Layer 2 source address**
- **Layer 3 IP Time-to-Live (TTL)**

Additionally, the Layer-2 and Layer-3 checksums must be updated to reflect the changes in header information.

Cisco's original implementation of multilayer switching was known as **NetFlow** or **route-cache switching**. NetFlow incorporated separate routing and switching engines.

NetFlow was eventually replaced with **Cisco Express Forwarding (CEF),** which addressed some of the disadvantages of NetFlow:

- CEF is less CPU intensive.
- CEF does not dynamically cache routes, eliminating the risk of stale routes in the cache if the routing topology changes.

Cisco Express Forwarding (CEF)

CEF consists of two basic components:

- **Layer-3 Engine**
- **Layer-3 Forwarding Engine –** *Switches* data based on the FIB.

The **Layer-3 Engine** is responsible for building the routing table and then *routing* traffic. The routing table can be built either *statically* or *dynamically*, via a routing protocol.

Each entry in the routing table will contain the following:

- Destination prefix
- Destination mask
- Layer-3 next-hop address

CEF reorganizes the routing table into a more efficient table called the **Forward Information Base (FIB)**. The *most specific routes* are placed at the *top* of the FIB, to accelerate forwarding lookups. Any change to the routing table is immediately reflected in the FIB.

The **Layer-3 Forwarding Engine** uses the FIB to *switch* traffic in hardware, which incurs less latency than *routing* it through the Layer-3 Engine. If a packet cannot be switched using the Forwarding Engine, it will be **punted** back to the Layer-3 Engine to be routed.

The FIB contains the **Layer-3 next-hop** for every destination network. CEF will additionally build an **Adjacency Table**, containing the **Layer-2 address** for each next-hop in the FIB. This eliminates the latency from ARP requests when forwarding traffic to the next-hop address.

If there is no next-hop entry in the Adjacency Table, a packet must be sent to the Layer-3 Engine to **glean** the next-hop's Layer-2 address, via an ARP request.

CEF is **enabled by default** on most Cisco multilayer switch platforms. In fact, it is impossible to disable CEF on many platforms.

To manually enable CEF, when necessary:

Switch(config)# *ip cef*

* * *

Cisco Express Forwarding (CEF) (continued)

CEF can be disabled on a per-interface basis on some platforms. Depending on the model, the syntax will be different:

Switch(config)# *interface gi1/15*
Switch(config-if)# *no ip route-cache cef*

Switch(config-if)# *no ip cef*

To view entries in the FIB table:

Switch# *show ip cef*

```
Prefix            Next Hop    Interface

172.16.1.0/24     10.50.1.1   Vlan50
172.16.2.0/24     10.50.1.2   Vlan50
172.16.0.0/16     10.50.1.2   Vlan50
0.0.0.0/0         10.10.1.1   Vlan10
```

The most specific entries are installed at the top of the table. Note that each FIB entry contains the following information:

- The destination prefix
- The destination mask
- The next-hop address
- The local interface where the next-hop exists

To view the Adjacency Table:

Switch# *show adjacency*

```
Protocol    Interface   Address
IP          Vlan50      10.50.1.1(6)
                        4 packets, 1337 bytes
                        0001234567891112abcdef120800
                        ARP 01:42:69

Protocol    Interface   Address
IP          Vlan50      10.50.1.2(6)
                        69 packets, 42012 bytes
                        000C765412421112abcdef120800
                        ARP 01:42:69
```

* * *

Section 11
- Switched Port Analyzer (SPAN) -

Traffic Monitoring

A common practice when troubleshooting network issues is to examine the headers and payload of packets, through the use of **packet sniffers** or **analyzers.**

A packet must first be **captured** before it can be analyzed. Packets can be captured and analyzed on a host using locally installed software. Wireshark and tcpdump are popular tools for this.

In legacy networks using hubs, *all traffic* on the network could be easily captured. A hub forwards a packet out *every port*, regardless of the destination. Thus, a single workstation with Wireshark could capture and analyze traffic between *any* two hosts.

This is no longer possible on modern networks that use switches. A packet will only be forwarded out the appropriate destination port. Thus, centrally analyzing all traffic on a network is more difficult.

Switch Port Analyzer (SPAN)

Cisco developed the **Switched Port Analyzer (SPAN)** feature to facilitate the capturing of packets. SPAN is supported on most Cisco switch platforms.

SPAN works by *copying* the traffic from one or more **source** ports. The copy is then sent out a SPAN **destination** port. The destination port will often be connected to a host running packet analyzing software, such as Wireshark.

Because SPAN only makes a *copy* of traffic, the source traffic is *never* affected. SPAN is an **out-of-band** process.

In addition to troubleshooting network issues and performance, SPAN is useful for intrusion detection systems (IDS) and application monitoring platforms.

SPAN is often referred to as *port mirroring*.

* * *

SPAN Sources and Destinations

A SPAN source is where traffic is mirrored *from.* A SPAN source can consist of one or more of the following:

- Access switchports
- Trunk ports
- Routed interfaces
- EtherChannels
- Entire VLANs

SPAN can mirror either **inbound** or **outbound** traffic on a source, or **both**.

A SPAN destination is where traffic is mirrored *to.* A SPAN destination port can consist of only a *single* switchport, and is completely dedicated for that purpose.

No other traffic is forwarded to or from a SPAN destination, including management traffic such as STP and CDP. A SPAN destination does not participate in the STP topology.

A SPAN destination port can only participate in one SPAN session, and cannot be a SPAN source port.

Most Cisco platforms *do not* support an EtherChannel as a SPAN destination. For the limited models that do, the EtherChannel must be manually configured as *on* – port aggregation protocols are not supported.

The traffic from the SPAN source can exceed the bandwidth capacity of the SPAN destination port. For example, a SPAN source of an entire VLAN can easily exceed the capacity of a single Gigabit Ethernet port.

In this circumstance, some packets will be dropped at the SPAN destination. **Remember:** *source* traffic is **never affected** by SPAN.

* * *

Configuring SPAN

Configuring SPAN involves two steps:

- Identifying the SPAN source or sources
- Identifying the SPAN destination

To configure SPAN sources:

Switch(config)# *monitor session 1 source interface gi0/10 rx*
Switch(config)# *monitor session 1 source interface gi0/11 tx*
Switch(config)# *monitor session 1 source vlan 100 both*

The command syntax begins *monitor session*, and assigns it a session number. In the above example, the session number is *1*. The SPAN destination must use the *same* session number.

The above example identifies three sources:

- Inbound or *rx* traffic on port *gi0/10*
- Outbound or *tx* traffic on port *gi0/11*
- *Both* inbound and outbound traffic on VLAN 100

When specifying a *trunk* port as a source, it is possible to restrict which VLANs are mirrored:

Switch(config)# *monitor session 1 filter vlan 1-5*

To configure a SPAN destination port:

Switch(config)# *monitor session 1 destination interface gi0/15*

Remember, the session number must match between the source and destination. To disable a specific monitoring session:

Switch(config)# *no monitor session 1*

To view the status of a SPAN session:

Switch(config)# *show monitor session 1*

```
Session 1
------------
Type : local
Source Ports:
    RX Only:       gi0/10
    TX Only:       gi0/11
    Both:          None
Source VLANs:
    RX Only:       None
    TX Only:       None
    Both:          100
Destination Ports: gi0/15
```

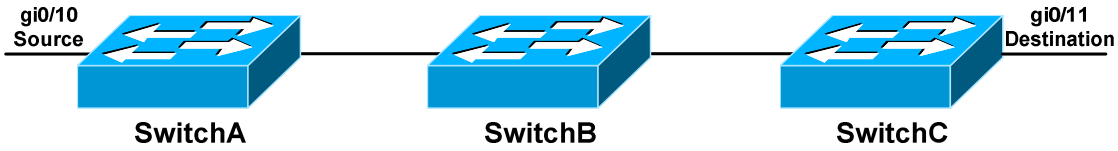
* * *

Remote SPAN (RSPAN)

The previous page describes the configuration of **Local SPAN**, where both the SPAN source and destination exist on the *same* switch.

Remote SPAN (RSPAN) allows the SPAN source and destination to exist on *different* switches. This involves configuring a RSPAN VLAN – the mirrored traffic will be carried across this VLAN from switch to switch.

Considering the following example:



The SPAN source exists on SwitchA, and the SPAN destination exists on SwitchC. Each switch must be configured with the RSPAN VLAN, including the intermediary SwitchB.

To configure RSPAN on SwitchA:

```
SwitchA(config)# vlan 200
SwitchA(config-vlan)# remote-span

SwitchA(config)# monitor session 1 source interface gi0/10
SwitchA(config)# monitor session 1 destination vlan 200
```

To configure RSPAN on SwitchB:

```
SwitchB(config)# vlan 200
SwitchB(config-vlan)# remote-span
```

To configure RSPAN on SwitchC:

```
SwitchC(config)# vlan 200
SwitchC(config-vlan)# remote-span

SwitchC(config)# monitor session 1 source vlan 200
SwitchC(config)# monitor session 1 destination interface gi0/11
```

Note that on SwitchA, the SPAN *destination* is the RSPAN VLAN, instead of a port. On SwitchC, the SPAN *source* is the RSPAN VLAN.

* * *

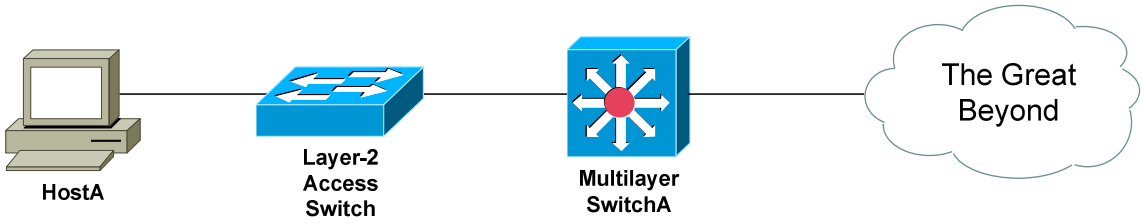
Part IV

Advanced Switch Services

Section 12
- Redundancy and Load Balancing -

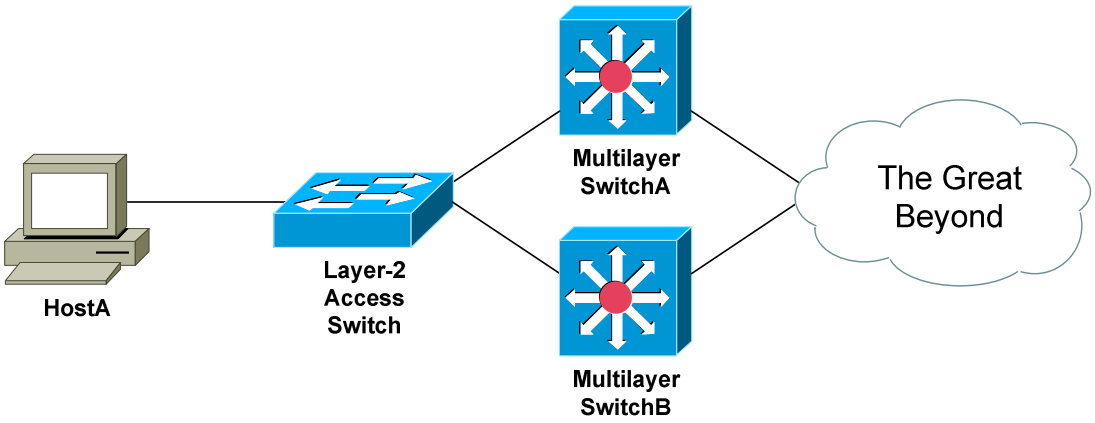
Importance of Redundancy

High availability is critical in most environments. Even a brief outage due to hardware failure may be considered unacceptable. Consider the following example:



To reach other networks, HostA must utilize a single *gateway* – SwitchA. The gateway represents a **single point of failure** on this network. If the gateway fails, hosts will lose access to *all* resources beyond the gateway.

Using multiple routers or multilayer switches can provide **Layer-3 redundancy** for hosts:



However, the Layer-3 redundancy must be **transparent** to each host. Hosts should not be configured with multiple default gateways.

Cisco supports three protocols to provide transparent Layer-3 redundancy:

- **Hot Standby Router Protocol (HSRP)**
- **Virtual Router Redundancy Protocol (VRRP)**
- **Gateway Load Balancing Protocol (GLBP)**

Note: The terms *multilayer switch* and *router* will be used interchangeably throughout this guide.

* * *

Hot Standby Router Protocol (HSRP)

Cisco developed the proprietary **Hot Standby Router Protocol (HSRP)** to allow multiple routers or multilayer switches to masquerade as a single gateway. This is accomplished by assigning a **virtual IP and MAC address** to all routers participating in an HSRP **group**.

Routers within the same HSRP group must be assigned the same **group number**, which can range from **0 to 255**. However, most Cisco platforms only support **16** configured HSRP groups.

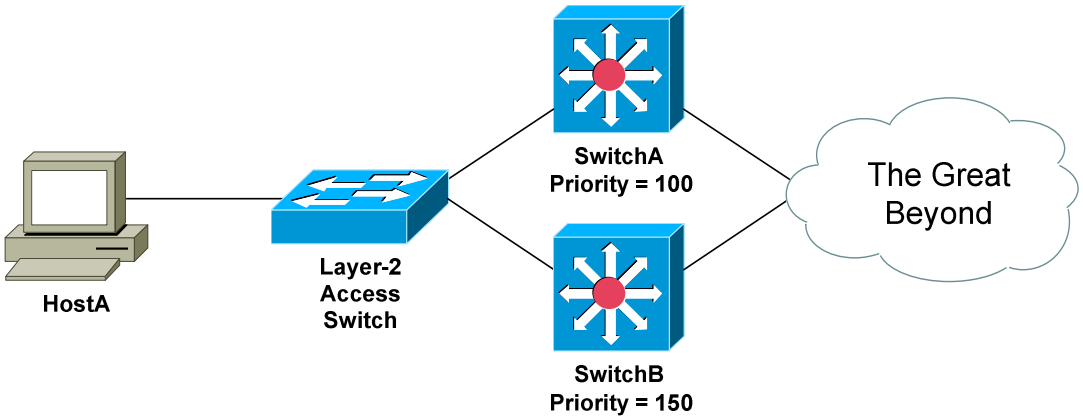
HSRP routers are elected to specific roles:

- **Active Router –** router currently serving as the gateway.
- **Standby Router –** backup router to the Active Router.
- **Listening Router –** all other routers participating in HSRP.

Only **one active** and **one standby** router are allowed per HSRP group. Thus, HSRP provides Layer-3 *redundancy*, but no inherent *load balancing*.

Hello packets are used to elect HSRP roles and to ensure all routers are functional. If the current active router fails, the standby router will immediately take over as active, and a new standby is elected. By default, hello packets are sent every **3 seconds.**

The role of an HSRP router is dictated by its **priority**. The priority can range from 0 – 255, with a default of **100.** A *higher* priority is preferred.



Thus, the router with the **highest priority** is elected the active router – *SwitchB* in the above example. The router with the **second highest priority** becomes the standby router – *SwitchA* in the example. If all priorities are equal, whichever router has the **highest IP Address** on its HSRP interface is elected the active router.

* * *

HSRP States

A router interface participating in HSRP must progress through several **states** before settling into a role:

- **Disabled**
- **Initial**
- **Learn**
- **Listen**
- **Speak**
- **Standby**
- **Active**

A **disabled state** indicates that the interface is either not configured for HSRP, or is administratively shutdown.

An interface begins in an **initial state** when first configured with HSRP, or taken out of an administratively shutdown state.

An interface enters a **learn state** if it does not know the HSRP virtual IP address. Normally the virtual IP is manually configured on the interface – otherwise, it will be learned from the current Active router via hello packets.

An interface in a **listen state** knows the virtual IP address, but was not elected as either the Active or Standby Router.

Interfaces in a **speak state** are currently participating in the election of an active or standby router. Elections are performed using hello packets, which are sent out every 3 seconds by default.

A **standby state** indicates that the interface is acting as a backup to the active router. The standby router continuously exchanges hello packets with the active router, and will take over if the active router fails.

An interface in an **active state** is the live gateway, and will forward traffic sent to the virtual IP address. Hosts will use the virtual IP address as their **default gateway**. The active router will respond to ARP requests for the virtual IP with the virtual MAC address.

Note that hello packets are only exchanged in three HSRP states:

- **Speak**
- **Standby**
- **Active**

Interfaces in a **listen** state will only *listen* for hello packets. If an active or standby router fails, a listen interface will transition to a speak state to participate in a new election.

* * *

HSRP Basic Configuration

HSRP is configured on the **interface** that is *accepting* traffic from hosts.

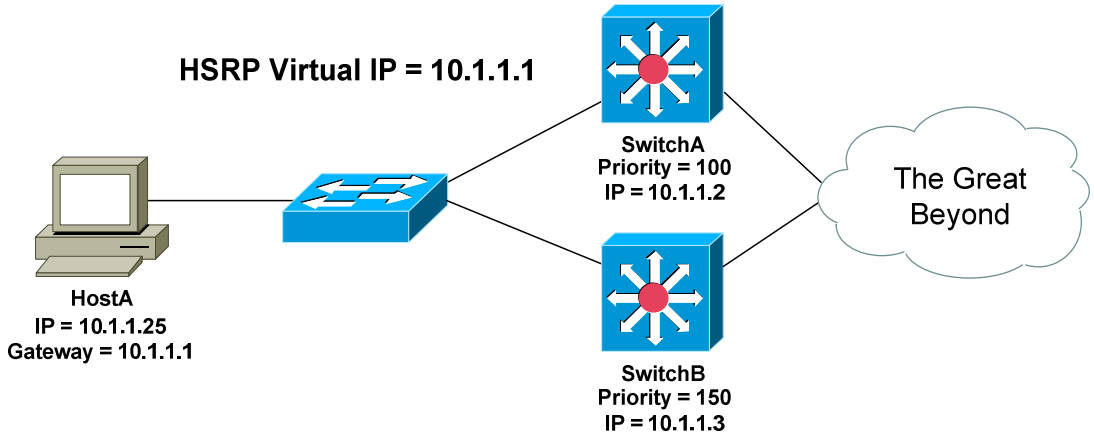
Recall that the interface with the *highest* priority is elected the active router. To configure the priority of a router from its default of **100**:

```
Router(config)# interface gi0/3
Router(config-if)# standby 1 priority 150
```

The *standby 1* command specifies the **HSRP group** the interface belongs to. HSRP can also be configured on a VLAN interface on a multilayer switch:

```
SwitchB(config)# interface vlan 100
SwitchB(config-if)# standby 1 priority 150
```

Each interface in the HSRP group retains its local IP address. The HSRP group itself is assigned a **virtual IP address**, which hosts will use as their default gateway:



To configure the virtual HSRP IP address:

```
SwitchA(config)# interface vlan 100
SwitchA(config-if)# ip address 10.1.1.2 255.255.255.0
SwitchA(config-if)# standby 1 ip 10.1.1.1

SwitchB(config)# interface vlan 100
SwitchB(config-if)# ip address 10.1.1.3 255.255.255.0
SwitchB(config-if)# standby 1 ip 10.1.1.1
```

Remember: while each multilayer switch is configured with its own *local* IP address, both are configured with the same *virtual* IP address. HostA will use this virtual IP address as its default gateway.

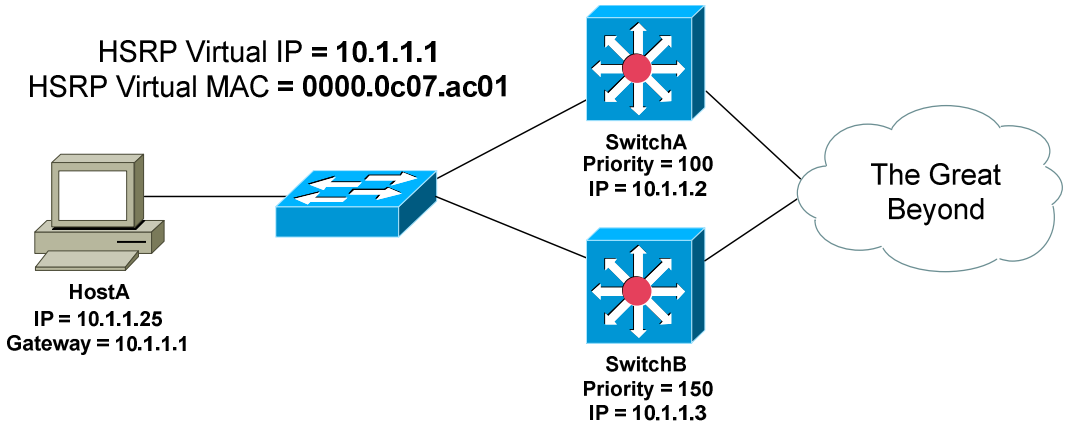
* * *

HSRP Basic Configuration (continued)

HSRP supports using multiple virtual IP addresses:

SwitchB(config-if)# *standby 1 ip 10.1.1.1*
SwitchB(config-if)# *standby 1 ip 10.1.1.5 secondary*

The active router will respond to ARP requests for the virtual IP with the **virtual MAC address.**



The virtual MAC is a reserved address in the following format:

0000.0c07.ac*xx*

…with *xx* representing the HSRP group number in hexadecimal. For example, if the HSRP group number is 8, the resulting virtual MAC address would be *0000.0c07.ac08*.

The HSRP virtual MAC address can be manually changed:

Switch(config-if)# *standby 1 mac-address 0000.00ab.12ef*

HSRP authentication prevents an unauthorized router from joining the HSRP group. All routers in the HSRP group must be configured with an identical authentication string.

To specify a **clear-text** authentication string**:**

Switch(config-if)# *standby 1 authentication STAYOUT*

To specify an MD5-hashed authentication string:

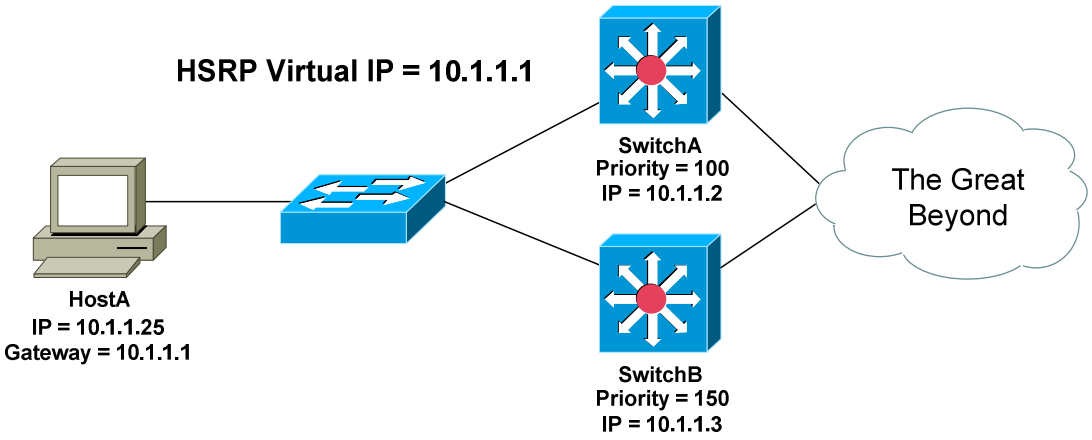
Switch(config-if)# *standby 1 authentication md5 key-string 7 STAYOUT*

* * *

HSRP Preempt

If a new router is added to an HSRP group, it will not *preemptively* assume the role of the active router, even if it has the best priority.

In fact, the router that is first powered on will become the active router, even if it has the lowest priority!



Consider the above example:

1. If SwitchB was powered on first, it would become the active router.
2. SwitchA would be elected the standby router.
3. If SwitchB fails, SwitchA would take over as the active router.
4. Once SwitchB recovers, it *will not* retake its role of active router, *despite* having a higher priority.

The ***preempt*** parameter will allow a router to forcibly assume the role of active router, if it has the highest priority. The *preempt* feature is **disabled** by default:

SwitchB(config-if)# *standby 1 preempt*

The optional *delay* parameter will force a router to wait before preempting as the active router. The delay is measured in *seconds:*

Switch(config-if)# *standby 1 preempt delay 10*

The router can also be forced to wait a specified number of seconds after a *reload* before preempting the active role:

Switch(config-if)# *standby 1 preempt reload 20*

This allows routing protocols to converge before the router becomes active.

* * *

HSRP Timers

Hello packets are used to elect the active and standby router, and to detect if there is a failure. By default, hello packets are exchanged every **3 seconds.**

HSRP Hello packets are sent to the multicast address **224.0.0.2** over UDP port **1985**. If no elections are occurring, only the active and standby routers exchange hello packets.

If no hello packets are received from the active router within the **holddown timer** duration, the standby router will assume it failed and take over as active. By default, the holddown timer is three times the hello timer, or **10 seconds.** Cisco's math, not mine.

To manually adjust the HSRP timers, measured in seconds:

SwitchB(config-if)# *standby 1 timers 4 12*

The first timer value represents the *hello* timer, while the second represents the *holddown* timer. The timers can also be specified in milliseconds:

SwitchB(config-if)# *standby 1 timers msec 800 msec 2400*

Troubleshooting HSRP

To view the status of each HSRP group:

SwitchB# *show standby*

```
VLAN 100 - Group 1
  State is Active
    1 state changes, last state change 00:02:19
  Virtual IP address is 10.1.1.1
  Active virtual MAC address is 0000.0c07.ac01
    Local virtual MAC address is 0000.0c07.ac01 (bia)
  Hello time 3 sec, hold time 10 sec
    Next hello sent in 1.412 secs
  Preemption enabled, min delay 50 sec, sync delay 40 sec
  Active router is local
  Standby router is 10.1.1.2, priority 100 (expires in 6.158 sec)
  Priority 150 (configured 150)
    Tracking 0 objects, 0 up
```

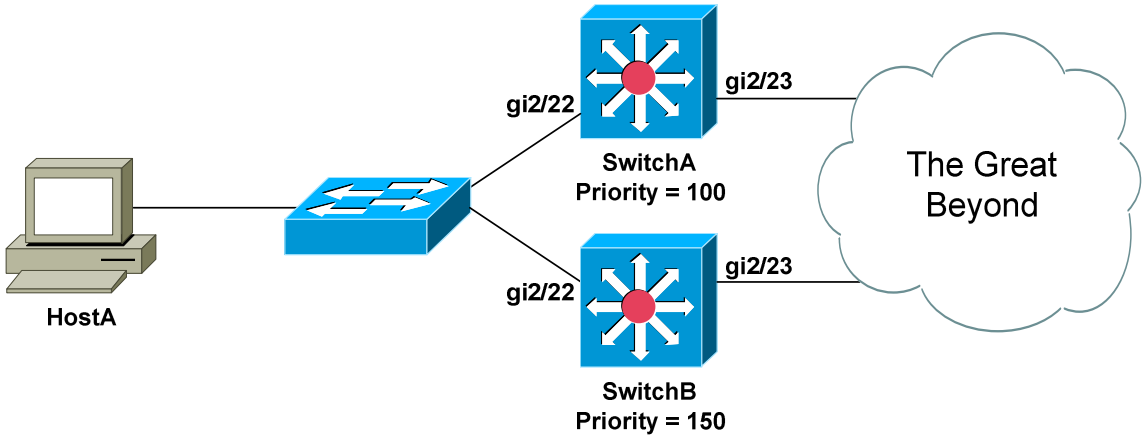
To view a more abbreviated version of this output:

SwitchB# *show standby brief*

```
P indicates configured to preempt.
|
Interface   Grp   Prio   P State   Active addr     Standby addr    Group addr
Vlan100     1     150    P Active  local           10.1.1.2        10.1.1.1
```

* * *

HSRP Tracking



In the above example, SwitchB will become the active router, and SwitchA the standby. Both SwitchA and SwitchB exchange periodic hello packets to update their status.

If interface gi2/23 goes down on SwitchB, hello packets can still be exchanged with SwitchA via interface gi2/22. SwitchA is unaware that SwitchB has a failure and can no longer forward traffic to other networks. SwitchB will remain as the active router, and traffic will be blackholed.

To mitigate a scenario like this, HSRP can **track** interfaces. If a tracked interface fails, the router's priority is *decreased* by a specified value – by default, this is **10.**

Consider the following *tracking* configuration on SwitchB:

SwitchB(config-if)# *standby 1 track gi2/23 60*

If interface gi2/23 on SwitchB fails, the priority of the switch will decrease by 60. The objective is to decrement the priority enough to allow the standby router to take over as active.

This requires conscientious planning - if SwitchB's priority decremented by only 40, it would remain as active, as its priority would still be higher than SwitchA.

For tracking to be successful, the standby router must be configured to *preempt:*

SwitchA(config-if)# *standby 1 preempt*

Otherwise, the standby router will never take over as active.

* * *

Virtual Router Redundancy Protocol (VRRP)

The **Virtual Router Redundancy Protocol (VRRP)** is an industry-standard Layer-3 redundancy protocol, originally defined in RFC 2338. VRRP is nearly identical to HSRP, with some notable exceptions:

- The router with the *highest* priority becomes the **master router**.
- All other routers become **backup routers.**
- The virtual MAC address is the reserved 0000.5e00.01*xx*, with *xx* representing the hexadecimal group number.
- Hello packets are sent every **1 second**, by default, and sent to multicast address 224.0.0.18.
- VRRP will *preempt* by default.
- VRRP cannot directly track interfaces – it can track an *object* which is tied to an interface, though.

Configuration of VRRP is also very similar to HSRP:

SwitchB(config)# *interface vlan 100*
SwitchB(config-if)# *ip address 10.1.1.3 255.255.255.0*
SwitchB(config-if)# *vrrp 1 priority 150*
SwitchB(config-if)# *vrrp 1 authentication STAYOUT*
SwitchB(config-if)# *vrrp 1 ip 10.1.1.1*

As with HSRP, the default VRRP priority is **100**, and a *higher* priority is preferred. Unlike HSRP, preemption is *enabled* by default.

To manually disable preempt:

Switch(config-if)# *no vrrp 1 preempt*

To view the status of each VRRP group:

Switch# *show vrrp*

```
Vlan 100 - Group 1
State is Master
Virtual IP address is 10.1.1.1
Virtual MAC address is 0000.5e00.0101
Advertisement interval is 3.000 sec
Preemption is enabled
min delay is 0.000 sec
Priority 150
Master Router is 10.1.1.3 (local), priority is 150
Master Advertisement interval is 3.000 sec
Master Down interval is 9.711 sec
```

* * *

HSRP and VRRP Pseudo Load Balancing

While HSRP and VRRP do provide redundant gateways for *fault tolerance*, neither provide *load balancing* between those gateways.

Cisco will claim that load balancing is possible. Two separate HSRP or VRRP groups can be configured on a single interface:

```
SwitchA(config)# interface vlan 100
SwitchA(config-if)# ip address 10.1.1.2 255.255.255.0

SwitchA(config-if)# standby 1 priority 150
SwitchA(config-if)# standby 1 preempt
SwitchA(config-if)# standby 1 ip 10.1.1.1

SwitchA(config-if)# standby 2 priority 50
SwitchA(config-if)# standby 2 preempt
SwitchA(config-if)# standby 2 ip 10.1.1.254
```

```
SwitchB(config)# interface vlan 100
SwitchB(config-if)# ip address 10.1.1.3 255.255.255.0

SwitchB(config-if)# standby 1 priority 50
SwitchB(config-if)# standby 1 preempt
SwitchB(config-if)# standby 1 ip 10.1.1.1

SwitchB(config-if)# standby 2 priority 150
SwitchB(config-if)# standby 2 preempt
SwitchB(config-if)# standby 2 ip 10.1.1.254
```

In the above configuration, each HSRP group has been assigned a unique virtual IP address – *10.1.1.1* and *10.1.1.254* respectively. By manipulating the priority, each multilayer switch will become the *active* router for one HSRP group, and the *standby* router for the other group.

```
SwitchA# show standby brief

Interface   Grp  Prio P State    Active addr     Standby addr    Group addr
Vlan100     1    150  P Active   local           10.1.1.3        10.1.1.1
Vlan100     2    50   P Standby  10.1.1.3        local           10.1.1.254
```

```
SwitchB# show standby brief

Interface   Grp  Prio P State    Active addr     Standby addr    Group addr
Vlan100     1    50   P Standby  10.1.1.2        local           10.1.1.1
Vlan100     2    150  P Active   local           10.1.1.2        10.1.1.254
```

To achieve load balancing with this HSRP setup, half of the hosts in VLAN 100 must point to the first virtual address as their gateway – *10.1.1.1*. The other half must use the *other* virtual address as their gateway – *10.1.1.254*.

Simple and practical, right? Certainly, it is perfectly feasible to *manually* configure half of the hosts to use one gateway, and *manually* configure the other half to use another.

But hey – it's not a limitation, it's a *feature*!

<unnecessary obscene commentary edited out>

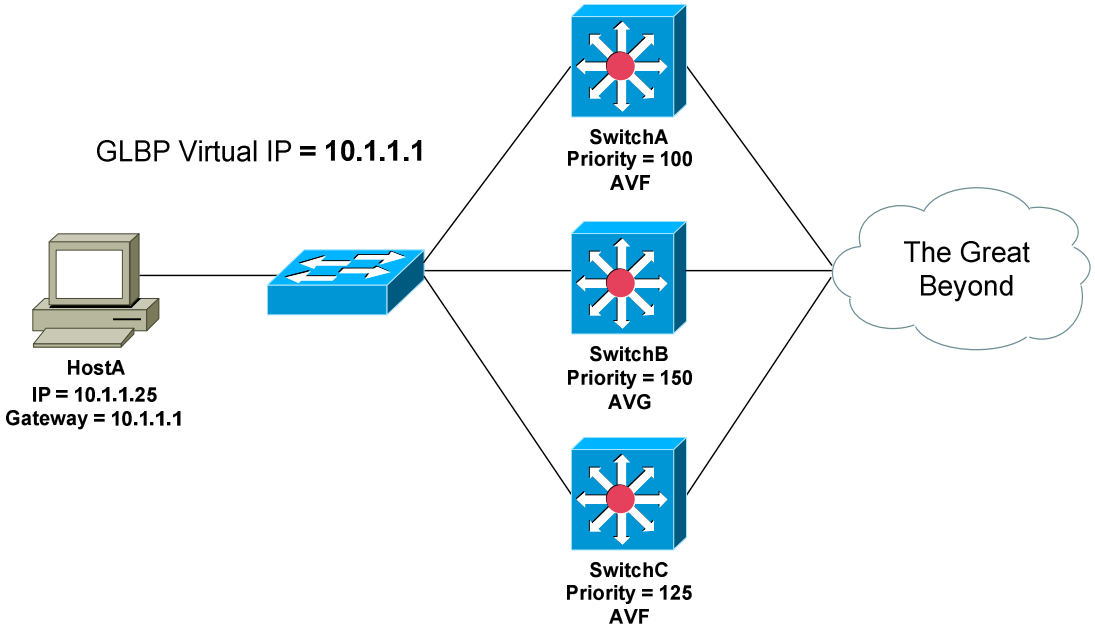
* * *

Gateway Load Balancing Protocol (GLBP)

To overcome the shortcomings in HSRP and VRRP, Cisco developed the proprietary **Gateway Load Balancing Protocol (GLBP).**

Routers are added to a **GLBP group,** numbered 0 to 1023. Unlike HSRP and VRRP, *multiple* GLBP routers can be active, achieving *both* redundancy and load balancing.

A priority is assigned to each GLBP interface - **100** by default. The interface with the highest priority becomes the **Active Virtual Gateway (AVG)**. If priorities are equal, the interface with the highest IP will become the AVG.



Routers in the GLBP group are assigned a single virtual IP address. Hosts will use this virtual address as their default gateway. The AVG will respond to ARP requests for the virtual IP with the virtual MAC address of an **Active Virtual Forwarder (AVF).**

Up to three routers can be elected as AVFs. The AVG assigns a virtual MAC address to each AVF, *and* to itself, for a maximum total of **4 virtual MAC addresses**. Only the AVG and AVFs can forward traffic for hosts.

Any router not elected as an AVF or AVG will become a **Secondary Virtual Forwarder (SVF),** and will wait in standby until an AVF fails.

* * *

Gateway Load Balancing Protocol (GLBP) (continued)

Basic GLBP configuration is nearly identical to HSRP:

```
SwitchB(config)# interface gi2/22
SwitchB(config-if)# glbp 1 priority 150
SwitchB(config-if)# glbp 1 preempt
SwitchB(config-if)# glbp 1 ip 10.1.1.1
```

Remember that the interface with the *highest* priority is elected as the AVG. Preemption is **disabled** by default with GLBP.

What determines whether a router becomes an AVF or SVF? Each router is assigned a **weight**, and the default weight is **100**. A *higher* weight is preferred.

Weight can be configured two ways:

- Statically
- Dynamically, by tracking an interface

To manually specify the weight of a router:

```
SwitchB(config)# interface gi2/22
SwitchB(config-if)# glbp 1 weighting 150
```

Like VRRP, GLBP cannot directly *track* an interface. Instead, an interface must be specified as a **tracked object:**

```
SwitchB(config)# track 10 interface gi2/23
```

The tracked object can then be referenced as part of a *weighting* command:

```
SwitchB(config)# interface gi2/22
SwitchB(config-if)# glbp 1 weighting track 10 decrement 65
```

Note that the tracking *object-number* must match. By default, the weight will decrement by 10.

Weight *thresholds* can be configured, forcing a router to relinquish its AVF role if it falls below the minimum threshold:

```
SwitchB(config)# interface gi2/22
SwitchB(config-if)# glbp 1 weighting 150 lower 90 upper 125
```

If the weight falls below the *lower* threshold, the router must stop functioning as an AVF. The router will become an AVF again once its weight reaches the *upper* threshold, as long as *preempt* is configured.

* * *

Gateway Load Balancing Protocol (GLBP) (continued)

The AVG assigns a virtual MAC address to itself and up to three AVFs. The AVG will respond to ARP requests for the virtual IP address with one of these virtual MAC addresses. This allows GLBP to provide load balancing in addition to redundancy.

GLBP supports three load balancing methods:

- **Round Robin**
- **Weighted**
- **Host-dependent**

The *default* load balancing method is per-host **round robin.** Traffic from hosts is distributed *equally* across all routers in the GLBP group. The AVG will respond to the first host ARP request with the first virtual MAC address. The second ARP request will receive the second virtual MAC address, etc.

The **weighted** load balancing method will distribute traffic proportionally, based on a router's *weight*. Routers with a higher weight will receive a proportionally higher percentage of traffic.

Host-dependent load balancing will provide a host device with the *same* virtual MAC address every time it performs an ARP request.

The load balancing method should be configured on the AVG:

```
SwitchB(config-if)# glbp 1 load-balancing round-robin
SwitchB(config-if)# glbp 1 load-balancing weighted
SwitchB(config-if)# glbp 1 load-balancing host-dependent
```

Other fun facts about GLBP:

- Hello packets are sent every **3 seconds.**
- Hello packets are sent to multicast address **224.0.0.102.**
- The default holddown time is **10 seconds.**
- The virtual MAC address is the reserved 0007.b4*xx.xxyy*, with *xxxx* representing the GLBP group number, and *yy* representing the AVF number.

To view the status of each GLBP group:

```
SwitchB# show glbp
```

* * *

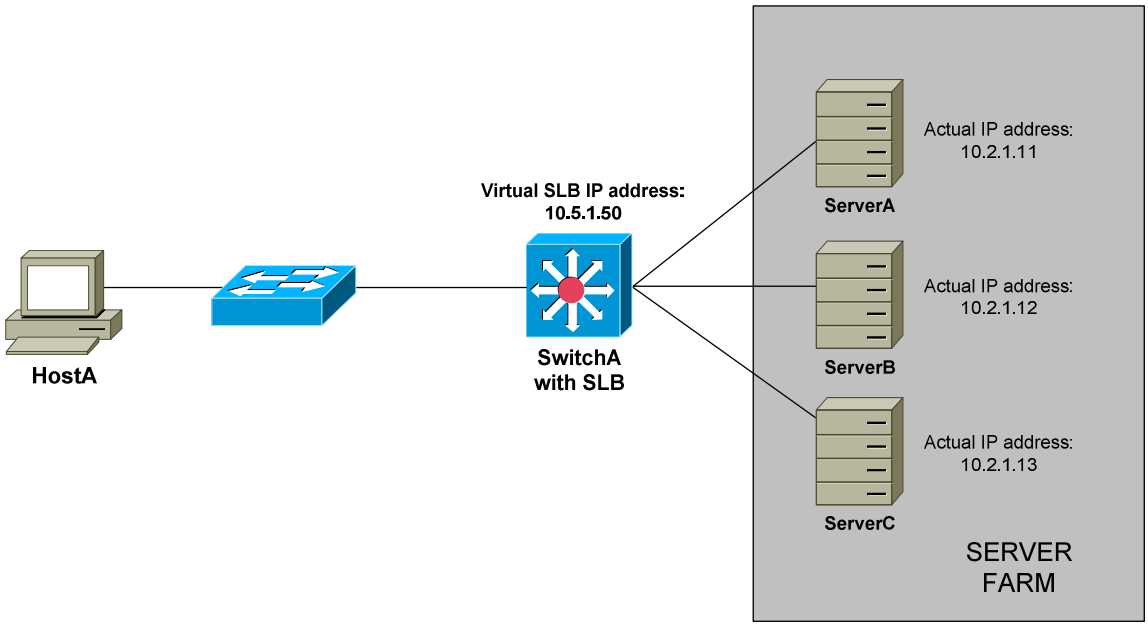
Server Load Balancing (SLB)

HSRP, VRRP, and GLBP provide *gateway* redundancy for clients. A limited number of Cisco platforms also support a basic *clustering* service.

Server Load Balancing (**SLB)** allows a router to apply a virtual IP address to a group of servers. Each server in the cluster should provide the same function and be configured identically, except for their IP addresses.

Hosts make requests to a single virtual IP address to access the server farm. If an individual server fails, the server farm will stay functional, as long as at least one server is online. This is also useful for seamless server repairs and maintenance.

The following diagram demonstrates SLB:



Three servers exist in the server farm. SwitchA is configured with a virtual SLB address of *10.5.1.50*. If HostA wants to access any service from the server farm, it will forward the request to virtual SLB address.

SwitchA accepts the request on behalf on the server farm, and redirects the request to one of the physical servers in the farm. This is **transparent** to the host – it is unaware of which physical server it is truly connecting to.

Note: Cisco is discontinuing support for server load-balancing mechanisms like SLB. There are many competitors that make superior products, such as F5 load balancers.

* * *

Server Load Balancing (SLB) (continued)

SLB supports two load balancing methods:

- **Weighted round robin**
- **Weighted least connections**

Weighted round robin is the default method. Traffic is forwarded to servers in a round robin fashion. However, servers with a higher *weight* will receive a proportionally higher percentage of traffic.

In contrast, **weighted least connections** will allocate traffic based on *current usage*, and will allocate traffic to the server with the least amount of current connections.

SLB requires configuration of two separate components:

- **Server farm**
- **Virtual server**

To configure the Server Farm:

```
SwitchA(config)#  ip slb serverfarm MYFARM
SwitchA(config-slb-sfarm)#  predictor leastconns

SwitchA(config-slb-sfarm)#  real 10.2.1.11
SwitchA(config-slb-real)#  weight 150
SwitchA(config-slb-real)#  inservice

SwitchA(config-slb-sfarm)#  real 10.2.1.12
SwitchA(config-slb-real)#  weight 100
SwitchA(config-slb-real)#  inservice

SwitchA(config-slb-sfarm)#  real 10.2.1.13
SwitchA(config-slb-real)#  weight 75
SwitchA(config-slb-real)#  inservice
```

The *ip slb serverfarm* command sets the server farm name, and enters SLB server farm configuration mode. The *predictor* command sets the load balancing method.

The *real* command identifies the IP address of a physical server in the farm. The *weight* command assigns the load balancing weight for that server. The *inservice* command activates the real server. To deactivate a specific server:

```
SwitchA(config-slb-sfarm)#  real 10.2.1.12
SwitchA(config-slb-real)#  no inservice
```

* * *

SLB Configuration (continued)

To configure the virtual server:

```
SwitchA(config)# ip slb vserver VSERVERNAME
SwitchA(config-slb-vserver)# serverfarm MYFARM

SwitchA(config-slb-vserver)# virtual 10.5.1.50
SwitchA(config-slb-vserver)# client 10.1.0.0 0.0.255.255
SwitchA(config-slb-vserver)# inservice
```

The *ip slb vserver* command sets the virtual server name, and enters SLB virtual server configuration mode. The *serverfarm* command associates the server farm to this virtual server.

The *virtual* command assigns the virtual IP address for the server farm.

The *client* command specifies which hosts can access the server farm. It utilizes a wildcard mask like an access-list.

The *inservice* activates the virtual server. To deactivate the virtual server:

```
SwitchA(config-slb-vserver)# no inservice
```

To view the status of SLB server farms and virtual servers:

```
SwitchA# show ip slb serverfarms
SwitchA# show ip slb vserver
SwitchA# show ip slb real
```

* * *

Switch Chassis Redundancy

Modular Cisco switches support multiple **supervisor engines** for redundancy. This redundancy can be configured in one of three **modes:**

- **Route processor redundancy (RPR)**
- **Route processor redundancy plus (RPR+)**
- **Stateful switchover (SSO)**

With **route processor redundancy (RPR),** the redundant supervisor engine is not fully initialized. If the primary supervisor fails, the standby supervisor must reinitialize all other switch modules in the chassis before functionality is restored. This results in an outage that lasts several minutes.

With **route processor redundancy plus (RPR+),** the redundant supervisor engine is fully initialized, but does not participate in or cache any Layer-2 or Layer-3 functions.

If the primary supervisor fails, the standby supervisor can immediately activate all Layer-2 and Layer-3 functions, without having to reinitialize all other switch modules in the chassis. This process usually takes less than a minute.

With **stateful switchover (SSO),** the redundant supervisor engine is fully initialized, and all Layer-2 and Layer-3 functions are synchronized with the primary supervisor. If the primary supervisor fails, failover to the standby supervisor will occur *immediately*.

To enable redundancy on the switch, and to choose the appropriate redundancy mode:

```
Switch(config)#  redundancy
Switch(config-red)#  mode rpr
Switch(config-red)#  mode rpr-plus
Switch(config-red)#  mode sso
```

The redundancy commands must be configured identically on *both* supervisor engines.

Note: RPR+ mode requires that both supervisor engines utilize the *exact* same version of the Cisco IOS.

* * *

Part V

Switch Security

Section 13
- Switch and VLAN Security -

Switch Port Security

Port Security adds an additional layer of security to the switching network.

The MAC address of a host generally does not change. If a specific host will always remain connected to a specific switch port, then the switch can filter all *other* MAC addresses on that port using Port Security.

Port Security supports both **statically** mapping MAC addresses, and **dynamically** learning addresses from traffic sent on the port.

To enable Port Security on an interface:

```
Switch(config)# interface gi1/10
Switch(config-if)# switchport port-security
```

By default, Port Security will allow **only one MAC** on an interface. To adjust the maximum number of allowed VLANs, up to 1024:

```
Switch(config-if)# switchport port-security maximum 2
```

To statically map the allowed MAC addresses on an interface:

```
Switch(config-if)# switchport port-security mac-address 0001.1111.2222
Switch(config-if)# switchport port-security mac-address 0001.3333.5555
```

Only hosts configured with the above two MAC addresses will be allowed to send traffic through this port.

If the *maximum* number of MAC addresses for this port had been set to *10*, but only two were statically mapped, the switch would dynamically learn the remaining eight MAC addresses.

Port Security refers to dynamically learned MAC addresses as **sticky addresses.** Sticky addresses can be *aged* out after a period of inactivity, measured in minutes:

```
Switch(config-if)# switchport port-security aging time 10
```

Port Security aging is *disabled* by default.

* * *

Switch Port Security (continued)

A **violation** occurs if an *unauthorized* MAC address attempts to forward traffic through a port. There are three violation **actions** a switch can perform:

- **Shutdown –** If a violation occurs, the interface is placed in an *errdisable* state. The interface will stop forwarding *all* traffic, including non-violating traffic, until it is removed from an errdisable state. This is the **default** action for Port Security.
- **Restrict –** If a violation occurs, the interface will remain online. Legitimate traffic will be forwarded, and unauthorized traffic will be dropped. Violations are **logged**, either via a syslog message or SNMP trap.
- **Protect –** If a violation occurs, the interface will remain online. Legitimate traffic will be forwarded and unauthorized traffic will be dropped, but *no* logging will occur.

To configure the desired Port Security violation action:

Switch(config-if)# *switchport port-security violation shutdown*
Switch(config-if)# *switchport port-security violation restrict*
Switch(config-if)# *switchport port-security violation protect*

To view Port Security configuration and status for a specific interface:

Switch# *show port-security interface gi1/10*

```
Port Security: Enabled
Port status: SecureUp
Violation mode: Shutdown
Maximum MAC Addresses: 10
Total MAC Addresses: 10
Configured MAC Addresses: 2
Aging time: 10 mins
Aging type: Inactivity
SecureStatic address aging: Enabled
Security Violation count: 0
```

Note that the *maximum MAC addresses* is set to *10,* and that the *total MAC addresses* is currently at *10* as well. The *violation mode* is set to *shutdown.*

If another MAC address attempts to forward traffic through this port, the port will be place in an errdisable state.

* * *

802.1x Port Authentication

802.1x Port Authentication forces a host device to *authenticate* with the switch, before the switch will forward traffic on behalf of that host. This is accomplished using the **Extensible Authentication Protocol over LANs (EAPOL).** 802.1x only supports **RADIUS** servers to provide authentication.

Both the switch *and* the host must support 802.1x to use port authentication:

- If the *host* supports 802.1x, but the switch does not – the host will not utilize 802.1x and will communicate normally with the switch.
- If the *switch* supports 802.1x, but the host does not – the interface will stay in an **unauthorized** state, and will not forward traffic.

A switch interface configured for 802.1x authentication stays in an **unauthorized** state until a client successfully authenticates. The only traffic permitted through an interface in an unauthorized state is as follows:

- EAPOL, for client authentication
- Spanning Tree Protocol (STP)
- Cisco Discovery Protocol (CDP)

To globally enable 802.1x authentication on the switch:

```
Switch(config)# dot1x system-auth-control
```

To specify the authenticating RADIUS servers, and configure 802.1x to utilize those RADIUS servers:

```
Switch(config)# aaa new-model
Switch(config)# radius-server host 192.168.1.42 key CISCO
Switch(config)# aaa authentication dot1x default group radius
```

Finally, 802.1x authentication must be configured on the desired interfaces:

```
Switch(config)# interface gi1/10
Switch(config-if)# dot1x port-control auto
```

An interface can be configured in one of three 802.1x states:

- **force-authorized –** The interface will *always* authorize any client, essentially disabling authentication. This is the **default** state.
- **force-unauthorized –** The interface will *never* authorize any client, essentially preventing traffic from being forwarded.
- **auto –** The interface will actively attempt to authenticate the client.

* * *

VLAN Access-Lists

Normally, access-lists are applied to interfaces to filter traffic *between* networks or VLANs.

VLAN Access-Lists (VACLs) filter traffic *within* a VLAN. Because *intra-VLAN* traffic does not traverse an interface, it cannot be directly filtered using a normal access-list.

For a VACL to function, traffic must first be *identified* using an access-list:

```
Switch(config)# ip access-list extended BLOCKTHIS
Switch(config-ext-nacl)# permit ip host 10.1.5.10 10.1.0.0 0.0.255.255
```

VACLs support three types of access-lists:

- **IP**
- **IPX**
- **MAC address**

Note that the access-list is **not used to deny or permit traffic**. Instead, it **identifies traffic** for the VACL to deny or permit. The ACL *permit* parameter functions as a *true* statement, and a *deny* parameter functions as a *false* statement.

To configure the VACL itself:

```
Switch(config)# vlan access-map MY_VACL 5
Switch(config-access-map)# match ip address BLOCKTHIS
Switch(config-access-map)# action drop

Switch(config-access-map)# vlan access-map MY_VACL 10
Switch(config-access-map)# action forward

Switch(config)# vlan filter MY_VACL vlan-list 102
```

The first line creates a *vlan access-map* named *MY_VACL*. Traffic that *match*es entries in the *BLOCKTHIS* access-list will be *drop*ped.

The final *vlan access-map* entry contains only an *action* to *forward*. This will apply to **all other traffic,** as no other access-list was specified. Finally the VACL is applied to *VLAN 102*.

Notice that every *access-map* statement contains a sequence number - *5* and *10* respectively in the above example. This dictates the order in which the rules should be processed.

* * *

Private VLANs

Private VLANs (PVLANs) allow for further segmentation of a subnet *within* a VLAN. This is accomplished by creating **secondary** VLANs within a **primary** VLAN.

The *secondary* VLAN **can communicate only** with the *primary* VLAN, and not any other secondary VLANs.

There are two types of secondary VLANs:

- **Community**
- **Isolated**

Hosts *within* a **community** VLAN can communicate with each other, and with the primary VLAN.

Hosts *within* an **isolated** VLAN *cannot* communicate with each other. However, the hosts can still communicate with the primary VLAN.

Private VLANs are only locally-significant to the switch. VTP will not pass information on Private VLANs to other switches.

Each switch port in a private VLAN must be configured with a specific *mode*:

- **Promiscuous**
- **Host**

A **promiscuous** port can communicate with the primary VLAN and all secondary VLANs. Gateway devices such as routers and switches should be configured as promiscuous ports.

A **host** port can communicate only with promiscuous ports, and other ports within the *local* community VLAN. Host devices such as workstations should be configured as host ports.

Private VLANs allow a group of hosts to be segmented within a VLAN, while still allowing those devices to reach external networks via a promiscuous gateway.

Service providers can use private VLANs to segregate the traffic of multiple customers, while allowing them to share a gateway.

Private VLAN Configuration

Configuring private VLANs involves the following:

- Creating secondary VLANs
- Creating primary VLANs
- Associating secondary VLANs to a primary VLAN
- Associating ports to either a primary or secondary VLAN

To create secondary VLANs:

```
Switch(config)# vlan 100
Switch(config-vlan)# private-vlan community

Switch(config)# vlan 101
Switch(config-vlan)# private-vlan isolated
```

To create a primary VLAN, and associate secondary VLANs:

```
Switch(config)# vlan 50
Switch(config-vlan)# private-vlan primary
Switch(config-vlan)# private-vlan association 100,101
```

To associate a port to a primary and secondary VLAN:

```
Switch(config)# interface gi1/10
Switch(config-if)# switchport private-vlan host
Switch(config-if)# switchport private-vlan host-association 50 101
```

The *gi1/10* port has been identified as a *host* port, and associated with primary VLAN *50,* and secondary VLAN *101*.

To configure a promiscuous port:

```
Switch(config)# interface gi1/20
Switch(config-if)# switchport private-vlan promiscuous
Switch(config-if)# switchport private-vlan mapping  50 100.101
```

The promiscuous *gi1/20* port is associated with primary VLAN *50,* and the two secondary VLANs, *100* and *101*.

To allow the primary VLAN SVI to perform Layer-3 forwarding for specified secondary VLANs:

```
Switch(config)# interface vlan 50
Switch(config-if)# private-vlan mapping 100,101
```

* * *

DHCP Snooping

Dynamic Host Control Protocol (DHCP) provides administrators with a mechanism to *dynamically* assign IP addresses, rather than manually configuring the address on each host.

DHCP servers **lease** out IP addresses to DHCP clients, for a specific period of time. There are four steps to this DHCP process:

- When a DHCP client first boots up, it broadcasts a **DHCPDiscover** message, searching for a DHCP server.
- If a DHCP server exists on the local segment, it will respond with a **DHCPOffer,** containing the offered IP address, subnet mask, etc.
- Once the client receives the offer, it will respond with a **DHCPRequest**, indicating that it will accept the offered protocol information.
- Finally, the server responds with a **DHCPACK**, acknowledging the clients acceptance of offered protocol information.

Malicious attackers can place a rogue DHCP server on the trusted network, intercepting DHCP packets while masquerading as a legitimate DHCP server. This is a form of **spoofing** attack, which intends to gain unauthorized access or steal information by sourcing packets from a *trusted* source. This is also referred to as a *man-in-the-middle* attack.

DHCP attacks of this sort can be mitigated by using **DHCP Snooping.** Only specified interfaces will accept DHCPOffer packets – unauthorized interfaces will discard these packets, and then place the interface in an errdisable state.

DHCP Snooping must first be globally enabled on the switch:

```
Switch(config)# ip dhcp snooping
```

Then, DHCP snooping must be enabled for one or more VLANs:

```
Switch(config)# ip dhcp snooping vlan 5
```

By default, all interfaces are considered *untrusted* by DHCP Snooping. Interfaces connecting to legitimate DHCP servers must be *trusted:*

```
Switch(config)# interface gi1/20
Switch(config)# ip dhcp snooping trust
```

* * *

Dynamic ARP Inspection

Another common man-in-the-middle attack is **ARP spoofing,** sometimes referred to as **ARP poisoning**. A malicious host can masquerade as another host, by intercepting ARP requests and responding with its *own* MAC address.

Dynamic ARP Inspection (DAI) mitigates the risk of ARP Spoofing, by inspecting all ARP traffic on *untrusted* ports. DAI will confirm that a legitimate MAC-to-IP translation has occurred, by comparing it against a trusted database.

This MAC-to-IP database can be statically configured, or DAI can utilize the DHCP Snooping table, assuming that DHCP Snooping has been enabled.

To globally enable DAI for one or more VLANs:

```
Switch(config)#  ip arp inspection vlan 100
```

By default, all interfaces in VLAN 100 will be considered **untrusted**, and thus subject to inspection by DAI. Trunk ports to other switches should be configured as **trusted**, indicating no inspection will occur, as each switch should handle DAI locally:

```
Switch(config)#  interface gi1/24
Switch(config-if)#  ip arp inspection trust
```

To create a manual MAC-to-IP database for DAI to reference:

```
Switch(config)#  arp access-list DAI_LIST
Switch(config-acl)#  permit ip host 10.1.1.5 mac host 000a.1111.2222
Switch(config-acl)#  permit ip host 10.1.1.6 mac host 000b.3333.4444

Switch(config)#  ip arp inspection filter DAI_LIST vlan 100
```

If an ARP response does not match the MAC-to-IP entry for a particular IP address, then DAI drops the ARP response and generates a log message.

* * *

Part VI

QoS

Section 14
- Introduction to QoS -

Obstacles to Network Communication

Modern networks support traffic beyond the traditional data types, such as email, file sharing, or web traffic. Increasingly, data networks share a common medium with more sensitive forms of traffic, like *voice* and *video.*

These sensitive traffic types often require *guaranteed* or *regulated* service, as such traffic is more susceptible to the various obstacles of network communication, including:

Lack of Bandwidth – Describes the simple lack of sufficient throughput, which can severely impact sensitive traffic. Increasing bandwidth is generally considered the *best* method of improving network communication, though often expensive and time-consuming.

Bandwidth is generally measured in **bits-per-second (bps)**, and can be offered at a fixed-rate (as Ethernet usually is), or at a variable-rate (as Frame-Relay often is). Various mechanisms, such as **compression**, can be used to pseudo-increase the capacity of a link.

Delay – Defines the latency that occurs when traffic is sent end-to-end across a network. Delay will occur at various points on a network, and will be discussed in greater detail shortly.

Jitter – Describes the fragmentation that occurs when traffic arrives at irregular times or in the wrong order. Jitter is thus a *varying* amount of delay. Voice communication is *especially* susceptible to jitter. Jitter can be somewhat mitigated using a **de-jitter buffer**.

Data Loss – Defines the *packet loss* that occurs due to link congestion. A full queue will drop newly-arriving packets - an effect known as **tail drop**.

All of above factors adversely affect network communication. Voice over IP (VoIP) traffic, for example, begins to degrade when delay is higher than **150 ms**, and when data loss is greater than **1%**.

Quality of Service (QoS) tools have been developed as an alternative to merely increasing bandwidth. These QoS mechanisms are designed to provide specific applications with guaranteed or consistent service in the absence of optimal bandwidth conditions.

* * *

Types of Delay

Delay can occur at many points on a network. Collectively, this is known as **end-to-end delay**. The various *types* of delay include:

- **Serialization Delay –** refers to the time necessary for an interface to encode bits of data onto a physical medium. Calculating serialization delay can be accomplished using a simple formula:

$$\frac{\textbf{\# of bits}}{\textbf{bits per second (bps)}}$$

  Thus, the serialization delay to encode 128,000 bits on a 64,000 bps link would be 2 seconds.

- **Propagation Delay** – refers to the time necessary for a single bit to travel end-to-end on a physical wire. For the incredibly anal geeks, the rough formula to estimate propagation delay on a copper wire:

$$\frac{\textbf{Length of the Physical Wire (in meters)}}{\textbf{2.1 x } 10^{8} \textbf{ meters/second}}$$

- **Forwarding** (or **Processing) Delay –** refers to the time necessary for a router or switch to move a packet between an *ingress* (input) queue and an *egress* (output) queue. Forwarding delay is affected by a variety of factors, such as the routing or switching method used, the speed of the device's CPU, or the size of the routing table.

- **Queuing Delay** – refers to the time spent in an egress queue, waiting for previously-queued packets to be serialized onto the wire. Queues that are too small can become congested, and start dropping newly arriving packets (**tail drop).** This forces a higher-layer protocol (such as TCP) to resend data**.** Queues that are too large can actually queue too many packets, causing long queuing delays.

- **Network (Provider) Delay –** refers to the time spent in a WAN provider's cloud. Network delay can be very difficult to quantify, as it is often impossible to determine the structure of the cloud.

- **Shaping Delay –** refers to the delay initiated by shaping mechanisms intended to slow down traffic to prevent dropped packet due to congestion.

* * *

QoS Methodologies

There are three key methodologies for implementing QoS:

- **Best-Effort**
- **Integrated Services (IntServ)**
- **Differentiated Services (DiffServ)**

Best-Effort QoS is essentially *no* QoS. Traffic is routed on a first-come, first-served basis. Sensitive traffic is treated no differently than normal traffic. Best-Effort is the default behavior of routers and switches, and as such is easy to implement and very scalable. The Internet forwards traffic on a Best-Effort basis.

Integrated Services (IntServ) QoS is also known as *end-to-end* or *hard* QoS. IntServ QoS requires an application to *signal* that it requires a specific level of service. An **Admission Control** protocol responds to this request by allocating or reserving resources end-to-end for the application. If resources *cannot* be allocated for a particular request, then it is denied.

Every device end-to-end must support the IntServ QoS protocol(s). IntServ QoS is not considered a scalable solution for two reasons:

- There is only a finite amount of bandwidth available to *reserved*.
- IntServ QoS protocols add significant overhead on devices end-to-end, as each traffic flow must be statefully maintained.

The **Resource Reservation Protocol (RSVP)** is an example IntServ QoS protocol.

Differentiated Services (DiffServ) QoS was designed to be a scalable QoS solution. Traffic types are organized into specific **classes,** and then **marked** to identify their classification. **Policies** are then created on a *per-hop basis* to provide a specific level of service, depending on the traffic's classification.

DiffServ QoS is popular because of its scalability and flexibility in enterprise environments. However, DiffServ QoS is considered *soft* QoS, as it does not absolutely guarantee service, like IntServ QoS. DiffServ QoS does not employ signaling, and does not enforce end-to-end reservations.

* * *

QoS Tools

Various tools have been developed to enforce QoS. Many of these tools are used in tandem as part of a complete QoS policy:

- **Classification and Marking**
- **Queuing**
- **Queue Congestion Avoidance**

Classification is a method of identifying and then organizing traffic based on service requirements. This traffic is then **marked** or *tagged* based on its classification, so that the traffic can be differentiated. Classification and marking are covered in great detail in another guide.

Queuing mechanisms are used to service *higher* priority traffic before *lower* priority traffic, based on classification. A variety of queuing methods are available:

- First-In First-Out (FIFO)
- Priority Queuing (PQ)
- Custom Queuing (CQ)
- Weighted Fair Queuing (WFQ)
- Class-Based Weighted Fair Queuing (CBWFQ)
- Low-Latency Queuing (LLQ)

Each will be covered in detail in a separate guide.

Queue Congestion Avoidance mechanisms are used to regulate queue usage so that saturation (and thus, tail drop) does not occur. Random Early Detection (RED) and Weighted RED (WRED) are two methods of congestion avoidance, and are both covered in a separate guide.

Configuring QoS on IOS Devices

There are four basic methods of implementing QoS on Cisco IOS devices:

- **Legacy QoS CLI**
- **Modular QoS CLI**
- **AutoQoS**
- **Security Device Manager (SDM) QoS Wizard**

Legacy QoS CLI is a limited and deprecated method of implementing QoS via the IOS command-line. Legacy CLI combined the *classification* of traffic with the *enforcement* of QoS policies. All configuration occurs on a per-interface basis.

Modular QoS CLI (MQC) is an improved command-line implementation of QoS. MQC is considered *modular* because it separates classification (using **class-maps** to match traffic) from policy configuration (using **policy-maps** to apply a specific level of service per classification). Policy-maps are then applied to an interface using a **service-policy.**

AutoQoS is an automated method of generating QoS configurations on IOS devices. AutoQoS, originally developed for VoIP traffic, can run a *discovery* process to analyze and classify a variety of traffic types. AutoQoS can then create QoS policies based on those classifications. Afterwards, MQC can be used to fine-tune AutoQoS's generated configuration.

The **Cisco Security Device Manager (SDM)** is a web-based management GUI for Cisco IOS devices. The SDM **QoS Wizard** provides a graphical method of configuring and monitoring QoS. The Wizard separates traffic into three categories:

- Real-Time – for VoIP and signaling traffic.
- Business-Critical – for transactional, network management, and routing traffic.
- Best Effort – for all other traffic.

A percentage of the interface bandwidth can then be allocated for each traffic category.

MQC and AutoQoS will be covered in greater detail in separate guides.

Section 15
- QoS Classification and Marking -

Classifying and Marking Traffic

Conceptually, DiffServ QoS involves three steps:

- Traffic must be *identified* and then *classified* into groups.
- Traffic must be *marked* on trust boundaries.
- *Policies* must be created to describe the *per-hop behavior* for classified traffic.

DiffServ QoS relies on the **classification** of traffic, to provide differentiated levels of service on a per-hop basis. Traffic can be classified based on a wide variety of criteria called **traffic descriptors**, which include:

- Type of application
- Source or destination IP address
- Incoming interface
- Class of Service (CoS) value in an Ethernet header
- Type of Service (ToS) value in an IP header (*IP Precedence* or *DSCP*)
- MPLS EXP value in a MPLS header

Access-lists can be used to *identify* traffic for classification, based on address or port. However, a more robust solution is Cisco's **Network-Based Application Recognition (NBAR)**, which will dynamically recognize standard or custom applications, and can classify based on payload.

Once classification has occurred, traffic should be **marked**, to indicate the required *level* of QoS service for that traffic. **Marking** can occur within either the *Layer-2* header or the *Layer-3* header.

The point on the network where traffic is classified and marked is known as the **trust boundary.** QoS marks originating from *outside* this boundary should be considered *untrusted*, and removed or changed. As a general rule, traffic should be marked as *close to the source* as possible. In VoIP environments, this is often accomplished on the VoIP phone itself. Traffic classification should not occur in the network core.

Configuring DiffServ QoS on IOS devices requires three steps:

- Classify traffic using a **class-map.**
- Define a QoS policy using a **policy-map.**
- Apply the policy to an interface, using the **service-policy** command.

* * *

Layer-2 Marking

Layer-2 marking can be accomplished for a variety of frame types:

- Ethernet – using the 802.1p Class of Service (CoS) field.
- Frame Relay – using the Discard Eligible (DE) bit.
- ATM - using the Cell Loss Priority (CLP) bit.
- MPLS - using the EXP field.

Marking Ethernet frames is accomplished using the 3-bit **802.1p Class of Service (CoS)** field. The CoS field is part of the 4-byte *802.1Q* field in an Ethernet header, and thus is only available when 802.1Q VLAN frame tagging is employed. The CoS field provides 8 priority values:

Type	***Decimal***	***Binary***	***General Application***
Routine	0	000	*Best effort* forwarding
Priority	1	001	Medium priority forwarding
Immediate	2	010	High priority forwarding
Flash	3	011	VoIP call signaling forwarding
Flash-Override	4	100	Video conferencing forwarding
Critical	5	101	VoIP forwarding
Internet	6	110	Inter-network control (*Reserved*)
Network Control	7	111	Network control (*Reserved*)

Frame Relay and ATM frames provide a less robust marking mechanism, compared to the Ethernet CoS field. Both Frame Relay and ATM frames reserve a 1-bit field, to prioritize which traffic should be dropped during periods of congestion.

Frame Relay identifies this bit as the **Discard Eligible (DE) field**, while ATM refers to this bit as the **Cell Loss Priority (CLP) field**. A value of *0* indicates a *lower likelihood* to get dropped, while a value of *1* indicates a *higher likelihood* to get dropped.

MPLS employs a **3-bit EXP (Experimental) field** within the 4-byte MPLS header. The EXP field provides similar QoS functionality to the Ethernet CoS field.

* * *

Layer-3 Marking

Layer-3 marking is accomplished using the 8-bit **Type of Service (ToS)** field, part of the IP header. A mark in this field will remain *unchanged* as it travels from hop-to-hop, unless a Layer-3 device is explicitly configured to overwrite this field.

There are two marking methods that use the ToS field:

- **IP Precedence** - uses the first three bits of the ToS field.
- **Differentiated Service Code Point (DSCP)** – uses the first six bits of the ToS field. When using DSCP, the ToS field is often referred to as the **Differentiated Services (DS)** field.

These values determine the **per-hop behavior (PHB)** received by each classification of traffic.

IP Precedence

IP Precedence utilizes the **first three bits** (for a total of **eight values**) of the ToS field to identify the *priority* of a packet. Packets with a higher IP Precedence value should be provided with a better level of service. IP Precedence values are comparable to Ethernet CoS values:

Type	***Decimal***	***Binary***	***General Application***
Routine	0	000	*Best effort* forwarding
Priority	1	001	Medium priority forwarding
Immediate	2	010	High priority forwarding
Flash	3	011	VoIP call signaling forwarding
Flash-Override	4	100	Video conferencing forwarding
Critical	5	101	VoIP forwarding
Internet	6	110	Inter-network control (*Reserved*)
Network Control	7	111	Network control (*Reserved*)

By default, all traffic has an IP Precedence of 000 (*Routine*), and is forwarded on a best-effort basis.

Normal network traffic should not (and in most cases, *cannot*) be set to 110 (*Inter-Network Control*) or 111 (*Network Control*), as it could interfere with critical network operations, such as STP calculations or routing updates.

* * *

Differentiated Service Code Point (DSCP)

DSCP utilizes the **first six bits** of the ToS header to identify the *priority* of a packet. The first three bits identify the **Class Selector** of the packet, and is backwards compatible with IP Precedence. The following three bits identify the **Drop Precedence** of the packet.

Class Name	*Binary*	*Class Selector*	*Drop Precedence*
Default	000 000	0	
AF11	001 010	1	Low
AF12	001 100		Medium
AF13	001 110		High
AF21	010 010	2	Low
AF22	010 100		Medium
AF23	010 110		High
AF31	011 010	3	Low
AF32	011 100		Medium
AF33	011 110		High
AF41	100 010	4	Low
AF42	100 100		Medium
AF43	100 110		High
EF	101 110	5	

DSCP identifies six Class Selectors for traffic (numbered 0 - 5). **Class 0** is default, and indicates *best-effort* forwarding. Packets with a higher Class value should be provided with a better level of service. **Class 5** is the highest DSCP value, and should be reserved for the most sensitive traffic.

Within each Class Selector, traffic is also assigned a **Drop Precedence**. Packets with a *higher* Drop Precedence are **more likely** to be dropped during congestion than packets with a *lower* Drop Precedence. Remember that this is applied only *within* the same Class Selector.

The Class Name provides a simple way of identifying the DSCP value. **AF** is short for **Assured Forwarding,** and is the type of service applied to Classes 1 – 4. If a packet is marked AF23, then the Class Selector is **2** (the *2* in 23) and its Drop Precedence is High (the *3* in 23).

Packets marked as Class 0 (Default) or Class 5 (**Expedited Forwarding** or **EF**) do not have a Drop Precedence.

* * *

Modular QoS CLI (MQC)

The **Modular QoS CLI (MQC)** is an improved command-line implementation of QoS that replaced legacy CLI commands on IOS devices. MQC is considered *modular* because it separates classification from policy configurations.

There are three steps to configuring QoS using MQC:

- Classify traffic using a **class-map.**
- Define a QoS policy using a **policy-map.**
- Apply the policy to an interface, using the **service-policy** command.

Classifying and Marking Traffic using MQC

Traffic is classified using one or more of the **traffic descriptors** listed earlier in this guide. This is accomplished using the **class-map** command:

Router(config)# *access-list 101 permit tcp any 10.1.5.0 0.0.0.255 eq www*

Router(config)# *class-map match-any LOWCLASS*
Router(config-cmap)# *match access-group 101*

The *access-list* matches all *http* traffic destined for *10.1.5.0*/24.

The *class-map* command creates a new classification named *LOWCLASS.* The *match-any* parameter dictates that traffic can match *any* of the traffic descriptors within the class-map. Alternatively, specifying *match-all* dictates that traffic must match *all* of the descriptors within the class-map.

Within the class-map, *match* statements are used to identify specific traffic descriptors. The above example (*match access-group*) references an access-list. To match other traffic descriptors:

Router(config)# *class-map match-any HICLASS*
Router(config-cmap)# *match input-interface fastethernet0/0*
Router(config-cmap)# *match ip precedence 4*
Router(config-cmap)# *match ip dscp af21*
Router(config-cmap)# *match any*

The above is *not* a comprehensive list of descriptors that can be matched. Reference the link below for a more complete list.

(Reference: http://www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfmcli2.html)

* * *

Network-Based Application Recognition (NBAR)

Cisco's **Network-Based Application Recognition (NBAR)** provides an alternative to using static access-lists to identify protocol traffic for classification. NBAR introduces three key features:

- Dynamic protocol discovery
- Statistics collection
- Automatic traffic classification

NBAR provides classification abilities beyond that of access-lists, including:

- Ability to classify services that use dynamic port numbers. This is accomplished using the stateful inspection of traffic flows.
- Ability to classify services based on sub-protocol information. For example, NBAR can classify HTTP traffic based on payload, such as the host, URL, or MIME type.

NBAR employs a **Protocol Discovery** process to determine the application traffic types traversing the network. The Protocol Discovery process will then maintain statistics on these traffic types.

NBAR recognizes applications using NBAR **Packet Description Language Modules (PDLMs)**, which are stored in flash on IOS devices. Updated PDLMs are provided by Cisco so that IOS devices can recognize newer application types.

NBAR has specific requirements and limitations:

- NBAR requires that Cisco Express Forwarding (CEF) be enabled.
- NBAR does not support Fast EtherChannel interfaces.
- NBAR supports only 24 concurrent host, URL, or MIME types.
- NBAR can only analyze the first 400 bytes of a packet. Note: This restriction is only for IOS versions previous to 12.3(7), which removed this restriction.
- NBAR cannot read sub-protocol information in secure (encrypted) traffic types, such as HTTPS.
- NBAR does not support fragmented packets.

(Reference Reference: CCNP ONT Official Exam Certification Guide. Amir Ranjbar. Pages 110-112: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6558/ps6612/ps6653/prod_qas09186a00800a3ded.pdf)

* * *

Configuring NBAR

To enable NBAR Protocol Discovery on an interface:

Router(config)# *ip cef*

Router(config)# *interface fa0/0*
Router(config-if)# *ip nbar protocol-discovery*

To view statistics for NBAR-discovered protocol traffic:

Router# *show ip nbar protocol-discovery*

```
FastEthernet0/0
                        Input                    Output
                        -----                    ------
Protocol                Packet Count             Packet Count
                        Byte Count               Byte Count
                        30sec Bit Rate           30sec Bit Rate
                        30sec Max Bit Rate       30sec Max Bit Rate
----------------------- ----------------------   ------------------
http                    15648                    15648
                        154861743                154861743
                        123654                   123654
                        654123                   654123
ftp                     4907                     4907
                        954604255                954604255
                        406588                   406588
                        1085994                  1085994
```

NBAR classification occurs within a MQC class-map, using the *match protocol* command:

Router(config)# *class-map match-any LOWCLASS*
Router(config-cmap)# *match protocol http*
Router(config-cmap)# *match protocol ftp*

Matching traffic based on sub-protocol information supports wildcards:

Router(config)# *class-map match-any HICLASS*
Router(config-cmap)# *match protocol http host *routeralley.com**
Router(config-cmap)# *match protocol http mime "*pdf"*

Custom protocol types can be manually added to the NBAR database:

Router(config)# *ip nbar port-map MYPROTOCOL tcp 1982*

Updated PDLMs can be downloaded into flash and then referenced for NBAR:

Router(config)# *ip nbar pdlm flash://unrealtournament.pdlm*

(Reference: http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t8/dtnbarad.pdf)

* * *

Creating and Applying a QoS Policy using MQC

After traffic has been appropriately classified, **policy-maps** are used to dictate how that traffic should be treated (the per-hop behavior).

```
Router(config)# policy-map THEPOLICY
Router(config-pmap)# class LOWCLASS
Router(config-pmap-c)# set ip precedence 1

Router(config-pmap)# class HICLASS
Router(config-pmap-c)# set ip dscp af41
```

The *policy-map* command creates a policy named *THEPOLICY*. The *class* commands associate the *LOWCLASS* and *HICLASS* class-maps created earlier to this policy-map.

Within the policy-map class sub-configuration mode, *set* statements are used to specify the desired actions for the classified traffic. In the above example, specific *ip precedence* or *ip dscp* values have been marked on their respective traffic classes.

A wide variety of policy actions are available:

```
Router(config)# policy-map LOWPOLICY
Router(config-pmap)# class LOWCLASS
Router(config-pmap-c)# bandwidth 64
Router(config-pmap-c)# queue-limit 40
Router(config-pmap-c)# random-detect
```

The above is *by no means* a comprehensive list of policy actions. Reference the link below for a more complete list. Policy actions such as queuing and congestion avoidance will be covered in great detail in other guides.

Once the appropriate class-map(s) and policy are created, the policy must be applied directionally to an interface. An interface can have up to **two** QoS policies, one each for inbound and outbound traffic.

```
Router(config)# int fa0/0
Router(config-if)# service-policy input THEPOLICY
```

Any traffic matching the criteria of class-maps *LOWCLASS* and *HICLASS*, coming inbound on interface *fa0/0*, will have the actions specified in the policy-map *THEPOLICY* applied.

(Reference: http://www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfmcli2.html)

* * *

Troubleshooting MQC QoS

To view all configured class-maps:

Router# *show class-map*

```
Class Map LOWCLASS
 Match access-group 101

Class Map HICLASS
 Match protocol http host *routeralley.com*
 Match protocol http mime "*pdf"
```

To view all configured policy-maps:

Router# *show policy-map*

```
Policy Map THEPOLICY
    Class LOWCLASS
      set ip precedence 1
    Class HIGHCLASS
      set ip dscp af41
```

To view the statistics of a policy-map on a specific interface:

Router# *show policy-map interface fastethernet0/1*

```
FastEthernet0/0

   Service-policy input: THEPOLICY

     Class-map: LOWCLASS (match-all)
       15648 packets, 154861743 bytes
       1 minute offered rate 512000 bps, drop rate 0 bps
       Match: access-group 101
       QoS Set
         ip precedence 1
           Packets marked 15648
```

Section 16
- QoS and Queuing -

Queuing Overview

A **queue** is used to store traffic until it can be processed or serialized. Both switch and router interfaces have **ingress** (*inbound*) **queues** and **egress** (*outbound*) **queues**.

An ingress queue stores packets until the switch or router CPU can forward the data to the appropriate interface. An egress queue stores packets until the switch or router can *serialize* the data onto the physical wire.

Switch ports and router interfaces contain both **hardware** and **software** queues. Both will be explained in detail later in this guide.

Queue Congestion

Switch (and router) queues are susceptible to **congestion**. Congestion occurs when the rate of *ingress* traffic is greater than can be successfully processed and serialized on an egress interface. Common causes for congestion include:

- The speed of an *ingress* interface is higher than the *egress* interface.
- The combined traffic of multiple ingress interfaces exceeds the capacity of a single egress interface.
- The switch/router CPU is insufficient to handle the size of the forwarding table.

By default, if an interface's queue buffer fills to capacity, new packets will be dropped. This condition is referred to as **tail drop**, and operates on a first-come, first-served basis. If a standard queue fills to capacity, any new packets are indiscriminately dropped, regardless of the packet's classification or marking.

QoS provides switches and routers with a mechanism to **queue** and **service** *higher* priority traffic before *lower* priority traffic. This guide covers various queuing methods in detail.

QoS also provides a mechanism to **drop** *lower* priority traffic before *higher* priority traffic, during periods of congestion. This is known as Weighted Random Early Detection (WRED), and is covered in detail in another guide.

* * *

Types of Queues

Recall that interfaces have both **ingress** (*inbound*) queues and **egress** (*outbound*) queues. Each interface has one or more **hardware queues** (also known as **transmit (TxQ) queues).** Traffic is placed into egress hardware queues to be serialized onto the wire.

There are two *types* of hardware queues. By default, traffic is placed in a **standard queue**, where all traffic is regarded equally. However, interfaces can also support **strict priority** queues, dedicated for higher-priority traffic.

DiffServ QoS can dictate that traffic with a higher DSCP or IP Precedence value be placed in strict priority queues, to be serviced first. Traffic in a strict priority queue is *never dropped* due to congestion.

A Catalyst switch interface may support multiple standard or strict priority queues, depending on the switch model. Cisco notates strict priority queues with a **"p"**, standard queues with a **"q"**, and WRED thresholds per queue (explained in a separate guide) with a **"t".**

If a switch interface supports one strict priority queue, two standard queues, and two WRED thresholds, Cisco would notate this as:

1p2q2t

To view the supported number of hardware queues on a given Catalyst switch interface:

Switch# *show interface fa0/12 capabilities*

The strict priority egress queue must be explicitly *enabled* on an interface:

Switch(config)# *interface fa0/12*
Switch(config-if)# *priority-queue out*

To view the size of the hardware queue of a router serial interface:

Router# *show controller serial*

The size of the interface hardware queue can modified on some Cisco models, using the following command:

Router(config)# *interface serial 0/0*
Router(config-if)# *tx-ring-limit 3*

(Reference: http://www.cisco.com/en/US/tech/tk389/tk813/technologies_tech_note09186a00801558cb.shtml)

* * *

Forms of Queuing

The default form of queuing on nearly all interfaces is **First-In First-Out (FIFO).** This form of queuing requires no configuration, and simply processes and forwards packets in the order that they arrive. If the queue becomes saturated, new packets will be dropped (**tail drop).**

This form of queuing may be insufficient for real-time applications, especially during times of congestion. FIFO will *never* discriminate or give preference to higher-priority packets. Thus, applications such as VoIP can be starved out during periods of congestion.

Hardware queues *always* process packets using the **FIFO** method of queuing. In order to provide a preferred level of service for high-priority traffic, some form of **software queuing** must be used. Software queuing techniques can include:

- First-In First-Out (FIFO) (*default*)
- Priority Queuing (PQ)
- Custom Queuing (CQ)
- Weighted Fair Queuing (WFQ)
- Class-Based Weighted Fair Queuing (CBWFQ)
- Low-Latency Queuing (LLQ)

Each of the above software queuing techniques will be covered separately in this guide.

Software queuing usually employs multiple queues, and each is assigned a specific *priority*. Traffic can then be assigned to these queues, using access-lists or based on classification. Traffic from a higher-priority queue is *serviced* before the traffic from a lower-priority queue.

Please note: traffic *within* a single software queue (sometimes referred to as *sub-queuing*) is always processed using FIFO.

Note also: if the hardware queue is *not congested*, software queues are **ignored.** Remember, software-based queuing is *only* used when the hardware queue is congested. Software queues serve as an *intermediary*, deciding which traffic types should be placed in the hardware queue *first* and *how often*, during periods of congestion.

Priority Queuing (PQ)

Priority Queuing (PQ) employs four separate queues:

- High
- Medium
- Normal (*default*)
- Low

Traffic must be assigned to these queues, usually using access-lists. Packets from the High queue are *always* processed before packets from the Medium queue. Likewise, packets from the Medium queue are *always* processed before packets in the Normal queue, etc. Remember that traffic *within* a queue is processed using FIFO.

As long as there are packets in the High queue, *no packets* from any other queues are processed. Once the High queue is empty, then packets in the Medium queue are processed… but *only* if no new packets arrive in the High queue. This is referred to as a **strict** form of queuing.

The obvious advantage of PQ is that higher-priority traffic is *always* processed first. The nasty disadvantage to PQ is that the lower-priority queues can often *receive no service at all*. A constant stream of High-priority traffic can starve out the lower-priority queues.

To configure PQ, traffic can first be identified using access-lists:

Router(config)# *access-list 2 permit 150.1.1.0 0.0.0.255*
Router(config)# *access-list 100 permit tcp any 10.1.1.0 0.0.0.255 eq www*

Then, the traffic should be placed in the appropriate queues:

Router(config)# *priority-list 1 protocol ip high list 2*
Router(config)# *priority-list 1 protocol ip medium list 100*
Router(config)# *priority-list 1 protocol ip normal*
Router(config)# *priority-list 1 protocol ipx low*
Router(config)# *priority-list 1 default normal*

The size of each queue (measured in *packets*) can be specified:

Router(config)# *priority-list 1 queue-limit 30 40 50 60*

Finally, the *priority-list* must be applied to an interface:

Router(config)# *interface serial0*
Router(config-if)# *priority-group 1*

* * *

Custom Queuing (CQ)

A less *strict* form of queuing is **Custom Queuing (CQ),** which employs a **weighed round-robin** queuing methodology.

Each queue is processed *in order*, but each queue can have a different *weight* or *size* (measured either in bytes, or the number of packets). Each *queue* processes its entire contents during its *turn.* CQ supports a maximum of **16 queues.**

To configure CQ, traffic must first be identified by *protocol* or with an *access-list,* and then placed in a custom queue:

```
Router(config)# access-list 101 permit tcp 172.16.0.0 0.0.255.255 any eq 1982

Router(config)# queue-list 1 protocol ip 1 list 101
Router(config)# queue-list 1 protocol ip 1 tcp smtp
Router(config)# queue-list 1 protocol ip 2 tcp domain
Router(config)# queue-list 1 protocol ip 2 udp domain
Router(config)# queue-list 1 protocol ip 3 tcp www
Router(config)# queue-list 1 protocol cdp 4
Router(config)# queue-list 1 protocol ip 5 lt 1000
Router(config)# queue-list 1 protocol ip 5 gt 800
```

Each custom queue is identified with a number (*1*, *2*, *3* etc.). Once traffic has been assigned to custom queues, then each queue's *parameters* must be specified. Parameters can include:

- A **limit** – size of the queue, measured in number of packets.
- A **byte-count** – size of the queue, measured in number of bytes.

Configuration of queue parameters is straight-forward:

```
Router(config)# queue-list 1 queue 1 limit 15
Router(config)# queue-list 1 queue 2 byte-count 2000
Router(config)# queue-list 1 queue 3 limit 25
Router(config)# queue-list 1 queue 4 byte-count 1024
Router(config)# queue-list 1 queue 4 limit 10
```

Finally, the custom queue must be applied to an interface:

```
Router(config)# interface serial0/0
Router(config-if)# custom-queue-list 1
```

(Reference: http://www.cisco.com/en/US/docs/ios/12_0/qos/configuration/guide/qccq.html)

* * *

Weighted Fair Queuing (WFQ)

Weighted Fair Queuing (WFQ) *dynamically* creates queues based on **traffic flows**. Traffic flows are identified with a hash value generated from the following header fields:

- Source and Destination IP address
- Source and Destination TCP (or UDP) port
- IP Protocol number
- Type of Service value (IP Precedence or DSCP)

Traffics of the same *flow* are placed in the same *flow queue*. By default, a maximum of **256** queues can exist, though this can be increased to **4096**.

If the priority (based on the ToS field) of all packets are the same, bandwidth is divided equally among all queues. This results in low-traffic flows incurring a minimal amount of delay, while high-traffic flows may experience latency.

Packets with a higher priority are *scheduled* before lower-priority packets arriving at the same time. This is accomplished by assigning a **sequence number** to each arriving packet, which is calculated from the last sequence number multiplied by an *inverse weight* (based on the ToS field). In other words a *higher* ToS value results in a *lower* sequence number, and the higher-priority packet will be serviced first.

WFQ is actually the **default** on **slow serial links** (2.048 Mbps or slower). To explicitly enable WFQ on an interface:

```
Router(config)#  interface s0/0
Router(config-if)#  fair-queue
```

The following are optional WFQ parameters:

```
Router(config)#  interface s0/0
Router(config-if)#  fair-queue 128 1024
```

The *128* value increases the maximum size of a queue, measured in packets (**64** is the default). The *1024* value increases the maximum number of queues from its default of **256.**

The following queuing methods are based on WFQ:

- Class-Based Weighted Fair Queuing (CBWFQ)
- Low Latency Queuing (LLQ)

* * *

Class-Based WFQ (CBWFQ)

WFQ suffers from several key disadvantages:

- Traffic cannot be queued based on *user-defined* classes.
- WFQ cannot provide specific bandwidth guarantees to a traffic flow.
- WFQ is only supported on slower links (2.048 Mbps or less).

These limitations were corrected with **Class-Based WFQ (CBWFQ).** CBWFQ provides up to **64** user-defined queues. Traffic within each queue is processed using FIFO. Each queue is provided with a configurable minimum bandwidth guarantee, which can be represented one of three ways:

- As a fixed amount (using the *bandwidth* command).
- As a percentage of the total interface bandwidth (using the *bandwidth percent* command).
- As a percentage of the remaining unallocated bandwidth (using the *bandwidth remaining percent* command).

Note: the above three commands must be used exclusively from each other – it is no possible to use the fixed *bandwidth* command on one class, and *bandwidth percent* command on another class within the same policy.

CBWFQ queues are only held to their minimum bandwidth guarantee during periods of congestion, and can thus exceed this minimum when the bandwidth is available.

By default, only 75% of an interface's total bandwidth can be reserved. This can be changed using the following command:

```
Router(config)# interface s0/0
Router(config-if)# max-reserved-bandwidth 90
```

The key disadvantage with CBWFQ is that no mechanism exists to provide a *strict-priority* queue for real-time traffic, such as VoIP, to alleviate latency. Low Latency Queuing (LLQ) addresses this disadvantage, and will be discussed in detail shortly.

Configuring CBWFQ

CBWFQ is implemented using the Modular Command-Line (MQC) interface. Specifically, a *class-map* is used to identify the traffic, a *policy-map* is used to enforce each queue's bandwidth, and a *service-policy* is used to apply the policy-map to an interface.

```
Router(config)# access-list 101 permit tcp 10.1.5.0 0.0.0.255 any eq http
Router(config)# access-list 102 permit tcp 10.1.5.0 0.0.0.255 any eq ftp

Router(config)# class-map HTTP
Router(config-cmap)# match access-group 101
Router(config)# class-map FTP
Router(config-cmap)# match access-group 102

Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HTTP
Router(config-pmap-c)# bandwidth 256
Router(config-pmap)# class FTP
Router(config-pmap-c)# bandwidth 128

Router(config)# interface serial0/0
Router(config-if)# service-policy output THEPOLICY
```

The above example utilizes the *bandwidth* command to assign a fixed minimum bandwidth guarantee for each class. Alternatively, a percentage of the interface bandwidth (75% of the total bandwidth, by default) can be guaranteed using the *bandwidth percent* command:

```
Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HTTP
Router(config-pmap-c)# bandwidth percent 40
Router(config-pmap)# class FTP
Router(config-pmap-c)# bandwidth percent 20
```

The minimum guarantee can also be based as a percentage of the remaining unallocated bandwidth, using the *bandwidth remaining percent* command.

```
Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HTTP
Router(config-pmap-c)# bandwidth remaining percent 20
Router(config-pmap)# class FTP
Router(config-pmap-c)# bandwidth remaining  percent 20
```

Remember, the *bandwidth, bandwidth percent*, and *bandwidth remaining percent* commands must be used exclusively, not in tandem, with each other.

* * *

Low Latency Queuing (LLQ)

Low-Latency Queuing (LLQ) is an improved version of CBWFQ that includes one or more **strict-priority** queues, to alleviate latency issues for real-time applications. Strict-priority queues are *always* serviced before standard class-based queues.

The key difference between LLQ and PQ (which also has a strict priority queue), is that the LLQ strict-priority queue will *not* starve all other queues. The LLQ strict-priority queue is *policed*, either by *bandwidth* or a *percentage* of the bandwidth.

As with CBWFQ, configuration of LLQ is accomplished using MQC:

```
Router(config)# access-list 101 permit tcp 10.1.5.0 0.0.0.255 any eq http
Router(config)# access-list 102 permit tcp 10.1.5.0 0.0.0.255 any eq ftp
Router(config)# access-list 103 permit tcp 10.1.5.0 0.0.0.255 any eq 666

Router(config)# class-map HTTP
Router(config-cmap)# match access-group 101
Router(config)# class-map FTP
Router(config-cmap)# match access-group 102
Router(config)# class-map SECRETAPP
Router(config-cmap)# match access-group 103

Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HTTP
Router(config-pmap-c)# bandwidth percent 20
Router(config-pmap)# class FTP
Router(config-pmap-c)# bandwidth percent 20
Router(config-pmap)# class SECRETAPP
Router(config-pmap-c)# priority percent 50

Router(config)# int serial0/1
Router(config-if)# service-policy output THEPOLICY
```

Note that the *SECRETAPP* has been assigned to a strict-priority queue, using the *priority percent* command.

(Reference: http://www.cisco.com/en/US/docs/ios/12_0t/12_0t7/feature/guide/pqcbwfq.html)

* * *

Troubleshooting Queuing

To view the configured queuing mechanism and traffic statistics on an interface:

Router# *show interface serial 0/0*

```
Serial 0/0 is up, line protocol is up
  Hardware is MCI Serial
  Internet address is 192.168.150.1, subnet mask is 255.255.255.0
  MTU 1500 bytes, BW 1544Kbit, DLY 20000 usec, rely 255/255, load 1/255
  Encapsulation HDLC, loopback not set
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:00, output 00:00:01, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: Class-based queueing
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations 0/1/256 (active/max active/max total)
     Reserved Conversations 1/1 (allocated/max allocated)
```

To view the packets currently stored in a queue:

Router# *show queue s0/0*

To view policy-map statistics on an interface:

Router# *show policy-map interface s0/0*

```
Serial0/0

   Service-policy input: THEPOLICY

     Class-map: SECRETAPP (match-all)
       123 packets, 44125 bytes
       1 minute offered rate 1544000 bps, drop rate 0 bps
       Match: access-group 103
       Weighted Fair Queuing
         Strict Priority
         Output Queue: Conversation 264
            Bandwidth 772 (Kbps)
            (pkts matched/bytes matched) 123/44125
```

* * *

Section 17
- QoS and Congestion Avoidance -

Queue Congestion

Switch (and router) queues are susceptible to **congestion**. Congestion occurs when the rate of *ingress* traffic is greater than can be successfully processed and serialized on an egress interface. Common causes for congestion include:

- The speed of an *ingress* interface is higher than the *egress* interface.
- The combined traffic of multiple ingress interfaces exceeds the capacity of a single egress interface.
- The switch/router CPU is insufficient to handle the size of the forwarding table.

By default, if an interface's queue buffer fills to capacity, new packets will be dropped. This condition is referred to as **tail drop**, and operates on a first-come, first-served basis. If a standard queue fills to capacity, any new packets are indiscriminately dropped, regardless of the packet's classification or marking.

QoS provides switches and routers with a mechanism to **queue** and **service** *higher* priority traffic before *lower* priority traffic. Queuing is covered in detail in a separate guide.

QoS also provides a mechanism to **drop** *lower* priority traffic before *higher* priority traffic, during periods of congestion. This is known as Weighted Random Early Detection (WRED), and is covered in detail in this guide.

* * *

Random Early Detection (RED) and Weighted RED (WRED)

Tail drop proved to be an inefficient method of congestion control. A more robust method was developed called **Random Early Detection (RED).** RED prevents the queue from filling to capacity, by randomly dropping packets in the queue. RED essentially takes advantage of TCP's ability to resend dropped packets.

RED helps alleviate two TCP issues caused by tail drop:

- **TCP Global Synchronization –** occurs when a large number of TCP packets are dropped simultaneously. Hosts will reduce TCP traffic (referred to as *slow start)* in response, and then ramp up again… *simultaneously*. This results in cyclical periods of extreme congestion, followed by periods of under-utilization of the link.
- **TCP Starvation** – occurs when TCP flows are *stalled* during times of congestion (as detailed above), allowing non-TCP traffic to saturate a queue (and thus starving out the TCP traffic).

RED will randomly drop queued packets based on configurable *thresholds.* By dropping only *some* of the traffic before the queue is saturated, instead of *all* newly-arriving traffic (*tail drop)*, RED limits the impact of TCP global synchronization.

RED will drop packets using one of three methods:

- **No drop** – used when there is no congestion.
- **Random drop** – used to prevent a queue from becoming saturated, based on thresholds.
- **Tail drop** – used when a queue *does* become saturated.

RED *indiscriminately* drops random packets. It has no mechanism to differentiate between traffic flows. Thus, RED is mostly deprecated.

Weighted Random Early Detection (WRED) provides more granular control – packets with a lower IP Precedence or DCSP value can be dropped *more frequently* than higher priority packets.

This guide will concentrate on the functionality and configuration of WRED.

* * *

WRED Fundamentals

There are two methods to configuring WRED. **Basic WRED configuration** is accomplished by configuring minimum and maximum packet *thresholds* for each IP Precedence or DSCP value.

- The **minimum threshold** indicates the minimum number of packets that must be queued, before packets of a specific IP Precedence or DSCP value will be *randomly* dropped.
- The **maximum threshold** indicates the number of packets that must be queued, before *all* new packets of a specific IP Precedence or DSCP value are dropped. When the maximum threshold is reached, WRED essentially mimics the tail drop method of congestion control.
- The **mark probability denominator (MPD)** determines the number of packets that will be dropped, when the size of the queue is in between the minimum and maximum thresholds. This is measured as a fraction, specifically 1/MPD. For example, if the MPD is set to 5, one out of every 5 packets will be dropped. In other words, the chance of each packet being dropped is 20%.

Observe the following table:

Precedence	Minimum Threshold	Maximum Threshold	MPD
0	10	25	5
1	12	25	5
2	14	25	5
3	16	25	5

If the WRED configuration matched the above, packets with a precedence of *0* would be randomly dropped once *10* packets were queued. Packets with a precedence of *2* would similarly be dropped once *14* packets were queued. The maximum queue size is *25*, thus all new packets of any precedence would be dropped once *25* packets were queued.

Advanced WRED configuration involves tuning WRED maximum and minimum thresholds on a per-queue basis, rather than to specific IP Precedence or DSCP values. In this instance, the min and max thresholds are based on *percentages*, instead of a specific number of packets. This is only supported on higher model Catalyst switches.

WRED only affects standard queues. Traffic from strict priority queues is never dropped by WRED.

* * *

Configuring Basic WRED

WRED configuration can be based on either **IP Precedence** or a **DSCP value**. To configure WRED thresholds using IP Precedence:

```
Router(config)# interface fa0/1
Router(config-if)# random-detect
Router(config-if)# random-detect precedence 0 10 25 5
Router(config-if)# random-detect precedence 1 12 25 5
Router(config-if)# random-detect precedence 2 14 25 5
Router(config-if)# random-detect precedence 3 16 25 5
Router(config-if)# random-detect precedence 4 18 25 5
Router(config-if)# random-detect precedence 5 20 25 5
```

The first *random-detect* command enables WRED on the interface. The subsequent *random-detect* commands apply a minimum threshold, maximum threshold, and MPD value, for each specified IP Precedence level.

To configure WRED thresholds using DSCP values:

```
Router(config)# interface fa0/10
Router(config-if)# random-detect
Router(config-if)# random-detect dscp-based af11 14 25 5
Router(config-if)# random-detect dscp-based af12 12 25 5
Router(config-if)# random-detect dscp-based af13 10 25 5
Router(config-if)# random-detect dscp-based af21 20 25 5
Router(config-if)# random-detect dscp-based af22 18 25 5
Router(config-if)# random-detect dscp-based af23 16 25 5
```

To view the WRED status and configuration on all interfaces:

```
Router# show interface random-detect
Router# show queuing
```

WRED is **not compatible** with Custom Queuing (CQ), Priority Queuing (PQ) or Weighted Fair Queuing (WFQ), and thus *cannot be enabled* on interfaces using one of those queuing methods.

(Reference: http://www.cisco.com/en/US/docs/ios/12_0/qos/configuration/guide/qcwred.html)

* * *

Configuring Advanced WRED with WRR

On higher-end Catalyst models, WRED can be handled on a per-queue basis, and is configured in conjunction with a feature called **Weighted Round Robin (WRR)**.

Recall that interfaces have both **ingress** (*inbound*) queues and **egress** (*outbound*) queues. Each interface has one or more **hardware queues** (also known as **transmit (TxQ) queues).** Traffic is placed into egress hardware queues to be serialized onto the wire.

There are two *types* of hardware queues. By default, traffic is placed in a **standard queue**, where all traffic is regarded equally. However, interfaces can also support **strict priority** queues, dedicated for higher-priority traffic.

DiffServ QoS can dictate that traffic with a higher DSCP or IP Precedence value be placed in strict priority queues, to be serviced first. Traffic in a strict priority queue is *never dropped* due to congestion.

A Catalyst switch interface may support multiple standard or strict priority queues, depending on the switch model. Cisco notates strict priority queues with a **"p"**, standard queues with a **"q"**, and WRED thresholds per queue (explained in a separate guide) with a **"t".**

If a switch interface supports one strict priority queue, two standard queues, and two WRED thresholds, Cisco would notate this as:

1p2q2t

To view the supported number of hardware queues on a given Catalyst switch interface:

Switch# *show interface fa0/12 capabilities*

The strict priority egress queue must be explicitly *enabled* on an interface:

Switch(config)# *interface fa0/12*
Switch(config-if)# *priority-queue out*

* * *

Configuring Advanced WRED with WRR(continued)

Standard egress queues can be assigned **weights**, which dictate the proportion of traffic sent across each queue:

Switch(config-if)# *wrr-queue bandwidth 127 255*

The above command would be used if a particular port has two standard egress queues (remember, the number of queues depends on the Catalyst model). The two numbers are the weights for Queue 1 and Queue 2, respectively. The weight is a number between 1 and 255, and serves as a ratio for sending traffic.

In the above example, Queue 2 would be allowed to transmit twice as much traffic as Queue 1 every **cycle** (*255* is roughly twice that of *127*). This way, the higher-priority traffic should always be serviced first, and more often.

Next, WRED/WRR can be enabled for a particular queue. Cisco's documentation on this is inconsistent on whether it is enabled by default, or not. To manually enable WRED/WRR on Queue 1:

Switch(config-if)# *wrr-queue random-detect 1*

To disable WRED/WRR and revert to tail-drop congestion control:

Switch(config-if)# *no wrr-queue random-detect 1*

Next, the WRED/WRR minimum and maximum thresholds must be tuned. Again, this is accomplished per standard queue, and based on a *percentage* of the capacity of the queue.

Recall that each switch port has a specific set of queues (for example, **1p2q2t**). The **2t** indicates that two WRED/WRR thresholds can exist per standard queue.

Switch(config-if)# *wrr-queue random-detect min-threshold 1 5 10*
Switch(config-if)# *wrr-queue random-detect max-threshold 1 40 100*

The first command sets two separate *min-threshold*s for Queue *1*, specifically *5* percent and *10* percent.

The second command sets two separate *max-threshold*s for Queue *1*, specifically *40* percent and *100* percent.

* * *

Configuring Advanced WRED with WRR (continued)

Why two separate minimum and maximum thresholds per queue? Because packets of a specific CoS value can be mapped to a specific threshold of a specific queue.

Observe:

Switch(config-if)# *wrr-queue cos-map 1 1 0 1*
Switch(config-if)# *wrr-queue cos-map 1 2 2 3*

The first command creates a *map*, associating queue *1*, threshold *1* with CoS values of *0* and *1*.

The second command creates a *map*, associating queue *1,* threshold *2* with CoS values of *2* and *3*.

All traffic marked with CoS value 0 or 1 will have a minimum threshold of 5 percent, and a maximum threshold of 40 percent (per the earlier commands). All traffic marked with CoS value 2 or 3 will have a minimum threshold of 10 percent, and a maximum threshold of 100 percent.

The above *wrr-queue* commands are actually the *default* settings on higher-end Catalyst switches.

To view the QoS settings on a Catalyst interface:

Switch# *show mls qos interface fa0/10*

To view the queuing information for a Catalyst interface:

Switch# *show mls qos interface fa0/10 queuing*

To view QoS mapping configurations:

Switch# *show mls qos maps*

* * *

Configuring Class-Based WRED (CBWRED)

The functionality of Class-Based Weighted Fair Queuing (CBWFQ) can be combined with WRED to form **Class-Based WRED (CBWRED).** CBWFQ is covered in detail in a separate guide.

CBWRED is implemented within a policy-map:

```
Router(config)# class-map HIGH
Router(config-cmap)# match ip precedence 5
Router(config)# class-map LOW
Router(config-cmap)# match ip precedence 0 1 2

Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HIGH
Router(config-pmap-c)# bandwidth percent 40
Router(config-pmap-c)# random-detect
Router(config-pmap-c)# random-detect precedence 5 30 50 5

Router(config-pmap)# class LOW
Router(config-pmap-c)# bandwidth percent 20
Router(config-pmap-c)# random-detect
Router(config-pmap-c)# random-detect precedence 0 20 50 5
Router(config-pmap-c)# random-detect precedence 1 22 50 5
Router(config-pmap-c)# random-detect precedence 2 24 50 5

Router(config)# int fa0/1
Router(config-if)# service-policy output THEPOLICY
```

DSCP values can be used in place of IP Precedence:

```
Router(config)# class-map HIGH
Router(config-cmap)# match ip dscp af31 af41

Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HIGH
Router(config-pmap-c)# bandwidth percent 40
Router(config-pmap-c)# random-detect dscp-based
Router(config-pmap-c)# random-detect dscp af31 28 50 5
Router(config-pmap-c)# random-detect dscp af41 30 50 5
```

To view CBWRED statistics:

```
Router# show policy-map
```

* * *