Cisco CCNA Study Guide

v2.71 © 2014

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

Foreword:

This study guide is intended to provide those pursuing the CCNA certification with a framework of what concepts need to be studied. This is ***not*** *a comprehensive document containing all the secrets of the CCNA, 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 Networking Concepts

* * *

Section 1
- Introduction to Networks -

What is a Network?

A **network** is simply defined as something that *connects* things together for a specific purpose. The term *network* is used in a variety of contexts, including telephone, television, computer, or even people networks.

A **computer network** connects two or more devices together to share a nearly limitless range of *information* and *services*, including:

- Documents
- Email and messaging
- Websites
- Databases
- Music
- Printers and faxes
- Telephony and videoconferencing

Protocols are *rules* that govern how devices communicate and share information across a network. Examples of protocols include:

- **IP –** Internet Protocol
- **HTTP -** Hyper Text Transfer Protocol
- **SMTP –** Simple Mail Transfer Protocol

Multiple protocols often work together to facilitate end-to-end network communication, forming protocol **suites** or **stacks.** Protocols are covered in great detail in other guides.

Network reference models were developed to allow products from different manufacturers to interoperate on a network. A network reference model serves as a blueprint, detailing standards for how protocol communication should occur.

The **Open Systems Interconnect (OSI)** and **Department of Defense (DoD)** models are the most widely recognized reference models. Both are covered in great detail in another guide.

* * *

Basic Network Types

Network *types* are often defined by function or size. The two most common categories of networks are:

- **LANs (Local Area Networks)**
- **WANs (Wide Area Networks)**

A **LAN** is generally a high-speed network that covers a small geographic area, usually contained within a single building or campus. A LAN is usually under the administrative control of a single organization. **Ethernet** is the most common LAN technology.

A **WAN** can be defined one of two ways. The *book definition* of a WAN is a network that spans large geographical locations, usually to connect multiple LANs. This is a general definition, and not always accurate.

A more *practical definition* of a WAN is a network that traverses a public or commercial carrier, using one of several *WAN technologies***.** A WAN is often under the administrative control of several organizations (or *providers*), and does not necessarily need to span large geographical distances.

A **MAN (Metropolitan Area Network**) is another category of network, though the term is not prevalently used. A MAN is defined as a network that connects LAN's across a city-wide geographic area.

An **internetwork** is a general term describing multiple networks connected together. The **Internet** is the largest and most well-known internetwork.

Some networks are categorized by their *function*, as opposed to their *size*. A **SAN (Storage Area Network)** provides systems with high-speed, lossless access to high-capacity storage devices.

A **VPN (Virtual Private Network**) allows for information to be securely sent across a public or unsecure network, such as the Internet. Common uses of a VPN are to connect branch offices or remote users to a main office.

* * *

Network Architectures

A **host** refers to any device that is connected to a network. A host can also be defined as any device assigned a **network address**.

A host can serve one or more functions:

- A host can *request* data, often referred to as a **client.**
- A host can *provide* data, often referred to as a **server.**
- A host can both request *and* provide data, often referred to as a **peer.**

Because of these varying functions, multiple network **architectures** have been developed, including:

- **Peer-to-Peer**
- **Client/Server**
- **Mainframe/Terminal**

In a basic **peer-to-peer** architecture, all hosts on the network can both *request* and *provide* data and services. For example, two Windows XP workstations configured to share files would be considered a peer-to-peer network.

Peer-to-peer networks are very simple to configure, yet this architecture presents several challenges. Data is difficult to manage and back-up, as it is **spread across multiple devices**. Security is equally problematic, as user accounts and permissions much be configured individually on each host.

In a **client/server** architecture, hosts are assigned specific roles. *Clients* request data and services stored on *servers*. An example of a client/server network would be Windows XP workstations accessing files off of a Windows 2003 server.

There are several advantages to the client/server architecture. Data and services are now **centrally located** on one or more servers, consolidating the management and security of that data. As a result, client/server networks can scale far larger than peer-to-peer networks.

One key disadvantage of the client/server architecture is that the server can present a **single point of failure.** This can be mitigated by adding *redundancy* at the server layer.

Network Architectures (continued)

In a **mainframe/terminal** architecture, a single device (the **mainframe**) stores all data and services for the network. This provides the same advantages as a client/server architecture – centralized management and security of data.

Additionally, the mainframe performs all processing functions for the **dumb terminals** that connect to the mainframe. The dumb terminals perform *no processing whatsoever*, but serve only as input and output devices into the mainframe.

In simpler terms, the mainframe handles all *thinking* for the dumb terminals. A dumb terminal typically consists of only a keyboard/mouse, a display, and an interface card into the network.

The traditional mainframe architecture is less prevalent now than in the early history of networking. However, the similar **thin-client** architecture has gained rapid popularity. A thin-client can be implemented as either a hardware device, or software running on top of another operating system (such as Windows or Linux).

Like dumb terminals, thin-clients require a centralized system to perform all (or most) processing functions. User sessions are spawned and managed completely within the server system.

Hardware thin-clients are generally inexpensive, with a small footprint and low power consumption. For environments with a large number of client devices, the thin-client architecture provides high scalability, with a lower total cost of ownership.

The two most common thin-client protocols are:

- **RDP (Remote Desktop Protocol) –** developed by Microsoft
- **ICA (Independent Computer Architecture) –** developed by Citrix

* * *

Section 2
- OSI Reference Model -

Network Reference Models

A **computer network** connects two or more devices together to share information and services. Multiple networks connected together form an **internetwork.**

Internetworking present challenges - interoperating between products from different manufacturers requires consistent standards. **Network reference models** were developed to address these challenges. A network reference model serves as a blueprint, detailing how communication between network devices should occur.

The two most recognized network reference models are:

- The **Open Systems Interconnection (OSI)** model
- The **Department of Defense (DoD)** model

Without the framework that network models provide, all network hardware and software would have been proprietary. Organizations would have been locked into a single vendor's equipment, and global networks like the Internet would have been impractical, if not impossible.

Network models are organized into **layers,** with each layer representing a specific networking function. These functions are controlled by **protocols**, which are *rules* that govern end-to-end communication between devices.

Protocols on one layer will interact with protocols on the layer above and below it, forming a protocol **suite** or **stack.** The **TCP/IP suite** is the most prevalent protocol suite, and is the foundation of the Internet.

A network model is not a physical entity – there is no OSI *device*. Manufacturers do not always strictly adhere to a reference model's blueprint, and thus not every protocol fits perfectly within a single layer. Some protocols can function across multiple layers.

* * *

OSI Reference Model

The **Open Systems Interconnection (OSI) model** was developed by the **International Organization for Standardization (ISO),** and formalized in 1984. It provided the first framework governing how information should be sent across a network.

The OSI model consists of seven layers, each corresponding to a specific network function:

7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data-link
1	Physical

Note that the *bottom* layer is Layer 1. Various mnemonics make it easier to remember the order of the OSI model's layers:

7	Application	**A**ll	**A**way
6	Presentation	**P**eople	**P**izza
5	Session	**S**eem	**S**ausage
4	Transport	**T**o	**T**hrow
3	Network	**N**eed	**N**ot
2	Data-link	**D**ata	**D**o
1	Physical	**P**rocessing	**P**lease

ISO further developed an entire protocol suite based on the OSI model; however, the **OSI protocol suite** was never widely implemented.

The OSI model itself is now somewhat deprecated – modern protocol suites, such as the TCP/IP suite, are difficult to fit cleanly within the OSI model's seven layers. This is especially true of the **upper three layers**.

The **bottom** (or ***lower)*** **four layers** are more clearly defined, and terminology from those layers is still prevalently used. Many protocols and devices are described by which lower layer they operate at.

* * *

OSI Model - The Upper Layers

The top three layers of the OSI model are often referred to as the **upper layers**:

- Layer-7 - **Application** layer
- Layer-6 - **Presentation** layer
- Layer-5 - **Session** layer

Protocols that operate at these layers manage application-level functions**,** and are generally implemented in *software*.

The function of the upper layers of the OSI model can be difficult to visualize. Upper layer protocols do not always fit perfectly within a layer, and often function across multiple layers.

OSI Model - The Application Layer

The **Application layer (Layer-7)** provides the interface between the user application and the network. A web browser and an email client are examples of user applications.

The user application itself *does not* reside at the Application layer - the *protocol* does. The user interacts with the application, which in turn interacts with the application protocol.

Examples of Application layer protocols include:

- **FTP**, via an FTP client
- **HTTP**, via a web browser
- **POP3** and **SMTP**, via an email client
- **Telnet**

The Application layer provides a variety of functions:

- Identifies communication partners
- Determines resource availability
- Synchronizes communication

The Application layer interacts with the Presentation layer below it. As it is the top-most layer, it does not interact with any layers above it.

(Reference: http://docwiki.cisco.com/wiki/Internetworking_Basics)

* * *

OSI Model - The Presentation Layer

The **Presentation layer (Layer-6)** controls the *formatting* and *syntax* of user data for the application layer. This ensures that data from the *sending* application can be understood by the *receiving* application.

Standards have been developed for the formatting of data types, such as text, images, audio, and video. Examples of Presentation layer formats include:

- **Text** - RTF, ASCII, EBCDIC
- **Images** - GIF, JPG, TIF
- **Audio** - MIDI, MP3, WAV
- **Movies** - MPEG, AVI, MOV

If two devices do not support the same format or syntax, the Presentation layer can provide **conversion** or **translation** services to facilitate communication.

Additionally, the Presentation layer can perform **encryption** and **compression** of data, as required. However, these functions can also be performed at lower layers as well. For example, the Network layer can perform encryption, using IPSec.

OSI Model - The Session Layer

The **Session layer (Layer-5)** is responsible for establishing, maintaining, and ultimately terminating *sessions* between devices. If a session is *broken,* this layer can attempt to recover the session.

Sessions communication falls under one of three categories:

- **Full-Duplex** – simultaneous two-way communication
- **Half-Duplex** – two-way communication, but not simultaneous
- **Simplex –** one-way communication

Many modern protocol suites, such as TCP/IP, do not implement Session layer protocols. Connection management is often controlled by lower layers, such as the Transport layer.

The lack of true Session layer protocols can present challenges for high-availability and failover. Reliance on lower-layer protocols for session management offers less flexibility than a strict adherence to the OSI model.

* * *

OSI Model - The Lower Layers

The bottom four layers of the OSI model are often referred to as the **lower layers**:

- Layer-4 – **Transport** layer
- Layer-3 – **Network** layer
- Layer-2 – **Data-Link** layer
- Layer-1 – **Physical** layer

Protocols that operate at these layers control the end-to-end transport of data between devices, and are implemented in both software and hardware.

OSI Model - The Transport Layer

The **Transport layer (Layer-4)** does *not* actually send data, despite its name. Instead, this layer is responsible for the *reliable* transfer of data, by ensuring that data arrives at its destination error-free and in order.

Transport layer communication falls under two categories:

- **Connection-oriented** – requires that a connection with specific agreed-upon parameters be established before data is sent.
- **Connectionless** – requires no connection before data is sent.

Connection-oriented protocols provide several important services:

- **Segmentation and sequencing –** data is *segmented* into smaller pieces for transport. Each segment is assigned a *sequence number*, so that the receiving device can reassemble the data on arrival.
- **Connection establishment –** connections are established, maintained, and ultimately terminated between devices.
- **Acknowledgments –** receipt of data is confirmed through the use of *acknowledgments*. Otherwise, data is retransmitted, guaranteeing delivery.
- **Flow control** (or **windowing) –** data transfer rate is negotiated to prevent congestion.

The TCP/IP protocol suite incorporates two Transport layer protocols:

- **Transmission Control Protocol (TCP) –** connection-oriented
- **User Datagram Protocol (UDP) -** connectionless

(Reference: http://www.tcpipguide.com/free/t_TransportLayerLayer4-2.htm)

* * *

OSI Model - The Network Layer

The **Network layer (Layer-3)** controls *internetwork* communication, and has two key responsibilities:

- **Logical addressing** – provides a unique address that identifies both the *host*, and the *network* that host exists on.
- **Routing** – determines the *best path* to a particular destination network, and then *routes* data accordingly.

Two of the most common Network layer protocols are:

- **Internet Protocol (IP)**
- Novell's **Internetwork Packet Exchange (IPX)**.

IPX is almost entirely deprecated. IP version 4 (IPv4) and IP version 6 (IPv6) are covered in nauseating detail in other guides.

OSI Model - The Data-Link Layer

While the Network layer is concerned with transporting data *between* networks, the **Data-Link layer (Layer-2)** is responsible for transporting data *within* a network.

The Data-Link layer consists of two sublayers:

- **Logical Link Control (LLC)** sublayer
- **Media Access Control (MAC)** sublayer

The LLC sublayer serves as the intermediary between the physical link and all higher layer protocols. It ensures that protocols like IP can function regardless of what type of physical technology is being used.

Additionally, the LLC sublayer can perform flow-control and error-checking, though such functions are often provided by Transport layer protocols, such as TCP.

The MAC sublayer controls access to the physical medium, serving as mediator if multiple devices are competing for the same physical link. Data-link layer technologies have various methods of accomplishing this - **Ethernet** uses *Carrier Sense Multiple Access* with *Collision Detection (CSMA/CD)*, and **Token Ring** utilizes a *token*.

Ethernet is covered in great detail in another guide.

* * *

OSI Model - The Data-Link Layer (continued)

The Data-link layer *packages* the higher-layer data into **frames,** so that the data can be put onto the physical wire. This packaging process is referred to as **framing** or **encapsulation.**

The encapsulation type will vary depending on the underlying technology. Common Data-link layer technologies include following:

- Ethernet – the most common LAN data-link technology
- Token Ring – almost entirely deprecated
- FDDI (Fiber Distributed Data Interface)
- 802.11 Wireless
- Frame-Relay
- ATM (Asynchronous Transfer Mode)

The data-link frame contains the source and destination **hardware** (or **physical**) address. Hardware addresses uniquely identify a host within a network, and are often hardcoded onto physical network interfaces. However, hardware addresses contain no mechanism for differentiating one *network* from another, and can only identify a host *within* a network.

The most common hardware address is the Ethernet **MAC address.**

OSI Model - The Physical Layer

The **Physical layer (Layer-1)** controls the signaling and transferring of raw bits onto the physical medium. The Physical layer is closely related to the Data-link layer, as many technologies (such as Ethernet) contain both data-link and physical functions.

The Physical layer provides specifications for a variety of hardware:

- Cabling
- Connectors and transceivers
- Network interface cards (NICs)
- Wireless radios
- Hubs

Physical-layer devices and topologies are covered extensively in other guides.

* * *

Encapsulation and Layered Communication

As data is passed from the user application down the virtual layers of the OSI model, each layer 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 called **encapsulation**. Note that in the TCP/IP protocol suite only the lower layers perform encapsulation, generally.

For example, a Transport layer protocol such as TCP will add a header containing flow control, port numbers, and sequencing. The Network layer header contains logical addressing information, and the Data-link header contains physical addressing and other hardware specific information.

The PDU of each layer is identified with a different term:

Layer	*PDU Name*
Application	**-**
Presentation	**-**
Session	**-**
Transport	**Segments**
Network	**Packets**
Data-Link	**Frames**
Physical	**Bits**

Each layer **communicates with the corresponding layer** on the receiving device. For example, on the sending device, source and destination hardware addressing is placed in a Data-link header. On the receiving device, that Data-link header is processed and stripped away (**decapsulated)** before being sent up to the Network and other upper layers.

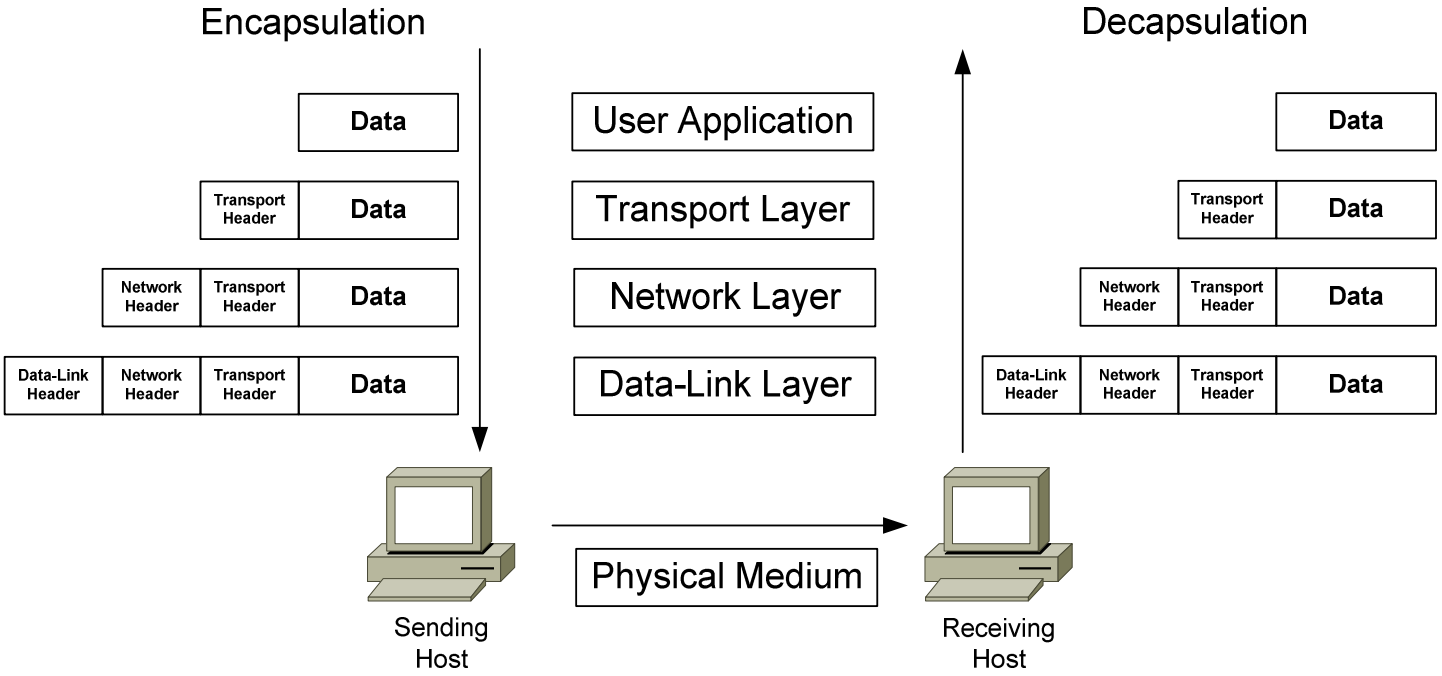
Network devices are commonly identified by the OSI layer they *operate* at; or, more specifically, what *header* or *PDU* the device processes.

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 Ethernet MAC addresses. 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.

* * *

Encapsulation Illustrated

The following illustrates how basic encapsulation occurs with the TCP/IP stack, which typically performs encapsulation only at the lower layers:



During **encapsulation** on the sending host**:**

- Data from the user application is handed off to the Transport layer.
- The Transport layer adds a header containing protocol-specific information, and then hands the *segment* to the Network layer.
- The Network layer adds a header containing source and destination logical addressing, and then hands the *packet* to the Data-Link layer.
- The Data-Link layer adds a header containing source and destination physical addressing and other hardware-specific information.
- The Data-Link *frame* is then handed off to the Physical layer to be transmitted on the network medium as *bits*.

During **decapsulation** on the receiving host, the reverse occurs:

- The frame is received from the physical medium.
- The Data-Link layer processes its header, strips it off, and then hands it off to the Network layer.
- The Network layer processes its header, strips it off, and then hands it off to the Transport layer.
- The Transport layer processes its header, strips it off, and then hands the data to the user application.

* * *

OSI Reference Model Example

A web browser serves as a good practical illustration of the OSI model and the TCP/IP protocol suite:

- The web browser serves as the user interface for accessing a website. The browser itself does not function at the **Application layer.** Instead, the web browser invokes the *Hyper Text Transfer Protocol (HTTP)* to interface with the remote web server, which is why *http://* precedes every web address.
- The Internet can provide data in a wide variety of *formats,* a function of the **Presentation layer**. Common formats on the Internet include *HTML, XML, PHP, GIF,* and *JPEG*. Any *encryption* or *compression* mechanisms used on a website are also considered a Presentation layer function.
- The **Session layer** is responsible for establishing, maintaining, and terminating the session between devices, and determining whether the communication is *half-duplex* or *full-duplex*. However, the TCP/IP stack generally does not include session-layer protocols, and is reliant on lower-layer protocols to perform these functions.
- HTTP utilizes the *TCP* **Transport layer** protocol to ensure the reliable delivery of data. TCP establishes and maintains a connection from the client to the web server, and packages the higher-layer data into *segments.* A sequence number is assigned to each segment so that data can be reassembled upon arrival.
- The best path to *route* the data between the client and the web server is determined by *IP*, a **Network layer** protocol. IP is also responsible for the assigned logical addresses on the client and server, and for encapsulating segments into *packets.*
- Data cannot be sent directly to a logical address. As packets travel from network to network, IP addresses are translated to *hardware* addresses, which are a function of the **Data-Link layer**. The packets are encapsulated into *frames* to be placed onto the physical medium.
- The data is finally transferred onto the network medium at the **Physical layer**, in the form of raw bits. Signaling and encoding mechanisms are defined at this layer, as is the hardware that forms the physical connection between the client and the web server.

* * *

IP and the DoD Model

The **Internet Protocol (IP)** was originally developed by the Department of Defense (DoD), and was a cornerstone for a group of protocols that became known as the **TCP/IP protocol suite**.

The DoD developed their own networking model, which became known as the **DoD** or **TCP/IP Model**. It consists of four layers:

	OSI Model		***DoD Model***
7	Application		
6	Presentation	4	Application
5	Session		
4	Transport	3	Host-to-Host
3	Network	2	Internet
2	Data-link	1	Network Access
1	Physical		

The consolidated DoD model is generally regarded as more practical than the OSI model. Upper layer protocols often provide services that span the top three layers. A converged Data-link and Physical layer is also sensible, as many technologies provide specifications for both layers, such as Ethernet.

The following chart illustrates where common protocols fit into the DoD model:

Layer	***Example Protocols***
Application	FTP, HTTP, SMTP
Host-to-Host	TCP, UDP
Internet	IP
Network Access	Ethernet

Despite the practicality of the DoD model, the OSI model is still the basis for most network terminology.

So, Please Do Not Throw Sausage Pizza Away. ☺

* * *

Section 3
- 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

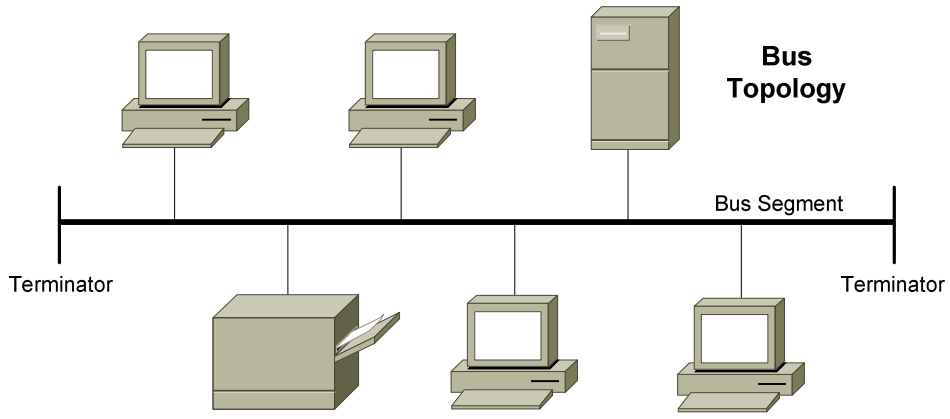
A **topology** defines both the *physical* and *logical* structure of a network. Topologies come in a variety of configurations, including:

- 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:

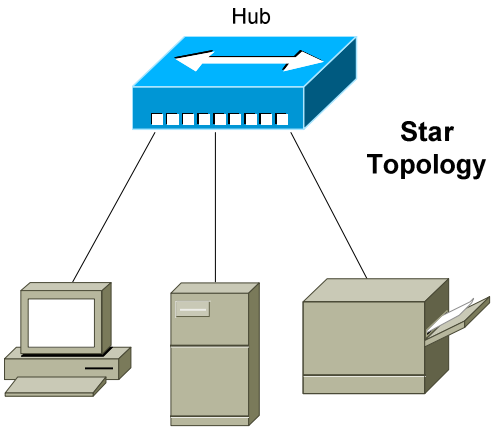
- Both ends of the bus must be **terminated,** otherwise a signal will *reflect* back and cause interference, severely degrading performance.
- Adding or removing hosts to the bus can be difficult.
- The bus represents a single point of failure - a break in the bus will affect *all* hosts on the segment.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

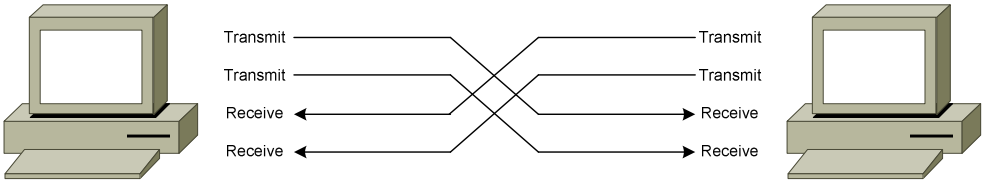
The wires themselves are assigned a *color* to distinguish them. The color is dictated by the cabling standard - **TIA/EIA-568B** is the current standard:

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 4
- 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).

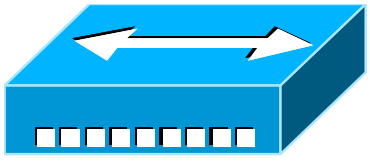
For example, **switches** are generally identified as Layer-2 devices, as switches process information stored in the **Data-Link** header of a frame (such as MAC addresses in Ethernet). Similarly, **routers** are identified as Layer-3 devices, as routers process *logical* addressing information in the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

* * *

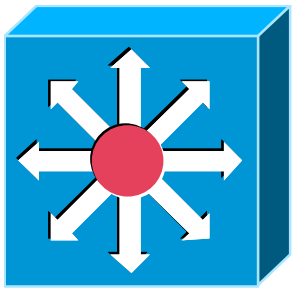
Icons for Network Devices

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

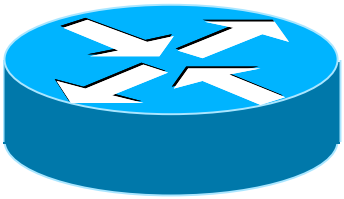


Hub

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

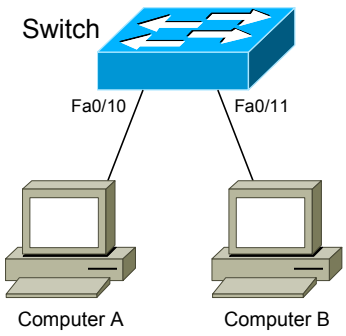
Using this information, Layer-2 devices will make intelligent **forwarding** decisions based on the frame (or data-link) headers. A frame can then be forwarded out *only* the appropriate destination port, instead of *all* ports.

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.**

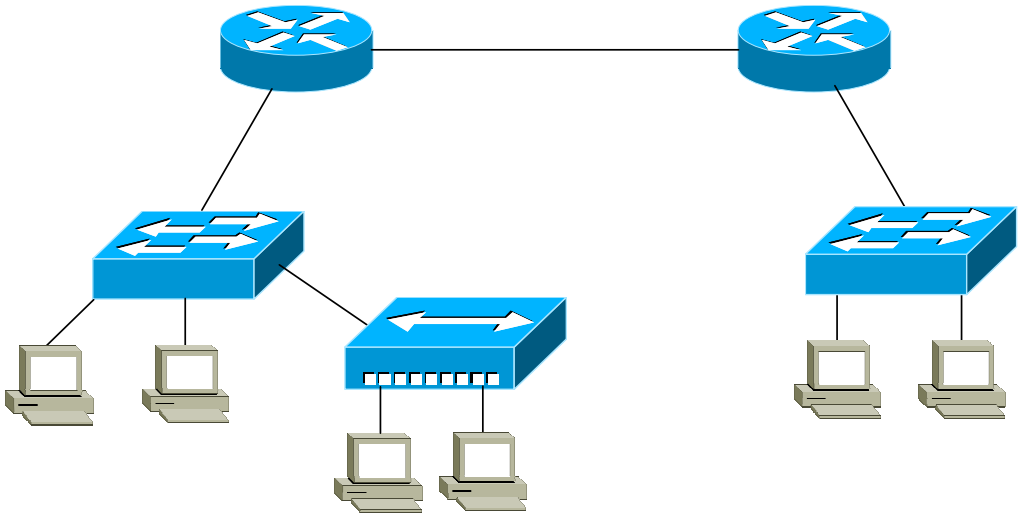
Unlike Layer-2 switches, Layer-3 routers also **separate broadcast domains**. As a rule, a router **will never forward broadcasts** from one network to another network (unless, of course, you explicitly configure it to). ☺

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.

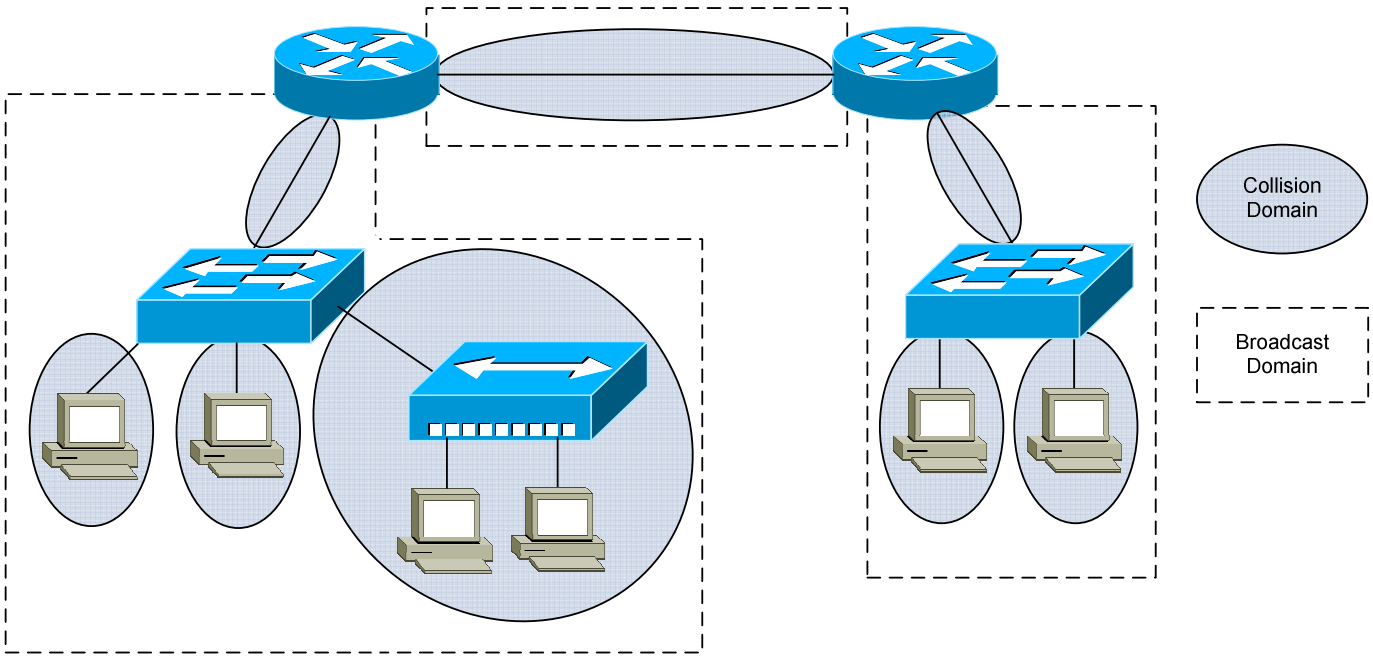
Collision vs. Broadcast Domain Example



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:

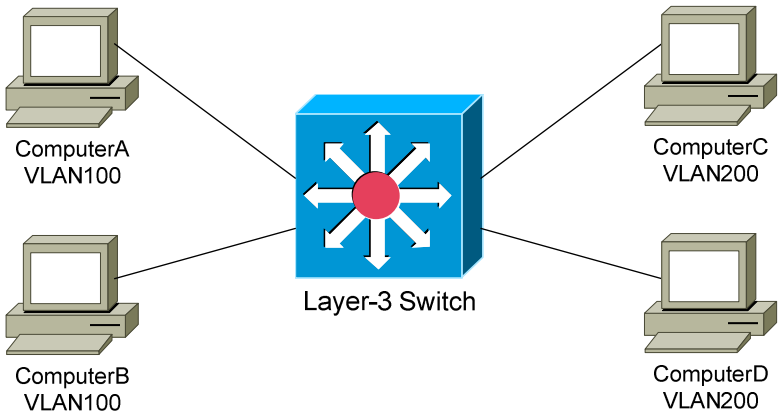
- The switch must be capable of making Layer-3 forwarding decisions (traditionally referred to as routing).
- The switch must cache network traffic flows, so that Layer-3 forwarding can occur in hardware.

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)

* * *

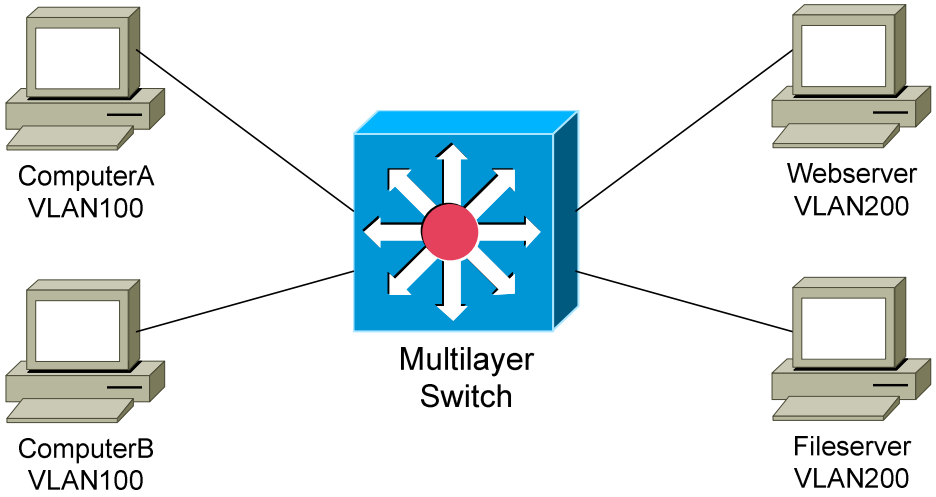
Multilayer Switching

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.**

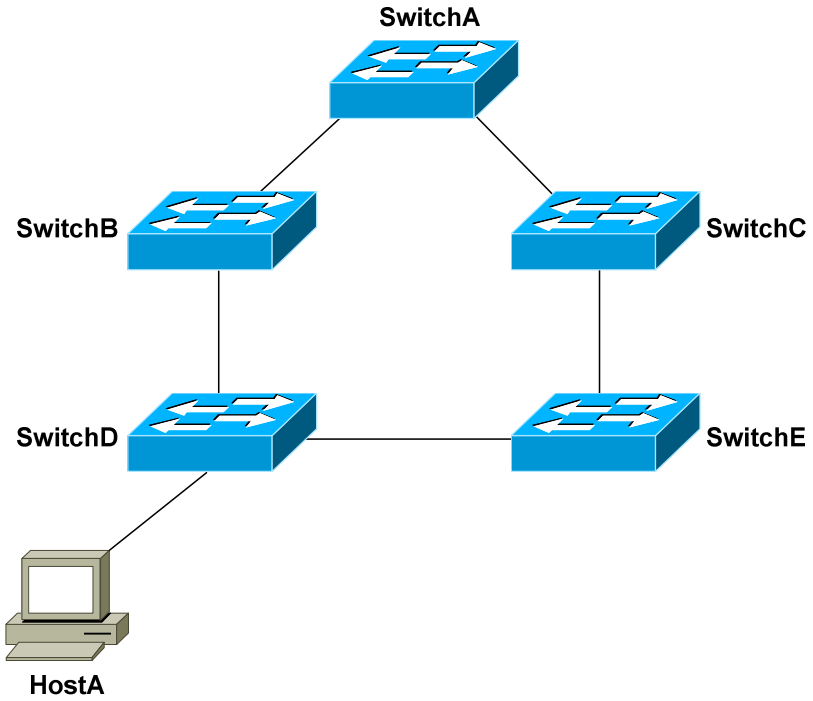
Section 5
- Spanning Tree Protocol -

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

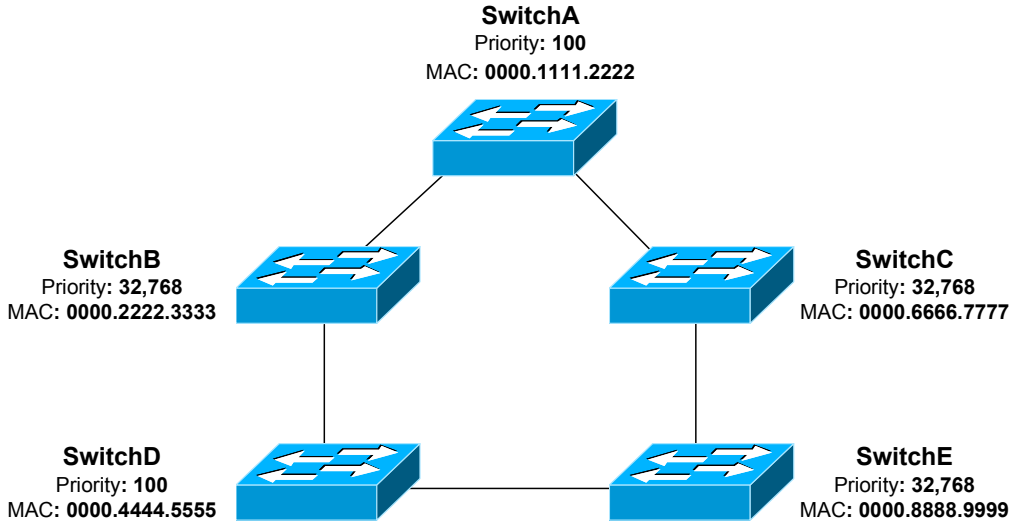
The first step in the STP convergence process is electing a **Root Bridge**, which is the central reference point for the STP topology. As a best practice, the Root Bridge should be the most centralized switch in the STP topology.

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

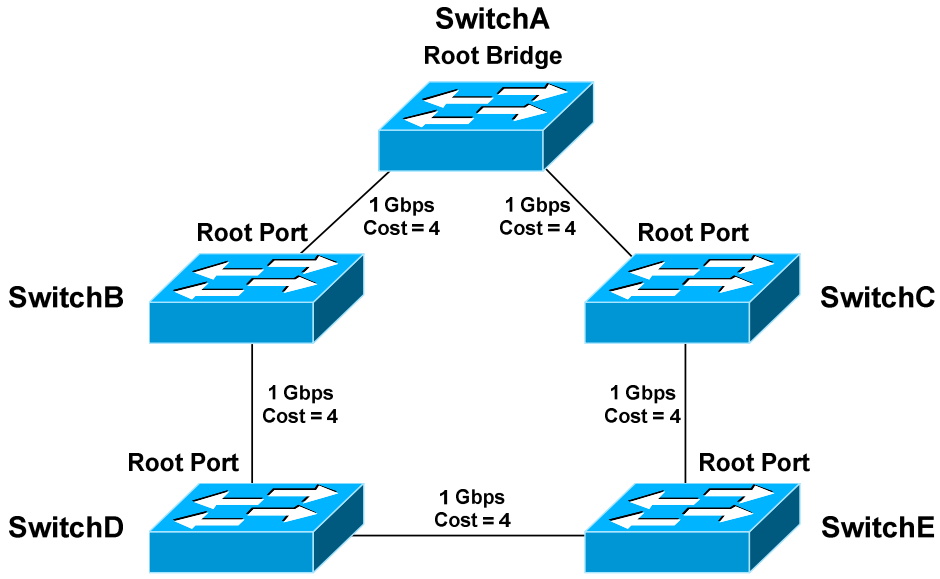
The second step in the STP convergence process is to identify **root ports.** The root port of each switch has the lowest **root path cost** to get to the Root Bridge.

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:



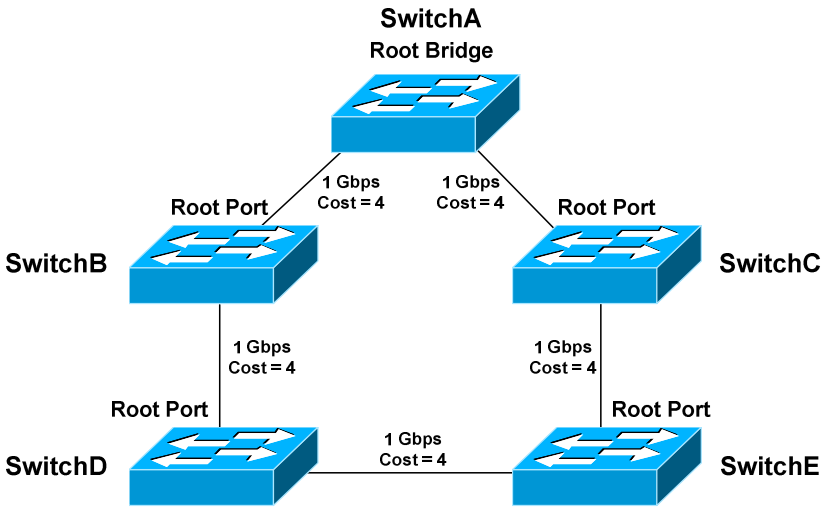
Each 1Gbps link has a path cost of *4*. SwitchA has a cumulative path cost of *0*, because it is the Root Bridge. Thus, when SwitchA sends out BPDU's, it advertises a root path cost of 0.

* * *

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

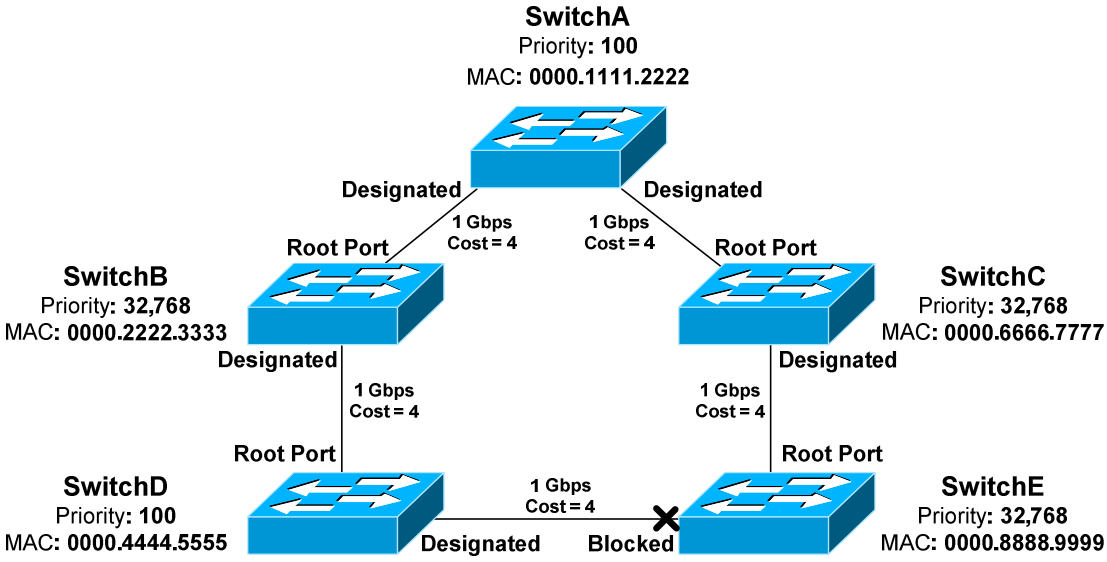
The third step in the STP convergence process is to identify **designated ports.** A single designated port is identified for each network *segment.* This port is responsible for forwarding BPDUs and frames to that segment.

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.

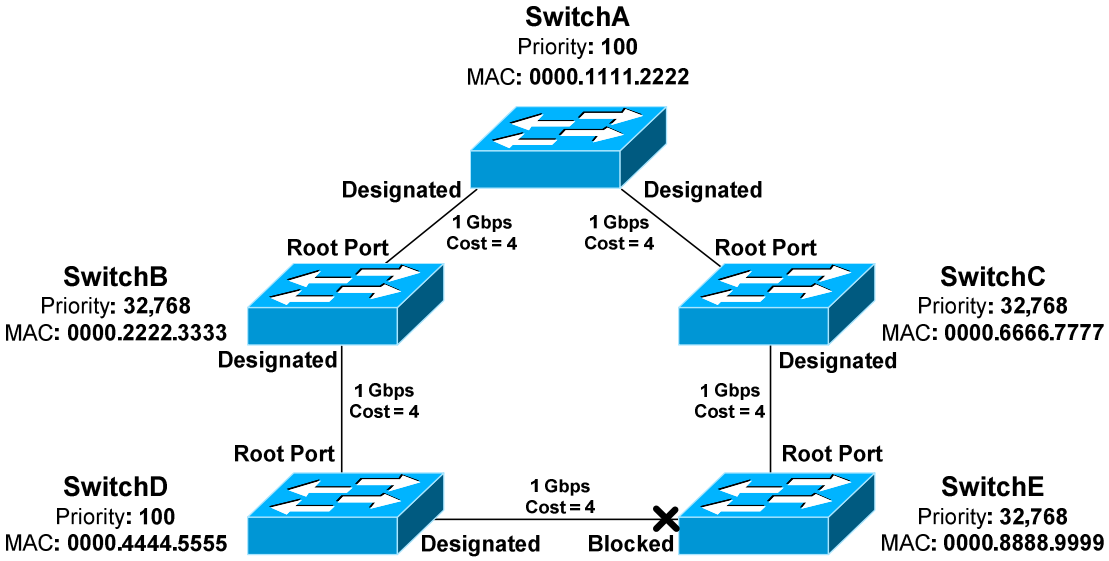
Remember, every network segment **must have one designated port**, regardless if a root port already exists on that segment.

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.

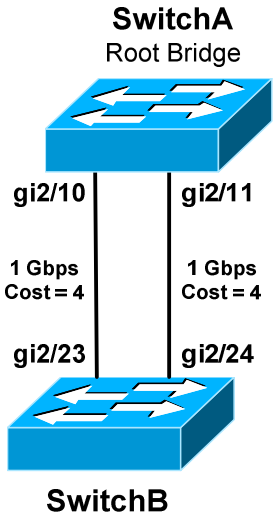
As with electing the Root Bridge, if there is a tie in priority, the **lowest MAC address** is used as the tie breaker.

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
```

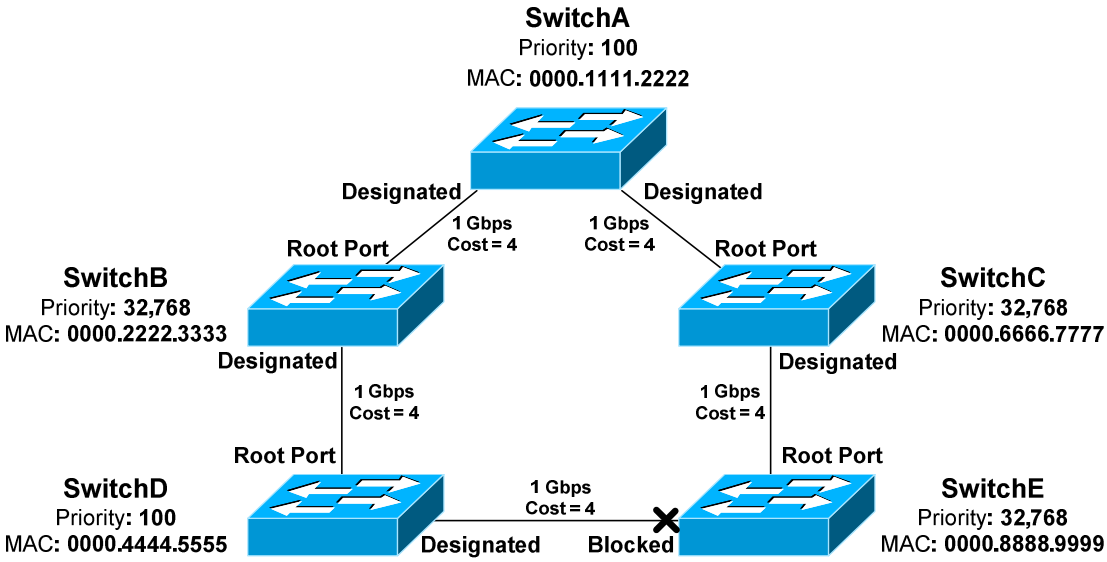
* * *

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.

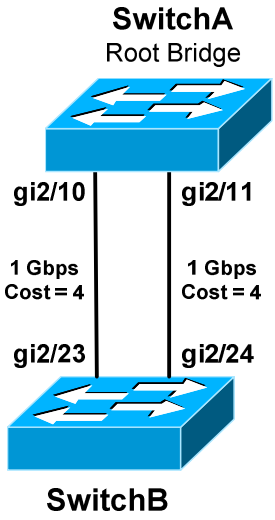
* * *

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*

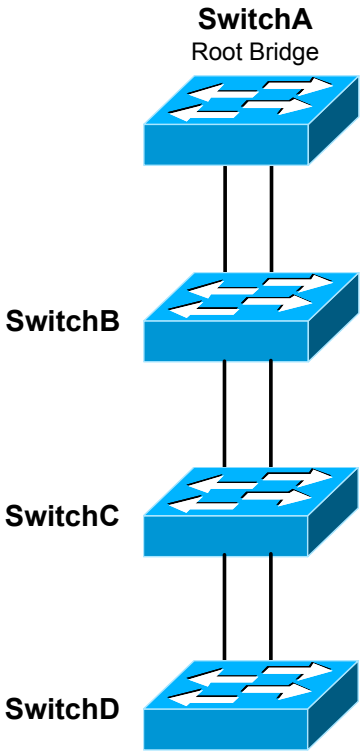
The timer values are measured in seconds, and the above represents the *maximum* possible value for each timer.

* * *

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**

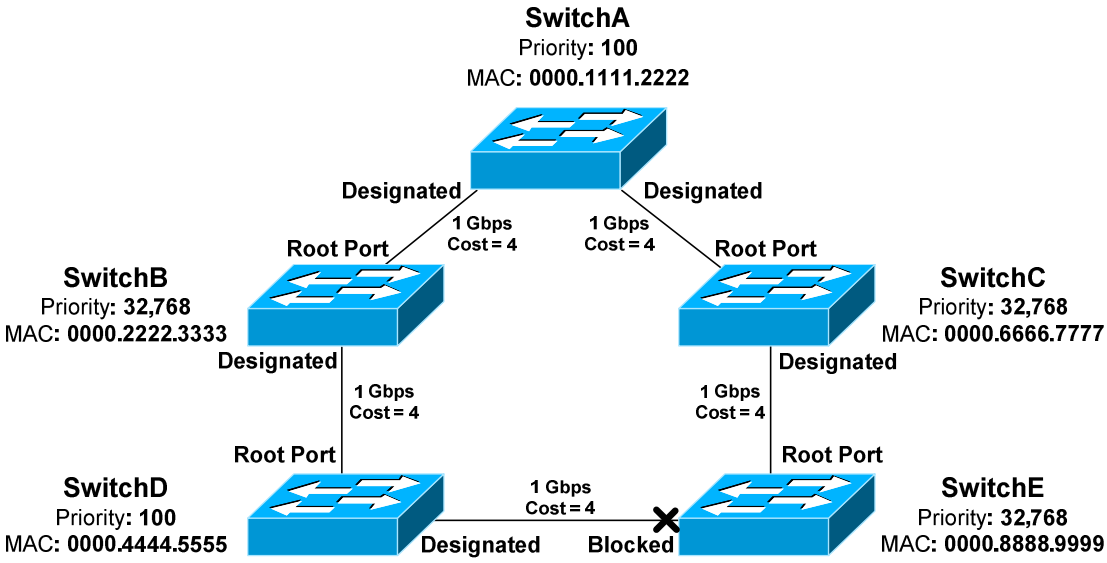
Configuration BPDUs are used to elect Root Bridges, root ports, and designated ports.

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:



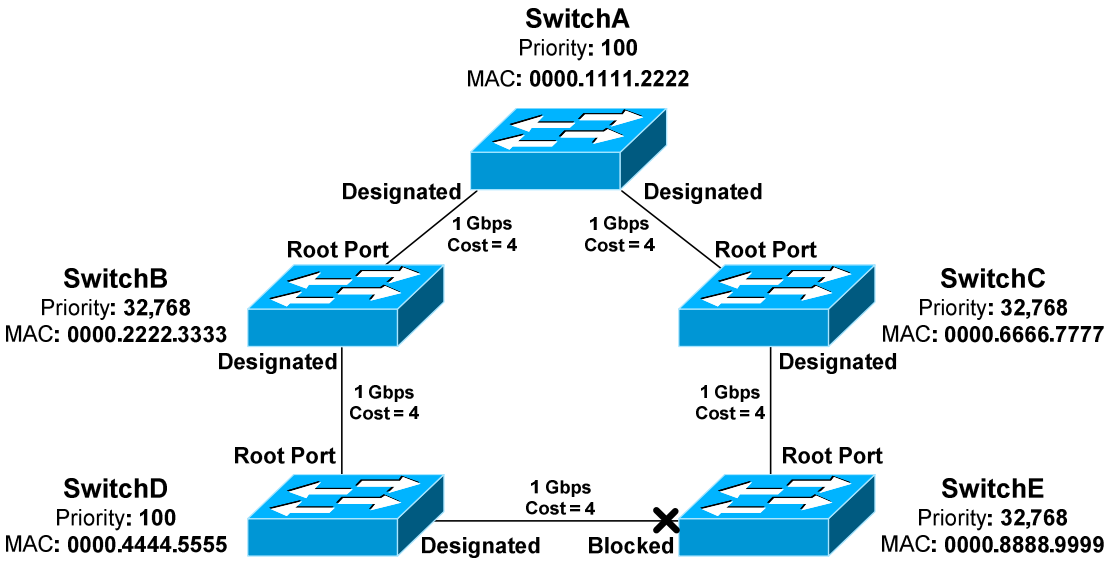
If the port on SwitchD connecting to SwitchE went down:

- 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

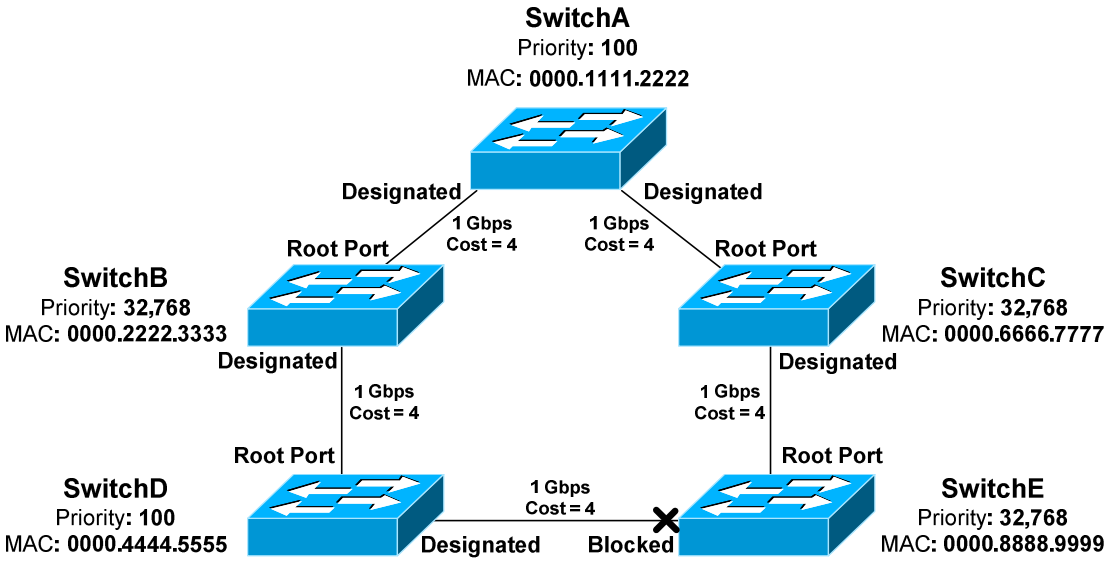
For example, if the root port on SwitchB fails, SwitchB would consider this a **direct failure.** SwitchB will detect immediately that the physical port is down, and STP will react accordingly.

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.

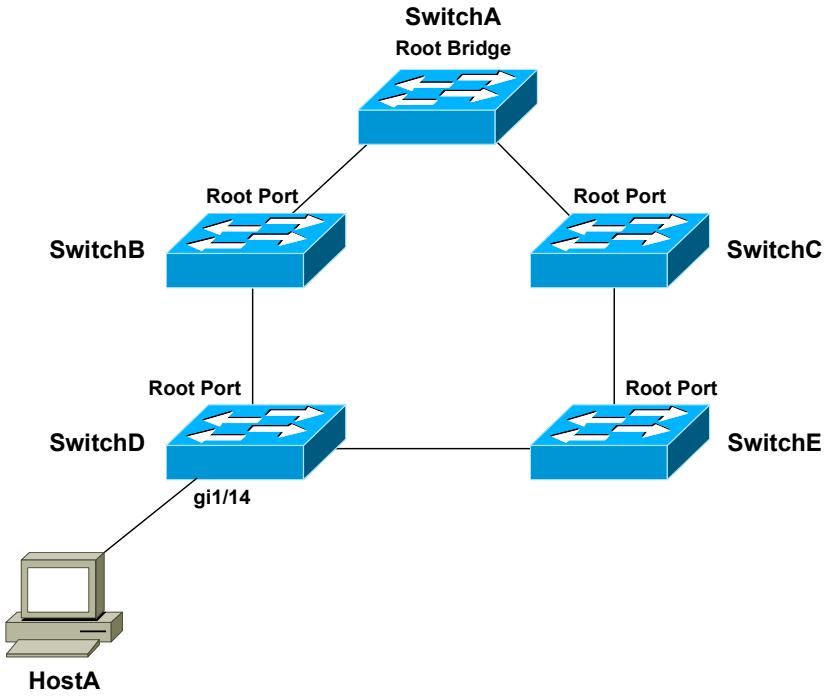
PortFast allows a switch port to bypass the usual progression of STP states. The port will instead transition from a *blocking* to a *forwarding* state **immediately**, eliminating the typical 30 second delay.

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*

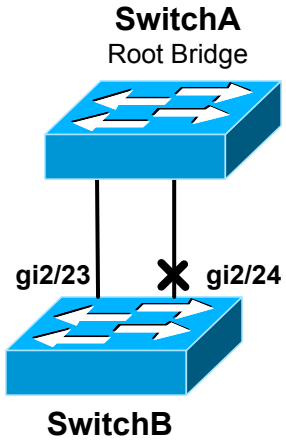
PortFast can also be globally enabled for all interfaces:

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*

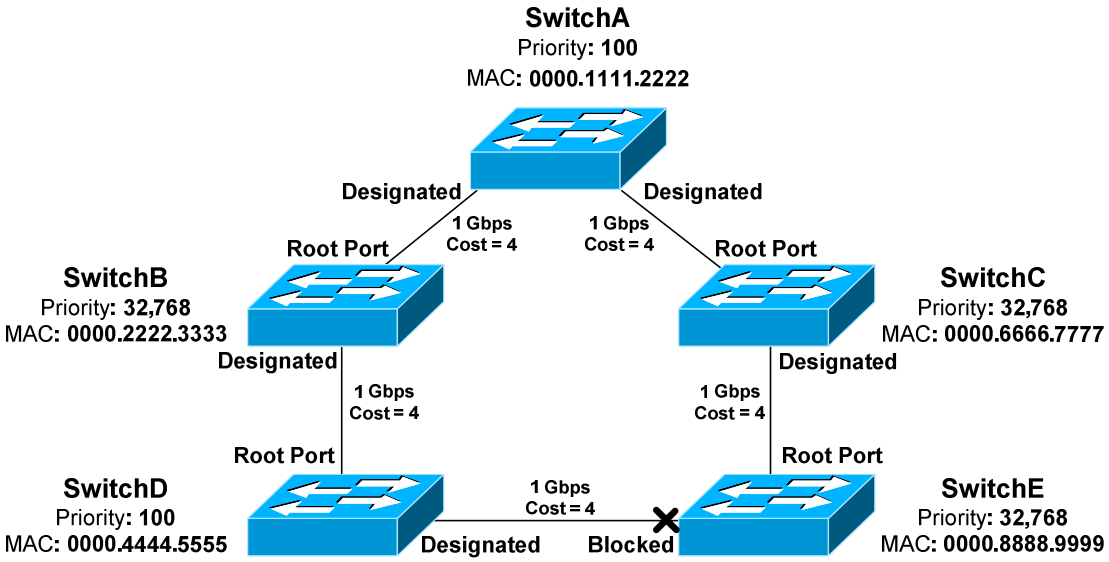
UplinkFast functions by tracking all possible links *to* the Root Bridge. Thus, UplinkFast is **not supported** on the Root Bridge. In fact, enabling this feature will automatically increase a switch's bridge priority to 49,152.

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*

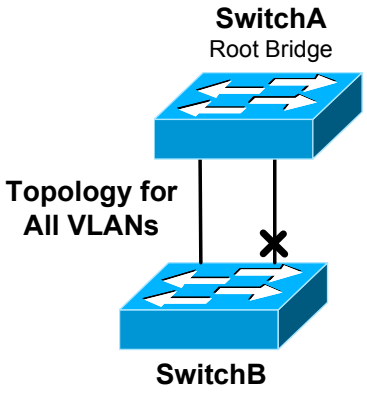
```
Vlan                 Role Sts Cost      Prio.Nbr Type
-------------------- ---- --- --------- -------- -----------------
VLAN0101             Root FWD 4         128.48   P2p
VLAN0102             Altn BLK 4         128.48   P2p
```

* * *

Per-VLAN Spanning Tree (PVST) Load Balancing

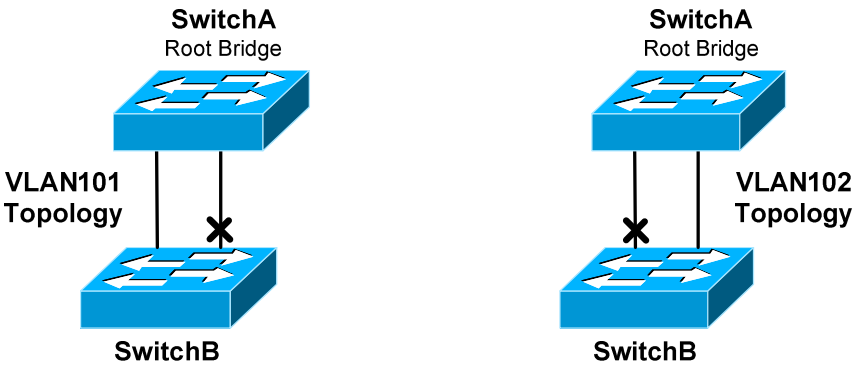
PVST and **PVST+** employ a *separate* STP instance for *each* VLAN. This provides superior flexibility over CST, which only supports a single STP instance for *all* VLANs.

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 6
- IPv4 Addressing and Subnetting -

Hardware Addressing

A **hardware address** is used to uniquely identify a host *within* a local network. Hardware addressing is a function of the **Data-Link** layer of the OSI model (Layer-2).

Ethernet utilizes the 48-bit **MAC address** as its hardware address. The MAC address is often hardcoded on physical network interfaces, though some interfaces support changing the MAC address using special utilities. In virtualization environments, dynamically assigning MAC addresses is very common.

A MAC address is most often represented in **hexadecimal,** using one of two accepted formats:

00:43:AB:F2:32:13
0043.ABF2.3213

The first six hexadecimal digits of a MAC address identify the *manufacturer* of the physical network interface. This is referred to as the **OUI (Organizational Unique Identifier).** The last six digits uniquely identify the host itself, and are referred to as the **host ID**.

The MAC address has one shortcoming – it contains no *hierarchy.* MAC addresses provide no mechanism to create **boundaries** between networks. There is no method to distinguish one network from another.

This lack of hierarchy poses *significant* difficulties to network scalability. If *only* Layer-2 hardware addressing existed, all hosts would technically exist on the *same* network. 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 single Layer-2 switched network. Switches, as a rule, will forward a broadcast out *every* port. With billions of hosts on the Internet, the resulting broadcast storms would be devastating. The Internet would simply collapse.

The scalability limitations of Layer-2 hardware addresses are mitigated using **logical addresses,** covered in great detail in this guide.

* * *

Logical Addressing

Logical addressing is a function of the **Network** layer of the OSI Model (Layer-3), and provides a hierarchical structure to separate networks. Logical addresses are never hardcoded on physical network interfaces, and can be dynamically assigned and changed freely.

A logical address contains two components:

- **Network ID** – identifies which network a host belongs to.
- **Host ID** – uniquely identifies the host on that network.

Examples of logical addressing protocols include **Internetwork Packet Exchange (IPX)** and **Internet Protocol (IP)**. IPX was predominantly used on Novell networks, but is now almost entirely deprecated. **IP** is the most widely-used logical address, and is the backbone protocol of the Internet.

Internet Protocol (IP)

In the 1970's, the Department of Defense developed the **Transmission Control Protocol (TCP)**, to provide both Network and Transport layer functions. When this proved to be an inflexible solution, those functions were separated - with the **Internet Protocol (IP)** providing Network layer services, and TCP providing Transport layer services.

Together, TCP and IP provide the core functionality for the **TCP/IP** or **Internet protocol suite**.

IP provides two fundamental Network layer services:

- **Logical addressing** – provides a unique address that identifies both the *host*, and the *network* that host exists on.
- **Routing** – determines the *best path* to a particular destination network, and then *routes* data accordingly.

IP was originally defined in RFC 760, and has been revised several times. IP Version 4 (**IPv4**) was the first version to experience widespread deployment, and is defined in RFC 791. IPv4 will be the focus of this guide.

IPv4 employs a **32-bit address**, which limits the number of possible addresses to 4,294,967,296. IPv4 will eventually be replaced by IP Version 6 (**IPv6**), due to a shortage of available IPv4 addresses. IPv6 is covered in great detail in another guide.

* * *

IPv4 Addressing

A core function of IP is to provide logical addressing for hosts. An **IP address** provides a hierarchical structure to both uniquely identify a *host*, and what *network* that host exists on.

An IP address is most often represented in **decimal,** in the following format:

158.80.164.3

An IP address is comprised of four **octets,** separated by periods:

First Octet	Second Octet	Third Octet	Fourth Octet
158	80	164	3

Each octet is an **8-bit** number, resulting in a **32-bit IP address**. The smallest possible value of an octet is *0,* or *00000000* in binary. The largest possible value of an octet is *255*, or *11111111* in binary.

The above IP address represented in binary would look as follows:

First Octet	Second Octet	Third Octet	Fourth Octet
10011110	01010000	10100100	00000011

Decimal to Binary Conversion

The simplest method of converting between decimal and binary is to remember the following table:

128	64	32	16	8	4	2	1

To convert a decimal number of *172* to binary, start with the leftmost column. Since *172* is greater than *128*, that binary bit will be set to *1*. Next, add the value of the next column (*128 + 64 = 192)*. Since *172* is less than *192,* that binary bit will be set to *0.*

Again, add the value of the next column (*128 + 32 = 160)*. Since *172* is greater than *160*, that binary bit will be set to *1*. Continue this process until the columns with binary bits set to *1* add up to 172:

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

* * *

Binary to Decimal Conversion

Converting from binary back to decimal is even simpler. Apply the binary number to the conversion table, and then add up any columns with binary bits set to 1.

For example, consider the binary number of *11110001:*

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

By adding *128 + 64 + 32 + 16+ 1,* it can be determined that *11110001* equals *241.*

The Subnet Mask

Part of an IP address identifies the *network.* The other part of the address identifies the *host.* A **subnet mask** is required to provide this distinction**:**

158.80.164.3 255.255.0.0

The above IP address has a subnet mask of *255.255.0.0*. The subnet mask follows two rules:

- If a binary bit is set to a **1** (or *on*) in a subnet mask, the corresponding bit in the address identifies the **network**.
- If a binary bit is set to a **0** (or *off*) in a subnet mask, the corresponding bit in the address identifies the **host**.

Looking at the above address and subnet mask in binary:

IP Address: 10011110.01010000.10100100.00000011
Subnet Mask: 11111111.11111111.00000000.00000000

The first 16 bits of the subnet mask are set to *1*. Thus, the first 16 bits of the address (*158.80*) identify the *network*. The last 16 bits of the subnet mask are set to *0*. Thus, the last 16 bits of the address (*164.3*) identify the unique *host* on that network.

The network portion of the subnet mask must be **contiguous**. For example, a subnet mask of *255.0.0.255* is not valid.

* * *

The Subnet Mask (continued)

Hosts on the same logical network will have *identical* network addresses, and can communicate freely. For example, the following two hosts are on the same network:

Host A: 158.80.164.100 255.255.0.0
Host B: 158.80.164.101 255.255.0.0

Both share the same network address (*158.80*), which is determined by the *255.255.0.0* subnet mask. Hosts that are on *different* networks cannot communicate without an intermediating device. For example:

Host A: 158.80.164.100 255.255.0.0
Host B: 158.85.164.101 255.255.0.0

The subnet mask has remained the same, but the network addresses are now different (*158.80* and *158.85* respectively). Thus, the two hosts are *not* on the same network, and cannot communicate without a **router** between them. **Routing** is the process of forwarding packets from one network to another.

Consider the following, trickier example:

Host A: 158.80.1.1 255.248.0.0
Host B: 158.79.1.1 255.248.0.0

The specified subnet mask is now *255.248.0.0*, which doesn't fall cleanly on an octet boundary. To determine if these hosts are on separate networks, first convert everything to binary:

Host A Address: 10011110.01010000.00000001.00000001
Host B Address: 10011110.01001111.00000001.00000001
Subnet Mask: 11111111.11111000.00000000.00000000

Remember, the **1** (or **on)** bits in the subnet mask identify the *network* portion of the address. In this example, the first *13 bits* (the 8 bits of the first octet, and the first 5 bits of the second octet) identify the network. Looking at only the first 13 bits of each address:

Host A Address: 10011110.01010
Host B Address: 10011110.01001

Clearly, the network addresses are *not* identical. Thus, these two hosts are on separate networks, and require a router to communicate.

* * *

IP Address Classes

The IPv4 address space has been structured into several **classes**. The value of the **first octet** of an address determines the class of the network:

Class	*First Octet Range*	*Default Subnet Mask*
Class A	1 - 127	255.0.0.0
Class B	128 - 191	255.255.0.0
Class C	192 - 223	255.255.255.0
Class D	224 - 239	-

Class A networks range from **1** to **127.** The *default* subnet mask is 255.0.0.0. Thus, by *default,* the first octet defines the network, and the last three octets define the host. This results in a maximum of **127** Class A networks, with **16,777,214** hosts per network!

Example of a Class A address:

Address:	64.32.254.100
Subnet Mask:	255.0.0.0

Class B networks range from **128** to **191**. The *default* subnet mask is 255.255.0.0. Thus, by *default,* the first two octets define the network, and the last two octets define the host. This results in a maximum of **16,384** Class B networks, with **65,534** hosts per network.

Example of a Class B address:

Address:	152.41.12.195
Subnet Mask:	255.255.0.0

Class C networks range from **192** to **223.** The *default* subnet mask is 255.255.255.0. Thus, by *default,* the first three octets define the network, and the last octet defines the host. This results in a maximum of **2,097,152** Class C networks, with **254** hosts per network.

Example of a Class C address:

Address:	207.79.233.6
Subnet Mask:	255.255.255.0

Class D networks are reserved for **multicast** traffic. Class D addresses do not use a subnet mask.

* * *

CIDR (Classless Inter-Domain Routing)

Classless Inter-Domain Routing (CIDR) is a simplified method of representing a subnet mask. CIDR identifies the number of binary bits set to a **1** (or *on*) in a subnet mask, preceded by a slash.

For example, a subnet mask of *255.255.255.240* would be represented as follows in binary:

11111111.11111111.11111111.11110000

The first 28 bits of the above subnet mask are set to *1*. The CIDR notation for this subnet mask would thus be ***/28***.

The CIDR mask is often appended to the IP address. For example, an IP address of *192.168.1.1* and a subnet mask of *255.255.255.0* would be represented as follows using CIDR notation:

192.168.1.1 /24

Address Classes vs. Subnet Mask

Remember the following three rules:

- The **first octet** on an address dictates the *class* of that address.
- The **subnet mask** determines what part of an address identifies the *network*, and what part identifies the *host*.
- Each class has a ***default*** subnet mask. A network using its default subnet mask is referred to as a **classful network**.

For example, *10.1.1.1* is a Class A address, and its default subnet mask is *255.0.0.0* (*/8* in CIDR).

It is entirely possible to use subnet masks *other* than the default. For example, a Class B subnet mask can be applied to a Class A address:

10.1.1.1 /16

However, **this does not change the class of the above address**. It remains a Class A *address*, which has been subnetted using a Class B *mask*.

Remember, the ***only*** thing that determines the class of an IP address is the first octet of that address. Likewise, the subnet mask is the ***only*** thing that determines what part of an address identifies the network, and what part identifies the host.

* * *

Subnet and Broadcast Addresses

On *each* IP network, two host addresses are reserved for special use:

- The **subnet** (or **network**) address
- The **broadcast** address

Neither of these addresses can be assigned to an actual host.

The **subnet** address is used to identify **the network itself**. A routing table contains a list of known networks, and each network is identified by its subnet address. Subnet addresses contain **all 0 bits in the host portion** of the address.

For example, *192.168.1.0/24* is a subnet address. This can be determined by looking at the address and subnet mask in binary:

IP Address:	11000000.10101000.00000001.00000000
Subnet Mask:	11111111.11111111.11111111.00000000

Note that all host bits in the address are set to *0*.

The **broadcast** address identifies *all* hosts on a particular network. A packet sent to the broadcast address will be received and processed by every host on that network. Broadcast addresses contain **all 1 bits in the host portion** of the address.

For example, *192.168.1.255/24* is a broadcast address. Note that all host bits are set to *1:*

IP Address:	11000000.10101000.00000001.11111111
Subnet Mask:	11111111.11111111.11111111.00000000

Broadcasts are one of three types of IP packets:

- **Unicasts** are packets sent from one host to one other host
- **Multicasts** are packets sent from one host to a *group* of hosts
- **Broadcasts** are packets sent from one host to all other hosts on the local network

A router, by default, will **never forward** a multicast or broadcast packet from one interface to another.

A switch, by default, will forward a multicast or broadcast packet **out every port**, except for the port that originated the multicast or broadcast.

* * *

Subnetting

Subnetting is the process of creating new networks (or *subnets)* by **stealing bits** from the host portion of a subnet mask. There is one caveat: stealing bits from hosts creates **more** networks but **fewer** hosts per network.

Consider the following Class C network:

192.168.254.0

The default subnet mask for this network is 255.255.255.0. This single network can be segmented, or *subnetted*, into multiple networks. For example, assume a minimum of *10* new networks are required. Resolving this is possible using the following magical formula:

2^n

The exponent '**n'** identifies the number of bits to steal from the host portion of the subnet mask. The default Class C mask (255.255.255.0) looks as follows in binary:

11111111.1111111.1111111.00000000

There are a total of 24 bits set to *1*, which are used to identify the network. There are a total of 8 bits set to *0*, which are used to identify the host, and these host bits can be *stolen*.

Stealing bits essentially involves changing host bits (set to *0* or *off*) in the subnet mask to network bits (set to *1* or *on*). Remember, network bits in a subnet mask **must always be contiguous** - skipping bits is not allowed.

Consider the result if three bits are stolen. Using the above formula:

2^n = 2^3 = 8 = **8 new networks created**

However, a total of 8 new networks *does not* meet the original requirement of at least 10 networks. Consider the result if four bits are stolen:

2^n = 2^4 = 16 = **16 new networks created**

A total of 16 new networks *does* meet the original requirement. Stealing four host bits results in the following *new* subnet mask:

11111111.11111111.11111111.11110000 = 255.255.255.240

* * *

Subnetting (continued)

In the previous example, a Class C network was subnetted to create *16* new networks, using a subnet mask of *255.255.255.240* (or */28* in CIDR). Four bits were stolen in the subnet mask, leaving only four bits for hosts.

To determine the number of hosts this results in, for each of the new 16 networks, a slightly modified formula is required:

$$2^n - 2$$

Consider the result if four bits are available for hosts:

$2^n - 2 \quad = \quad 2^4 - 2 \quad = \quad 16 - 2 \quad =$ **14 usable hosts per network**

Thus, subnetting a Class C network with a /28 mask creates 16 new networks, with 14 usable hosts per network.

Why is the formula for calculating usable hosts $2^n - 2$? Because it is **never possible** to assign a host an address with all *0* or all *1* bits in the *host* portion of the address**.** These are reserved for the subnet and broadcast addresses, respectively. Thus, every time a network is subnetted, useable host addresses are lost.

The 2^n-2 Rule and Subnetted Networks

To avoid confusion, it was historically unacceptable to use the first and last new *networks* created when subnetting, as it is possible for a classful network to have the same subnet and broadcast address as its subnetted networks. This required the $2^n - 2$ formula to also be used when calculating the number of new *networks* created while subnetting.

However, this is **no longer a restriction** for modern equipment and routing protocols. Specifically, on Cisco IOS devices, the following command is now enabled by default:

Router(config)# *ip subnet-zero*

The *ip subnet-zero* commands allows for the use of networks with all ***0*** or all ***1* bits** in the *stolen* network portion of the address. Thus, the formula for calculating the number of new networks created is simply $\mathbf{2^n}$.

Remember though, the formula for calculating usable ***hosts*** **is always $\mathbf{2^n - 2}$**.

(Reference: http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080093f18.shtml)

* * *

Determining the Range of Subnetted Networks

Determining the *range* of the newly created networks can be accomplished using several methods. The *long* method involves some binary magic.

Consider the example *192.168.254.0* network again, which was subnetted using a *255.255.255.240* mask:

192.168.254.0: 11000000.10101000.11111110.00000000
255.255.255.240: 11111111.11111111.11111111.11110000

Subnetting stole four bits in the fourth octet, creating a total of *16* new networks. Looking at *only* the fourth octet, the first newly created network is *0000*. The second new network is *0001*. Calculating all possible permutations of the four stolen bits:

Binary	***Decimal***	***Binary***	***Decimal***	***Binary***	***Decimal***
.0000 *xxxx*	.0	.0110 *xxxx*	.96	.1100 *xxxx*	.192
.0001 *xxxx*	.16	.0111 *xxxx*	.112	.1101 *xxxx*	.208
.0010 *xxxx*	.32	.1000 *xxxx*	.128	.1110 *xxxx*	.224
.0011 *xxxx*	.48	.1001 *xxxx*	.144	.1111 *xxxx*	.240
.0100 *xxxx*	.64	.1010 *xxxx*	.160		
.0101 *xxxx*	.80	.1011 *xxxx*	.176		

Note that this equates to exactly *16* new networks. The decimal value represents the first (or the *subnet*) address of each newly created network. To determine the range for the hosts of the *first* new network:

Binary	***Decimal***	***Binary***	***Decimal***	***Binary***	***Decimal***
.0000 0000	.0	.0000 0110	.6	.0000 1100	.12
.0000 0001	.1	.0000 0111	.7	.0000 1101	.13
.0000 0010	.2	.0000 1000	.8	.0000 1110	.14
.0000 0011	.3	.0000 1001	.9	.0000 1111	.15
.0000 0100	.4	.0000 1010	.10		
.0000 0101	.5	.0000 1011	.11		

The binary value has been split to emphasize the separation of the stolen *network* bits from the *host* bits. The first address has all *0* bits in the host portion (*0000*), and is the **subnet address** for this network. The last address has all *1* bits in the host portion, and thus is the **broadcast address** for this network. Note that there are exactly **14 usable addresses** to assign to hosts.

* * *

Determining the Range of Subnetted Networks (continued)

Calculating the ranges of subnetted networks can quickly become tedious when using the long binary method. The *shortcut* method involves taking the subnet mask (*255.255.255.240* from the previous example), and subtracting the subnetted octet (*240*) from *256*.

$$256 - 240 = 16$$

Assuming *ip subnet-zero* is enabled, the first network will begin at *0*. Then, simply continue adding *16* to identify the first address of each new network:

0 16 32 48 64 80 96 112 128 144 160 176 192 208 224 240

Knowing the *first* address of each new network makes it simple to determine the *last* address of each network:

First address of network	0	16	32	48	64	80	96	112	128	144
Last address of network	15	31	47	63	79	95	111	127	143	159

Only the first 10 networks were calculated, for brevity. The first address of each network becomes the **subnet address** for that network**.** The last address of each network becomes the **broadcast address** for that network**.**

Once the first and last address of each network is known, determining the usable range for hosts is straightforward:

Subnet address	0	16	32	48	64	80	96	112	128	144
	1	17	33	49	65	81	97	113	129	145
Usable Range	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
	14	30	46	62	78	94	110	126	142	158
Broadcast address	15	31	47	63	79	95	111	127	143	159

Hosts on the same network (such as *192.168.254.2* and *192.168.254.14*) can communicate freely.

Hosts on different networks (such as *192.168.254.61* and *192.168.254.66*) require a router to communicate.

* * *

Class A Subnetting Example

Consider the following subnetted Class A network: 10.0.0.0 255.255.248.0

Now consider the following questions:

- How many new networks were created?
- How many usable hosts are there per network?
- What is the full range of the first three networks?

By default, the *10.0.0.0* network has a subnet mask of *255.0.0.0*. To determine the number of bits stolen:

255.0.0.0:	11111111.00000000.00000000.00000000
255.255.248.0:	11111111.11111111.11111000.00000000

Clearly, **13 bits** have been stolen to create the new subnet mask. To calculate the total number of new networks:

2^n = 2^{13} = **8192 new networks created**

There are clearly **11 bits** remaining in the host portion of the mask:

$2^n - 2$ = $2^{11} - 2$ = 2048 – 2 = **2046 usable hosts per network**

Calculating the ranges is a bit tricky. Using the shortcut method, subtract the third octet (*248)* of the subnet mask (*255.255.248.0*) from *256.*

256 – 248 = 8

The first network will begin at *0*, again. **However**, the ranges are spread across multiple octets. The ranges of the first three networks look as follows:

Subnet address	10.0.0.0	10.0.8.0	10.0.16.0
Usable Range	10.0.0.1 ↕ 10.0.7.254	10.0.8.1 ↕ 10.0.15.254	10.0.16.1 ↕ 10.0.23.254
Broadcast address	10.0.7.255	10.0.15.255	10.0.23.255

* * *

Private vs. Public IPv4 Addresses

The rapid growth of the Internet resulted in a shortage of available IPv4 addresses. In response, a specific subset of the IPv4 address space was designated as *private,* to temporarily alleviate this problem.

A **public address** can be routed on the Internet. Thus, hosts that must be Internet-accessible must be configured with (or *reachable* by) public addresses. Allocation of public addresses is governed by the Internet Assigned Numbers Authority (IANA).

A **private address** is intended for internal use within a home or organization, and can be freely used by anyone. However, private addresses can *never be routed* on the Internet. In fact, Internet routers are configured to immediately drop traffic with private addresses.

Three private address ranges were defined in RFC 1918, one for each IPv4 class:

- Class A - **10.x.x.x /8**
- Class B - **172.16.x.x /12**
- Class C - **192.168.x.x /24**

It is possible to *translate* between private and public addresses, using **Network Address Translation (NAT).** NAT allows a host configured with a private address to be *stamped* with a public address, thus allowing that host to communicate across the Internet. It is also possible to translate multiple privately-addressed hosts to a single public address, which conserves the public address space.

NAT provides an additional benefit – hiding the specific addresses and addressing structure of the internal (or *private*) network.

Note: NAT is *not* restricted to private-to-public address translation, though that is the most common application. NAT can also perform public-to-public address translation, as well as private-to-private address translation.

NAT is only a temporarily solution to the address shortage problem. IPv4 will eventually be replaced with IPv6, which supports a vast address space.

Both NAT and IPv6 are covered extensively in other guides.

* * *

Reserved IPv4 Addresses

In addition to the three private IPv4 ranges, several other addresses and ranges are reserved for specific purposes:

- The **0.0.0.0 /0** network is used to identify **all networks,** and is referred to as the **default route.** If a default route exists in a routing table, it will be used only if there is *not* a more *specific* route to a particular destination. Routing and default routes are covered extensively in another guide.

- The **0.0.0.0 /8** range is used to identify hosts on the *local* network. Addresses in this range can only be used as a *source* address. The most commonly used address in this range is **0.0.0.0 /32,** which a host will use when dynamically attempting to learn its IP address via Dynamic Host Configuration Protocol (DHCP). DHCP is covered extensively in another guide.

- The entire **127.x.x.x /8** range is reserved for diagnostic purposes. The most commonly used address in this range is **127.0.0.1**, which identifies the local host, and is referred to as the **loopback** or **localhost** address.

- The **169.254.x.x /16** range is reserved for Automatic Private IP Addressing (APIPA). A host assigns itself an address in this range, if it cannot dynamically obtain an address from a DHCP server.

- The **224.x.x.x – 239.x.x.x** ranges are reserved for **multicast**, and are referred to as **Class D** addresses.

- The **240.x.x.x – 255.x.x.x** ranges are reserved for future and experimental use, and were formerly referred to as **Class E** addresses.

- The **255.255.255.255** address can be used as a broadcast address for the local network.

* * *

The IPv4 Header

The IPv4 header is comprised of **12 required fields** and **1 optional field.** The minimum length of the header is **160 bits (20 bytes).**

Field	*Length*	*Description*
Version	4 bits	*Version of IP (in this case, IPv4)*
Internet Header Length	4 bits	*Specifies the length of the IP header (minimum 160 bits)*
DSCP	8 bits	*Classifies traffic for QoS*
Total Length	16 bits	*Specifies the length of both the header and data payload*
Identification	16 bits	*Uniquely identifies fragments of a packet*
Flags	3 bits	*Flags for fragmentation*
Fragment Offset	13 bits	*Identifies the fragment relative to the start of the packet*
Time to Live	8 bits	*Decremented by each router traversed*
Protocol	8 bits	*Specifies the next upper layer protocol*
Header Checksum	16 bits	*Checksum for error checking*
Source Address	32 bits	*Source IPv4 address*
Destination Address	32 bits	*Destination IPv4 address*
Options	Variable	*Optional field for various parameters*

The 4-bit **Version field** is set to a value of *4* for IPv4.

The 4-bit **Internet Header Length field** identifies the length of the IPv4 header, measured in 32-bit *words*. The minimum of length of an IPv4 header is 160 bits, or 5 words (32 x 5 = 160).

The 8-bit **Differentiated Service Code Point (DSCP) field** is used to classify traffic for Quality of Service (QoS) purposes. QoS is covered in great detail in other guides. This field was previously referred to as the Type of Service (ToS) field.

The 16-bit **Total Length field** identifies the total packet size, measured in *bytes,* including both the IPv4 header and the data payload. The minimum size of an IPv4 packet is 20 bytes – essentially a header with no payload. The maximum packet size is **65,535 bytes**.

* * *

The IPv4 Header (continued)

Field	*Length*	*Description*
Version	4 bits	*Version of IP (in this case, IPv4)*
Internet Header Length	4 bits	*Specifies the length of the IP header (minimum 160 bits)*
DSCP	8 bits	*Classifies traffic for QoS*
Total Length	16 bits	*Specifies the length of both the header and data payload*
Identification	16 bits	*Uniquely identifies fragments of a packet*
Flags	3 bits	*Flags for fragmentation*
Fragment Offset	13 bits	*Identifies the fragment relative to the start of the packet*
Time to Live	8 bits	*Limits the lifetime of a packet*
Protocol	8 bits	*Specifies the next upper layer protocol*
Header Checksum	16 bits	*Checksum for error checking*
Source Address	32 bits	*Source IPv4 address*
Destination Address	32 bits	*Destination IPv4 address*
Options	Variable	*Optional field for various parameters*

An IPv4 packet that is larger than the **Maximum Transmission Unit (MTU)** size of a link must be **fragmented**. By default, the MTU for Ethernet is **1500 bytes.**

Three fields are used when a packet must be fragmented - the 16-bit **Identification field**, the 3-bit **Flags field**, and the 13-bit **Fragment Offset field**. Each fragment of the packet is marked with the same *Identification* number. The *Fragment Offset* allows the destination host to reassemble the fragments in the proper order.

The *Flags* field dictates two conditions:

- **Don't Fragment (DF)** – indicates the packet cannot be fragmented. If a packet exceeds a link's MTU size and this flag is set, then the packet is dropped. An ICMP error message will then be sent back to the source host.
- **More Fragments (MF)** – all fragments have this bit set to one, *except* for the last fragment, where the bit is set to zero. This allows the destination host to know when it has received all fragments.

* * *

The IPv4 Header (continued)

Field	*Length*	*Description*
Version	4 bits	*Version of IP (in this case, IPv4)*
Internet Header Length	4 bits	*Specifies the length of the IP header (minimum 160 bits)*
DSCP	8 bits	*Classifies traffic for QoS*
Total Length	16 bits	*Specifies the length of both the header and data payload*
Identification	16 bits	*Uniquely identifies fragments of a packet*
Flags	3 bits	*Flags for fragmentation*
Fragment Offset	13 bits	*Identifies the fragment relative to the start of the packet*
Time to Live	8 bits	*Limits the lifetime of a packet*
Protocol	8 bits	*Specifies the next upper layer protocol*
Header Checksum	16 bits	*Checksum for error checking*
Source Address	32 bits	*Source IPv4 address*
Destination Address	32 bits	*Destination IPv4 address*
Options	Variable	*Optional field for various parameters*

The 8-bit **Time to Live (TTL) field** limits the lifetime of the packet, preventing it from being endlessly forwarded. When a router forwards a packet, it will decrement the TTL value by one. Once the TTL value reaches zero, the packet is dropped.

The 8-bit **Protocol field** identifies the next upper-layer header, and is covered in the next section.

The 16-bit **Header Checksum field** is used to error-check the IPv4 header. The receiving host will discard the packet if it fails the checksum calculation.

The 32-bit **Source Address field** identifies the *sending* host. The 32-bit **Destination Address field** identifies the *receiving* host. The value of both of these fields can be changed as the packet is forwarded, using NAT.

The variable-length **Options field** provides additional optional IPv4 parameters, outside the scope of this guide.

* * *

IPv4 Protocol Numbers

The 8-bit **Protocol field** specifies the next upper-layer header within the data payload of the packet. These upper-layer protocols are identified using **IP Protocol Numbers**.

The following is a list of common IP Protocol Numbers, as assigned by the IANA:

Protocol Number	**Upper-Layer Protocol**
1	ICMP
2	IGMP
6	TCP
9	IGRP
17	UDP
46	RSVP
47	GRE
50	IPSEC ESP
51	IPSEC AH
88	EIGRP
89	OSPF

In IPv6, this field is referred to as the **Next Header field.**

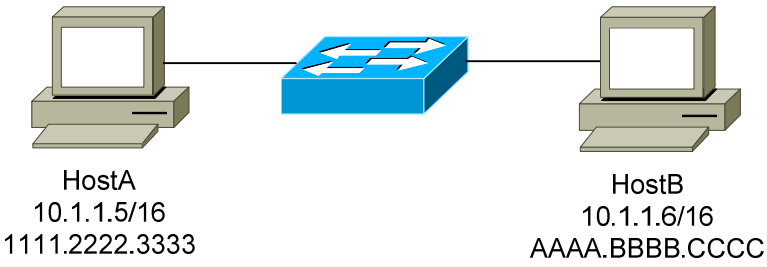
(Reference: http://www.iana.org/assignments/protocol-numbers)

* * *

Resolving Logical Addresses to Hardware Addresses

A host cannot directly send data to another host's logical address. A destination logical address must be *mapped* to a hardware address, so that the Data-Link layer can package a frame to transmit on the physical medium.

The **Address Resolution Protocol (ARP)** provides this mechanism for IPv4 on Ethernet networks. ARP allows a host to determine the MAC address for a particular destination IP address.



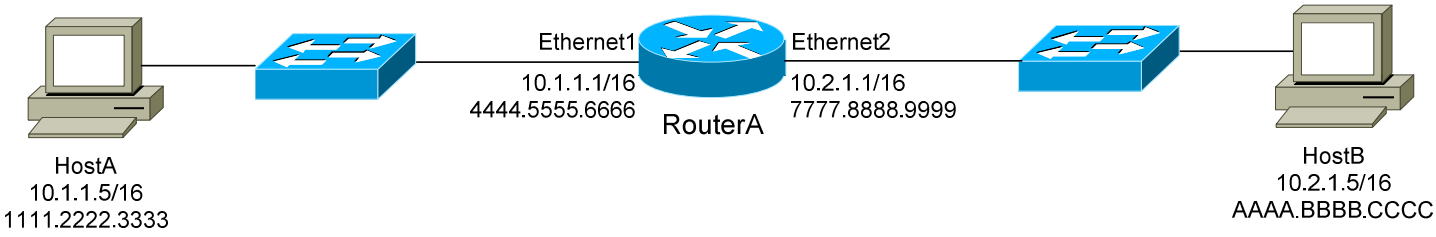
Consider the above diagram. The following demonstrates the steps required for HostA to communicate with HostB:

- First, HostA will determine if the destination IP address of 10.1.1.6 is *itself*. If that address is configured on a local interface, the packet never leaves HostA. In this example, 10.1.1.6 is *not* locally configured on HostA.
- Next, HostA will determine if the 10.1.1.6 address is on the *same network* or *subnet* as itself. HostA consults its **local routing table** to make this determination. In this example, the subnet mask is */16*. Thus, HostA's IP address of 10.1.1.5 and the destination address of 10.1.1.6 are on the same network (*10.1*).
- Because HostA and HostB are on the same network, HostA will then broadcast an **ARP request**, asking for the MAC address of the 10.1.1.6 address.
- HostB responds to the ARP request with an **ARP reply,** containing its MAC address (AAAA.BBBB.CCCC).
- HostA can now construct a Layer-2 frame, with a destination of HostB's MAC address. HostA forwards this frame to the switch, which then forwards the frame to HostB.

* * *

Resolving Logical Addresses to Hardware Addresses (continued)

Now consider a slightly modified scenario between HostA and HostB:

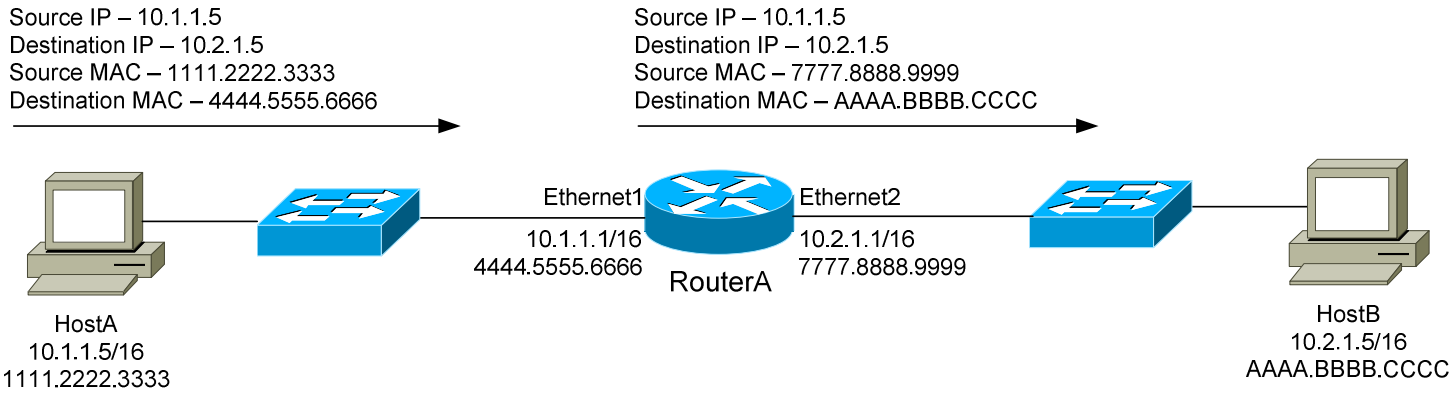


- Again, HostA will determine if the destination IP address of 10.2.1.5 is *itself*. In this example, 10.2.1.5 is *not* locally configured on HostA.
- Next, HostA will determine if the 10.2.1.5 address is on the *same network* or *subnet* as itself. In this example, the subnet mask is */16.* Thus, HostA's IP address of 10.1.1.5 and the destination address of 10.2.1.5 are ***not*** on the same network.
- Because HostA and HostB are *not* on the same network, HostA will parse its local routing table for a route to this destination network of 10.2.x.x/16. Hosts are commonly configured with a **default gateway** to reach all other destination networks.
- HostA determines that the 10.1.1.1 address on RouterA is its default gateway. HostA will then broadcast an ARP request, asking for the MAC address of the 10.1.1.1 address.
- RouterA responds to the ARP request with an ARP reply containing its MAC address (4444.5555.6666). HostA can now construct a Layer-2 frame, with a destination of RouterA's MAC address.
- Once RouterA receives the frame, it will parse its own routing table for a route to the destination network of 10.2.x.x/16. It determines that this network is directly attached off of its *Ethernet2* interface. RouterA then broadcasts an ARP request for the 10.2.1.5 address.
- HostB responds to the ARP request with an ARP reply containing its MAC address (AAAA.BBBB.CCCC). RouterA can now construct a Layer-2 frame, with a destination of HostB's MAC address.

* * *

Resolving Logical Addresses to Hardware Addresses (continued)

Consider the following example again:



Note that as a packet is *routed,* the source and destination IP address remain unchanged. However, both the source and destination MAC address *did* change.

This is because a MAC address contains no network hierarchy, and thus is only significant on the *local* network. In the above scenario, HostA and HostB could not communicate directly using Layer-2 addressing. At every routed *hop,* the source and destination MAC address are adjusted to reflect the source and destination hosts on the *local* network.

The source and destination IP address will *only* be changed if NAT is used.

The ARP Table

A host can build an **ARP table** that contains a list of IP to MAC address translations. The ARP table is only locally significant to that host. There are two methods to populate an ARP table:

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

A **static ARP entry** is created manually on a host, and will remain permanently until purposely removed. More commonly, ARP tables are built **dynamically** by caching ARP replies. Cached entries will eventually be aged out of the ARP table. The aging time will vary depending on the operating system, and can range from several seconds to several hours.

* * *

Troubleshooting IP using ICMP

The **Internet Control Message Protocol (ICMP)** is used for a multitude of informational and error messaging purposes.

The following is a list of common ICMP types and codes:

Type	*Code*	*Description*
0	**0**	**Echo Reply**
3	**-**	**Destination Unreachable**
	0	*Network Unreachable*
	1	*Host Unreachable*
	2	*Protocol Unreachable*
	3	*Port Unreachable*
	4	*Fragmentation Needed – Don't Fragment Flag Set*
	6	*Destination Network Unknown*
	7	*Destination Host Unknown*
	9	*Destination Network Administratively Prohibited*
	10	*Destination Host Administratively Prohibited*
5		**Redirect**
8		**Echo**
11		**TTL Exceeded**

The two most common troubleshooting tools that utilize ICMP are:

- **Packet Internet Groper (ping)**
- **Traceroute**

Ping is a core connectivity troubleshooting tool, which utilizes the Echo Request and Echo Reply ICMP messages to determine if an IP address is reachable and responding. Ping will additionally provide the **round-trip time** between the source and destination, usually measured in milliseconds.

Traceroute determines the routing path a packet takes to reach its destination. Traceroute will not only identify each router the packet has been forwarded through, but will also measure the delay experienced at each router *hop*.

* * *

Section 7
- TCP and UDP -

Transport Layer Protocols

The **Transport layer (OSI Layer-4)** does *not* actually transport data, despite its name. Instead, this layer is responsible for the *reliable* transfer of data, by ensuring that data arrives at its destination error-free and in order.

The Transport layer is referred to as the **Host-to-Host layer** in the Department of Defense (DoD) reference model.

Transport layer communication falls under two categories:

- **Connection-oriented** – requires that a connection with specific agreed-upon parameters be established before data is sent.
- **Connectionless** – requires no connection before data is sent.

Connection-oriented protocols provide several important services:

- **Connection establishment –** connections are established, maintained, and ultimately terminated between devices.
- **Segmentation and sequencing –** data is *segmented* into smaller pieces for transport. Each segment is assigned a *sequence number*, so that the receiving device can reassemble the data on arrival.
- **Acknowledgments –** receipt of data is confirmed through the use of *acknowledgments*. If a segment is lost, data can be retransmitted to guarantee delivery.
- **Flow control** (or **windowing) –** data transfer rate is negotiated to prevent congestion.

The TCP/IP protocol suite incorporates two Transport layer protocols:

- **Transmission Control Protocol (TCP) –** connection-oriented
- **User Datagram Protocol (UDP) -** connectionless

Both TCP and UDP provide a mechanism to differentiate applications running on the same host, through the use of **port numbers.** When a host receives a packet, the port number tells the transport layer which higher-layer application to hand the packet off to.

Both TCP and UDP will be covered in detail in this guide. Please note that the *best* resource on the Internet for TCP/UDP information is the exemplary TCP/IP Guide, found here: http://www.tcpipguide.com/free/index.htm

* * *

Port Numbers and Sockets

Both TCP and UDP provide a mechanism to differentiate applications (or **services**) running on the same host, through the use of **port numbers.** When a host receives a segment, the port number tells the transport layer which higher-layer application to hand the packet off to. This allows multiple network services to operate simultaneously on the same logical address, such as a web *and* an email server.

The range for port numbers is **0 – 65535**, for both TCP and UDP.

The combination of the IP address and port number (identifying both the *host* and *service)* is referred to as a **socket**, and is written out as follows:

192.168.60.125:443

Note the colon separating the IP address (*192.168.60.125*) from the port number (*443*).

The first 1024 ports (**0-1023**) have been reserved for widely-used services, and are recognized as **well-known** ports. Below is a table of several common TCP/UDP ports:

Port Number	Transport Protocol	Application
20, 21	TCP	FTP
22	TCP	SSH
23	TCP	Telnet
25	TCP	SMTP
53	UDP or TCP	DNS
80	TCP	HTTP
110	TCP	POP3
443	TCP	SSL
666	TCP	Doom

Ports ranging from **1024 – 49151** are referred to as **registered ports**, and are allocated by the IANA upon request. Ports ranging from **49152 – 65535** cannot be registered, and are considered **dynamic.** A client *initiating* a connection will randomly choose a port in this range as its *source* port (for some operating systems, the dynamic range starts at *1024* and higher).

For a complete list of assigned port numbers, refer to the IANA website:

http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml

* * *

Transmission Control Protocol (TCP)

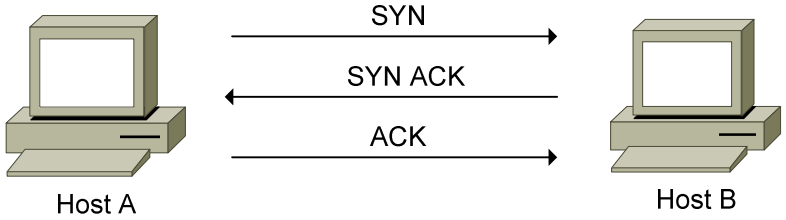
The **Transmission Control Protocol (TCP)** is a **connection-oriented** transport protocol, providing reliable delivery over an Internet Protocol (IP) network. Together, TCP and IP provide the core functionality for the **TCP/IP** or **Internet protocol suite**.

TCP was originally defined in RFC 675, and initially designed to perform *both* Network and Transport layer functions. When this proved to be an inflexible solution, those functions were separated - with IP providing Network layer services, and TCP providing Transport layer services. This separation was formalized in **version 4** of TCP, defined in RFC 793.

Because TCP is connection-oriented, parameters must be agreed upon by both the sending and receiving devices before a connection is established.

Establishing a TCP Connection

TCP employs a **three-way handshake** to form a connection. Control messages are passed between the two hosts as the connection is set up:



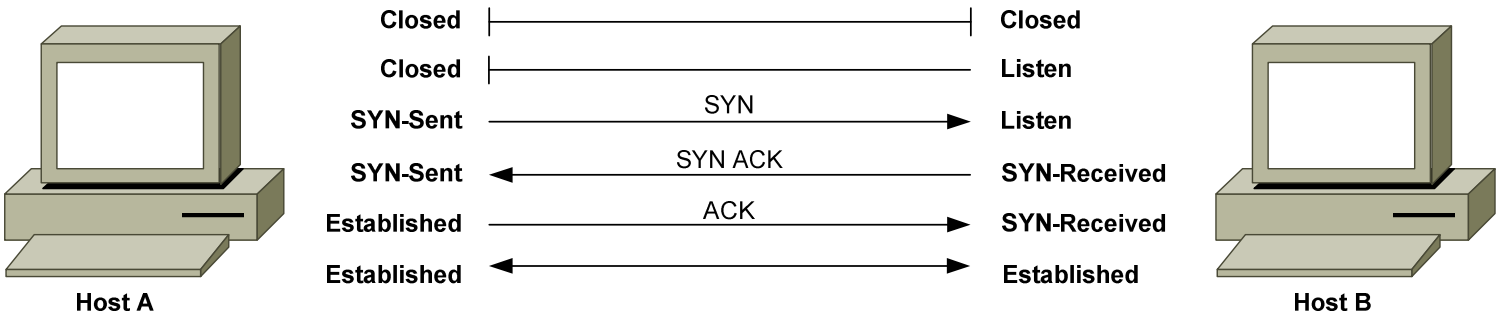
- HostA sends a **SYN** (short for **synchronize)** message to HostB to initiate a connection.
- HostB responds with an **ACK** (short for **acknowledgement)** to HostA's SYN message, and sends its own **SYN** message. The two messages are combined to form a single **SYN+ACK** message.
- HostA completes the three-way handshake by sending an **ACK** to HostB's SYN.

The TCP header contains six different **flags,** including a SYN flag and an ACK flag. Thus, when a particular *type* of message needs to be sent, the appropriate flag is marked as **on,** or changed from a *0* to a *1*. A SYN+ACK message has both flags set to on (*1*).

* * *

Establishing a TCP Connection (continued)

As the three-way handshake occurs, the sending and receiving hosts will pass through several **states:**



A **closed** state indicates a complete absence of a TCP connection.

Before a host can accept a request for a TCP connection, the host must enter a **listen** state, also known as a ***passive* open**. For example, a web server will passively *listen* on the HTTP port, waiting for incoming connection requests. A host must listen on each port it wishes to accept connections on.

A host will enter a **SYN-sent** state once it sends a **SYN message** to initiate a connection, also known as an ***active* open**. The sending host will remain in this state as it waits for the remote host's **ACK message.**

The receiving host will respond to the SYN message with a **SYN+ACK message,** and enter a **SYN-received** state**.**

The sending host will respond to the SYN+ACK message with its own **ACK message** and enter an **Established** state. The receiving host will enter an Established state once it receives this final ACK.

An Established state indicates that data transfer can occur. The communication becomes **bidirectional**, regardless of which host initiated the connection.

TCP can support many simultaneous connections, and must track and maintain each connection individually. Connections are identified by the *sockets* of both the source and destination host, and data specific to each connection is maintained in a **Transmission Control Block (TCB).**

(Reference: http://www.tcpipguide.com/free/t_TCPConnectionEstablishmentProcessTheThreeWayHandsh-3.htm)

* * *

TCP Segmentation and Sequencing

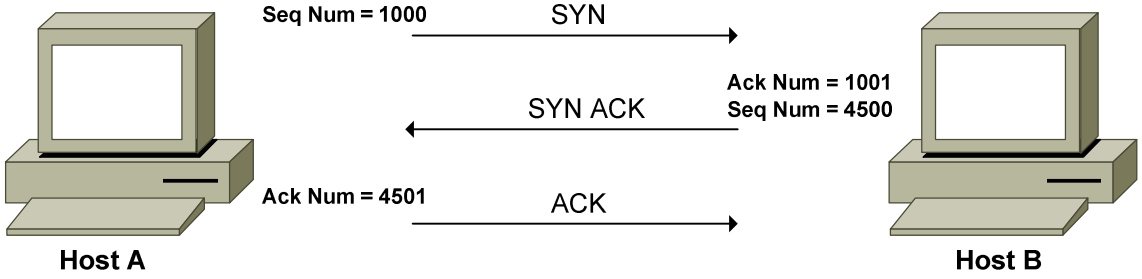
TCP is a **stream-oriented** transport protocol. This allows the application layer to send a continuous stream of unstructured data and rely on TCP to package the data as **segments**, regardless of the amount of data.

TCP will not only segment data into smaller pieces for transport, but will also assign a **sequence number** to each segment. Note though that this sequence number identifies the data (bytes) *within* the segment rather than the segment *itself.*

Sequencing serves two critical purposes:

- It allows the receiving host to reassemble the data from multiple segments in the correct order, upon arrival.
- It allows receipt of data within a segment to be *acknowledged,* thus providing a mechanism for dropped segments to be detected and resent.

When establishing a connection, a host will choose a 32-bit **initial sequence number (ISN).** The ISN is chosen from a randomizing timer, to prevent accidental overlap or predictability.



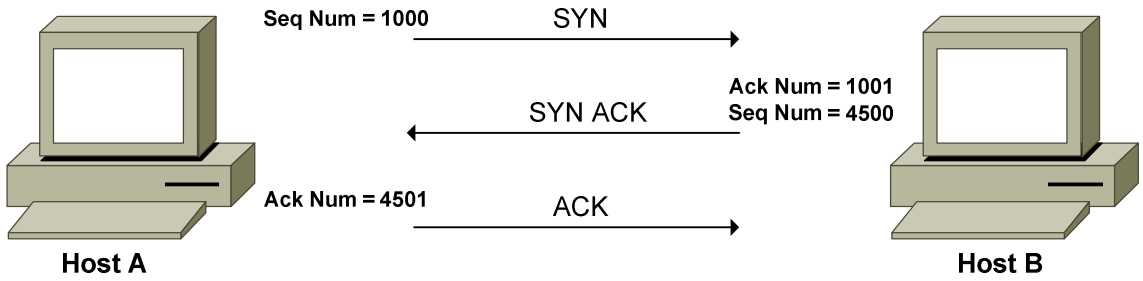
The receiving host responds to this sequence number with an **acknowledgment number**, set to the **sequence number + 1**. In the above example, HostB's acknowledgment number would thus be *1001*.

HostB includes an initial sequence number with *its* SYN message as well – *4500* in the above example. HostA would respond to this sequence number with an acknowledgement number of *4501*.

The TCP header contains both a 32-bit Sequence Number and 32-bit Acknowledgement Number field.

* * *

TCP Sliding Window



Once the TCP connection is established, the sequence numbers are used to identify the *data* within the segment. Using the above example again, HostA's first byte of data will be assigned a sequence number *1001*. Note that this is HostB's acknowledgment number, which essentially identifies which byte the receiving host is expecting *next*. HostB's first byte of data will be assigned a sequence number of *4501*.

Note that each *individual* byte of data **is not** assigned a sequence number and acknowledged independently, as this would introduce massive overhead. Instead, data is sequenced and acknowledged in *groups*, dictated by the **TCP window size.** The window size can never exceed the **maximum segment size (MSS)**, which is **536 bytes** by default.

The TCP window size is dictating by the *receiving host*, and informs the sender how many bytes it is permitted to send, before waiting for an acknowledgement. This window size can be dynamically changed to provide a measure of **flow control,** preventing buffer congestion on the receiving host.

A window size of **0** would instruct the sender to send no further data, usually indicating significant congestion on the receiving host.

TCP employs a ***sliding* window mechanism**. Bytes in a sliding window fall into one of four categories:

- Bytes that have *already* been sent *and* acknowledged.
- Bytes that have been sent, but *not acknowledged.*
- Bytes that have *not yet been sent*, but the receiving host *is ready for*.
- Bytes that have *not yet been sent*, and the receiving host *is not ready for.*

(Reference: http://www.tcpipguide.com/free/t_TCPSlidingWindowAcknowledgmentSystemForDataTranspo-5.htm; http://docwiki.cisco.com/wiki/Internet_Protocols#Transmission_Control_Protocol_.28TCP.29)

* * *

TCP Sliding Window (continued)

Consider the following conceptual example:

	Byte #	***Category***
TCP Window	1-50	Bytes sent and acknowledged
	51-75	Bytes sent and not yet acknowledged
	76-100	Bytes not sent, receiving host is ready
	101-200	Bytes not sent, receiving host is *not ready*

Several determinations can be made:

- The TCP stream is 200 bytes total.
- The TCP window size is 50 bytes total.
- The sending host can immediately send another 25 bytes of data (bytes 76-100)

Once bytes 51-75 are acknowledged, and bytes 76-100 are sent, the window will *slide* down*:*

	Byte #	***Category***
TCP Window	1-75	Bytes sent and acknowledged
	76-100	Bytes sent and not yet acknowledged
	101-125	Bytes not sent, receiving host is ready
	126-200	Bytes not sent, receiving host is *not ready*

This assumes that that TCP window stays at 50 bytes. Remember that the window size is dictated by the receiving host (in a 16-bit **Window field** in the TCP header), and can be dynamically adjusted.

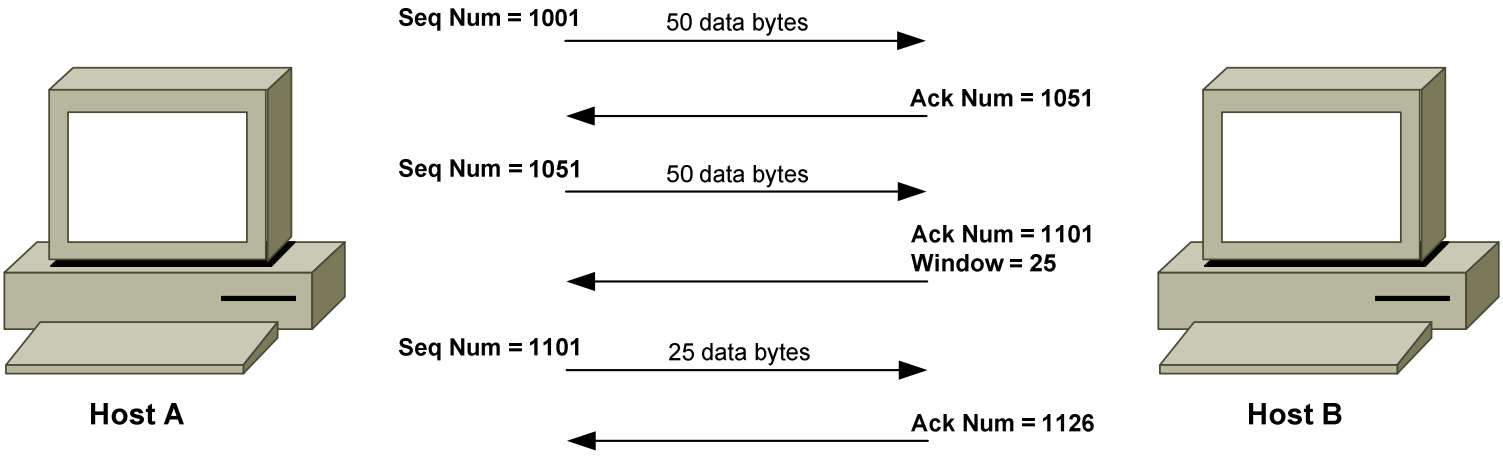
For efficiency, TCP will generally wait to send a segment until the agreed-upon TCP window size is full. This may not be acceptable for certain types of applications, which may not tolerate this latency.

The TCP header provides a **PSH (Push) flag** to accommodate this, allowing data to be sent *immediately*, regardless if the TCP window has been filled. The PSH flag can be used in conjunction with the **URG (Urgent) flag,** which allows specified data to be prioritized over other data. The URG flag must be used with the **Urgent Pointer field**, which identifies the *last* byte of urgent data, to identify where non-urgent data begins in a segment.

* * *

TCP Sliding Window (continued)

How do sequence and acknowledgement numbers fit within the sliding window concept? Consider the following *very* basic example, and assume the TCP connection is already established:



Recall that during the setup of a TCP connection, the acknowledgement number was set to the sequence number + 1. However, during data transfer, the acknowledgement number is used to acknowledge receipt of a *group* of data bytes.

In the above example, the initial TCP window size is set to *50* bytes, and the first byte in the stream is assigned a sequence number of *1001*. HostB acknowledges receipt of these 50 data bytes with an acknowledgement number of *1051* (for the mathematically disinclined, this is 1001 + 50). ☺

Once acknowledged, HostA then sends another 50 bytes of data, identifying the first byte with a sequence number of 1051. HostB acknowledges receipt again, with an ACK number of 1101. However, HostB also adjusts the TCP window size to 25 bytes, perhaps due to congestion.

HostA's next segment will thus only contain 25 bytes of data, with a sequence number of 1101. HostB acknowledges these 25 bytes with an ACK number of 1126.

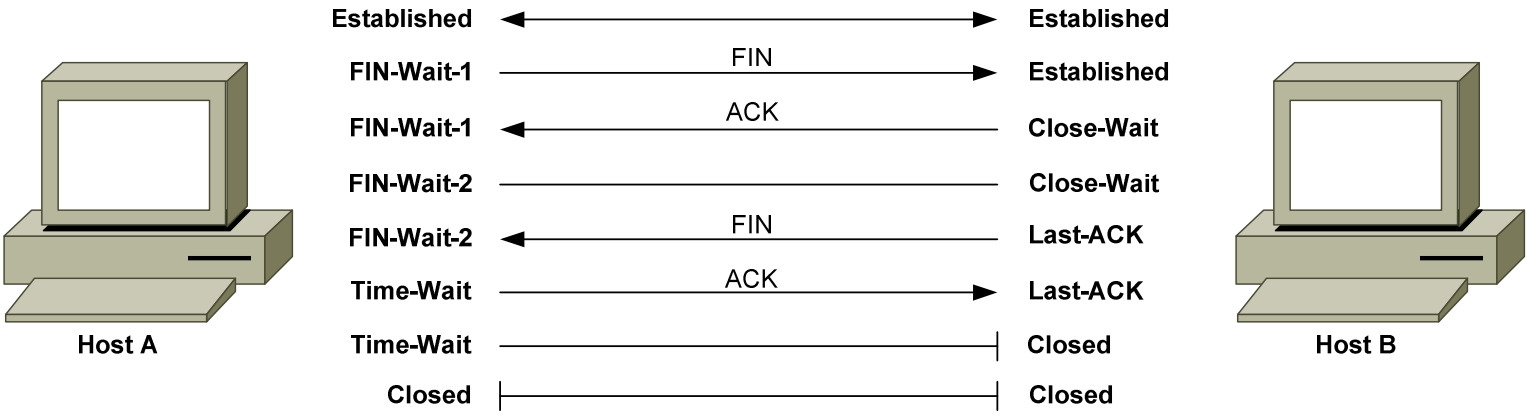
Every time a segment is sent, the sending host starts a **retransmission timer,** dynamically determined (and adjusted) based on the round-trip time between the two hosts. If an acknowledgement is not received before the retransmission timer expires, the segment is resent. This allows TCP to guarantee delivery, even when segments are lost.

* * *

Gracefully Terminating a TCP Connection

A TCP connection will remain established until it is purposely **terminated** by either host. The most common reason for connection termination is that both hosts have finished sending data. The termination process is handled separately by each host, allowing both hosts to fully complete data transfer before the connection is terminated.

Hosts can terminate an established TCP connection by sending a message with the **FIN (Finish) flag** set:



Once HostA sends the FIN message, it will enter a **FIN-Wait-1** state, waiting for the FIN to be acknowledged.

HostB responds to the FIN with an ACK message, and enters a **Close-Wait** state, allowing the local application to finish its processes. HostA receives the ACK and enters a **FIN-Wait-2** state, waiting for HostB to send a FIN message of its own, indicating it is safe to close the connection.

HostB sends a FIN message to HostA once the application process is complete, and enters a **Last-ACK** state.

HostA receives the FIN message and responds with an ACK message. HostA then enters a **Time-Wait** state, allowing time for the ACK to be received by HostB.

HostB receives the ACK message and enters a **Closed** state.

HostA's Time-Wait timer expires, and it also enters a Closed state. The connection is now gracefully terminated.

(Reference: http://www.tcpipguide.com/free/t_TCPConnectionTermination-2.htm)

* * *

Less than Graceful TCP Connection Termination

A TCP connection can become **half-open,** indicating that one host is an established state while the other is not. Half-open connections can result from **interruption** by an intermediary device (such as a firewall), or from a software or hardware issue.

TCP utilizes the **Reset message,** using the **RST flag**, to address half-open connections. Sending a RST message will force the remote host to reset the TCP connection and return to a *closed* state, or return to a passive *listen* state if the remote host is a server listening on that port.

There are a few scenarios in which a RST might be sent:

- A host receives a TCP segment from a host that it does not have a connection with.
- A host receives a segment with an incorrect sequence or acknowledgement number.
- A host receives a SYN request on a port it is not listening on.

Note on half-open connections: A **SYN flood** is a common denial-of-service attack that sends a large number of TCP SYN messages to a host, while spoofing the source address. The host will respond with an equal number of SYN+ACK messages, and will wait for the final ACK message that never comes. This spawns a large amount of half-open connections, which can prevent the host from responding to legitimate requests.

Modern firewalls can detect SYN flood attacks and minimize the number of accepted half-open connections.

(Reference: http://www.tcpipguide.com/free/t_TCPConnectionManagementandProblemHandlingtheConnec.htm)

* * *

The TCP Header

The TCP header is comprised of **12 fields,** and has a minimum size of **160 bits (20 bytes)**:

Field	***Length***	***Description***
Source Port	16 bits	*Source TCP Port*
Destination Port	16 bits	*Destination TCP Port*
Sequence Number	32 bits	*Sequence Number*
Ack Number	32 bits	*Acknowledgement Number*
Data Offset	4 bits	*Indicates where the data begins in a TCP segment*
Reserved	6 bits	*Always set to 0*
Control Bits	6 bits	*URG, ACK, PSH, RST, SYN, and FIN flags*
Window	16 bits	*Used for Flow Control*
Checksum	16 bits	*Used for Error-Checking*
Urgent Pointer	16 bits	*Identifies last byte of Urgent traffic*
Options	Variable	
Padding	Variable	*To ensure the TCP header ends at a 32-bit boundary*

The 16-bit **Source Port field** identifies the application service on the *sending* host. The 16-bit **Destination Port field** identifies the application service on the *remote* host.

The 32-bit **Sequence Number field** is used both during connection establishment, and during data transfer. During connection establishment (SYN message), an *initial sequence number* is randomly chosen. Subsequently, sequence numbers are used to identify data bytes in a stream.

The 32-bit **Acknowledgement Number field**, as its name suggests, is used to acknowledge a sequence number. During connection setup, this is set to the sending host's initial sequence number + 1. During data transfer, this value is used to acknowledge receipt of a group of data bytes.

The 4-bit **Data Offset field** indicates where data begins in a TCP segment, by identifying the number of 32-bit multiples in the TCP header. A TCP header *must* end on a 32-bit boundary.

Following the data offset field is the 6-bit **Reserved** (for future use) **field,** which is always set to zeroes.

* * *

The TCP Header (continued)

Field	*Length*	*Description*
Source Port	16 bits	*Source TCP Port*
Destination Port	16 bits	*Destination TCP Port*
Sequence Number	32 bits	*Sequence Number*
Ack Number	32 bits	*Acknowledgement Number*
Data Offset	4 bits	*Indicates where the data begins in a TCP segment*
Reserved	6 bits	*Always set to 0*
Control Bits	6 bits	*URG, ACK, PSH, RST, SYN, and FIN flags*
Window	16 bits	*Used for Flow Control*
Checksum	16 bits	*Used for Error-Checking*
Urgent Pointer	16 bits	*Identifies last byte of Urgent traffic*
Options	Variable	
Padding	Variable	*To ensure the TCP header ends at a 32-bit boundary*

The 6-bit **Control Bits** field contains six 1-bit flags, in the following order:

- **URG (Urgent) –** prioritizes specified traffic.
- **ACK (Acknowledgment) –** acknowledges a SYN or receipt of data.
- **PSH (Push) –** forces an immediate send even if window is not full.
- **RST (Reset) –** forcefully terminates an improper connection.
- **SYN (Synchronize) –** initiates a connection.
- **FIN (Finish) –** gracefully terminates a connection when there is further data to send.

The 16-bit **Window field** identifies the number of data octets that the receiver is able to accept.

The 16-bit **Checksum field** is used for error-checking, and is computed using both the TCP segment and select fields from the IP header. The receiving host will discard the segment if it fails the checksum calculation.

The 16-bit **Urgent Pointer field** is used to identify the *last* byte of prioritized traffic in a segment, when the URG flag is set.

The variable-length **Options field** provides additional optional TCP parameters, outside the scope of this guide.

The variable-length **Padding field** ensures the TCP header ends on a 32-bit boundary, and is always set to zeroes.

* * *

User Datagram Protocol (UDP)

The **User Datagram Protocol (UDP)** is a **connectionless** transport protocol, and is defined in RFC 768.

UDP, above all, is *simple.* It provides no three-way handshake, no flow-control, no sequencing, and no acknowledgment of data receipt. UDP essentially forwards the segment and takes no further interest.

Thus, UDP is **inherently unreliable,** especially compared to a connection-oriented protocol like TCP. However, UDP **experiences less latency** than TCP, due to the reduced overhead. This makes UDP ideal for applications that require speed over reliability. For example, DNS primarily uses UDP as its transport protocol, though it supports TCP as well.

Like TCP, UDP does provide basic error-checking using a checksum, and uses port numbers to differentiate applications running on the same host.

The UDP header has only 4 **fields**:

Field	***Length***	***Description***
Source Port	16 bits	*Source UDP Port*
Destination Port	16 bits	*Destination UDP Port*
Length	16 bits	*Length of the header and the data*
Checksum	16 bits	*Used for Error-Checking*

The following provides a quick comparison of TCP and UDP:

TCP	***UDP***
Connection-oriented	Connectionless
Guarantees delivery	Does *not* guarantee delivery
Sends acknowledgments	Does *not* send acknowledgments
Reliable, but slower than UDP	Unreliable, but faster than TCP
Segments and sequences data	Does *not* provide sequencing
Resends dropped segments	Does *not* resend dropped segments
Provides flow control	Does *not* provide flow control
Performs CRC on data	Also performs CRC on data
Uses port numbers	Also uses port numbers

* * *

Section 8
- IPv6 Addressing -

IPv6 Basics

The most widespread implementation of IP currently is IPv4, which utilizes a 32-bit address. Mathematically, a 32-bit address can provide roughly 4 billion unique IP addresses (2^{32} = 4,294,967,296). Practically, the number of usable IPv4 addresses is much lower, as many addresses are reserved for diagnostic, experimental, or multicast purposes.

The explosive growth of the Internet and corporate networks quickly led to an IPv4 address shortage. Various solutions were developed to alleviate this shortage, including CIDR, NAT, and Private Addressing. However, these solutions could only serve as temporary fixes.

In response to the address shortage, **IPv6** was developed. IPv6 increases the address size to 128 bits, providing a nearly unlimited supply of addresses (340,282,366,920,938,463,463,374,607,431,768,211,456 to be exact). This provides roughly 50 *octillion* addresses *per person alive* on Earth today, or roughly 3.7 x 10^{21} addresses per square inch of the Earth's surface.

(References: http://cc.uoregon.edu/cnews/spring2001/whatsipv6.html; http://en.wikipedia.org/wiki/IPv6)

IPv6 offers the following features:

- **Increased Address Space and Scalability –** *providing the absurd number of possible addresses stated previously.*
- **Simplified Configuration –** *allows hosts to auto-configure their IPv6 addresses, based on network prefixes advertised by routers.*
- **Integrated Security –** *provides built-in authentication and encryption into the IPv6 network header*
- **Compatibility with IPv4 –** *simplifies address migration, as IPv6 is backward-compatible with IPv4*

The IPv6 Address

The IPv6 address is **128 bits**, as opposed to the 32-bit IPv4 address. Also unlike IPv4, the IPv6 address is represented in hexadecimal notation, separate by colons.

An example of an IPv6 address would be:

1254:1532:26B1:CC14:0123:1111:2222:3333

Each "grouping" (from here on called **fields**) of hexadecimal digits is 16 bits, with a total of eight fields. The hexadecimal values of an IPv6 address are **not case-sensitive.**

We can drop any leading zeros in each field of an IPv6 address. For example, consider the following address:

1423:0021:0C13:CC1E:3142:0001:2222:3333

We can condense that address to: 1423:21:C13:CC1E:3142:1:2222:3333

Only leading zeros can be condensed. If we have an entire field comprised of zeros, we can further compact the following address:

F12F:0000:0000:CC1E:2412:1111:2222:3333

The condensed address would be: F12F::CC1E:2412:1111:2222:3333

Notice the double colons (::). We can only condense one set of contiguous zero fields. Thus, if we had the following address:

F12F:0000:0000:CC1E:2412:0000:0000:3333

We could not condense that to: F12F::CC1E:2412::3333

The address would now be ambiguous, as we wouldn't know how many "0" fields were compacted in each spot. Remember that we can only use one set of double colons in an IPv6 address!

The IPv6 Prefix

IPv4 utilizes a *subnet mask* to define the network "prefix" and "host" portions of an address. This subnet mask can also be represented in Classless Inter-Domain Routing (CIDR) format.

IPv6 always use **CIDR** notation to determine what bits **notate the prefix** of an address:

Full Address:	1254:1532:26B1:CC14:123:1111:2222:3333/64
Prefix ID:	1254:1532:26B1:CC14:
Host ID:	123:1111:2222:3333

The */64* indicates that the first 64 bits of this address identify the prefix.

The IPv6 Interface ID and EUI-64 Format

The host portion of an IPv4 address is not based on the hardware address of an interface. IPv4 relies on **Address Resolution Protocol (ARP)** to map between the logical IP address and the **48-bit** hardware **MAC address**.

IPv6 unicasts generally allocate the first 64 bits of the address to identify the network (**prefix**), and the last 64 bits to identify the host (referred to as the **interface ID**). The interface ID *is* based on the interface's hardware address.

This interface ID adheres to the IEEE **64-bit Extended Unique Identifier** (**EUI-64**) format. Since most interfaces still use the 48-bit MAC address, the MAC must be converted into the EUI-64 format.

Consider the following MAC address: 1111.2222.3333. The first 24 bits, the Organizationally Unique Identifier (OUI), identify the manufacturer. The last 24 bits uniquely identify the host. To convert this to EUI-64 format:

1. The **first 24 bits** of the MAC (the **OUI**), become the first 24 bits of the EUI-64 formatted interface ID.
2. The ***seventh*** bit of the OUI is changed from a "**0**" to a "**1**".
3. The next 16 bits of the interface ID are **FFFE.**
4. The **last 24 bits** of the MAC (the **host ID**), become the last 24 bits of the interface ID.

Thus, the MAC address 1111.2222.3333 in EUI-64 format would become **1311:22FF:FE22:3333**, which becomes the interface ID.

* * *

The IPv6 Address Hierarchy

IPv4 separated its address space into specific **classes.** The class of an IPv4 address was identified by the high-order bits of the first octet:

- **Class A** - (00000001 – 01111111, or 1 - 127)
- **Class B -** (10000000 – 10111111, or 128 - 191)
- **Class C -** (11000000 – 11011111, or 192 - 223)
- **Class D -** (11100000 – 11101111, or 224 - 239)

IPv6's addressing structure is far more scalable. Less than 20% of the IPv6 address space has been designated for use, currently. The potential for growth is enormous.

The address space that *has* been allocated is organized into several types, determined by the high-order bits of the first field:

- **Special Addresses –** addresses begin **00*xx*:**
- **Link Local –** addresses begin **FE8*x*:**
- **Site Local –** addresses begin **FEC*x*:**
- **Aggregate Global –** addresses begin **2*xxx*:** or **3*xxx*:**
- **Multicasts –** addresses begin **FF*xx*:**
- **Anycasts**

(Note: an "x" indicates the value can be any hexadecimal number)

There are **no broadcast addresses** in IPv6. Thus, any IPv6 address that is not a *multicast* is a *unicast* address.

Anycast addresses identify a group of interfaces on multiple hosts. Thus, multiple hosts are configured with an *identical* address. Packets sent to an anycast address are sent to the *nearest* (i.e., least amount of hops) host. Anycasts are indistinguishable from any other IPv6 unicast address.

Practical applications of anycast addressing are a bit murky. One possible application would be a server farm providing an identical service or function, in which case anycast addressing would allow clients to connect to the nearest server.

Special (Reserved) IPv6 Addresses

The first field of a **reserved** or **special** IPv6 address will always begin **00*xx*.** Reserved addresses represent 1/256th of the available IPv6 address space.

Various reserved addresses exist, including:

- **0:0:0:0:0:0:0:0** (or **::**) – is an **unspecified** or **unknown** address. It is the equivalent of the IPv4 0.0.0.0 address, which indicates the absence of a configured or assigned address. In routing tables, the unspecified address is used to identify **all** or **any** possible hosts or networks.
- **0:0:0:0:0:0:0:1** (or **::1**) – is the **loopback** or **localhost** address. It is the equivalent of the IPv4 127.0.0.1 address.

Reserved Addresses - IPv4 and IPv6 Compatibility

To alleviate the difficulties of immediately migrating from IPv4 to IPv6, specific reserved addresses can be used to *embed* an IPv4 address into an IPv6 address.

Two types of addresses can be used for IPv4 embedding, **IPv4-compatible IPv6 addresses**, and **IPv4-mapped IPv6 addresses**.

- **0:0:0:0:0:0:a.b.c.d** (or **::a.b.c.d)** – is an **IPv4-*compatible* IPv6 address**. This address is used on devices that support both IPv4 *and* IPv6. A prefix of /96 is used for IPv4-compatible IPv6 addresses:

  ::192.168.1.1/96

- **0:0:0:0:0:FFFF:a.b.c.d** (or **::FFFF:a.b.c.d) –** is an **IPv4-*mapped* IPv6 address.** This address is used by IPv6 routers and devices to identify *non*-IPv6 capable devices. Again, a prefix of /96 is used for IPv4-mapped IPv6 addresses:

  ::FFFF:192.168.1.1/96

* * *

Link-Local IPv6 Addresses

Link-local IPv6 addresses are used only on a single link (subnet). Any packet that contains a link-local source or destination address is *never routed* to another link. Every IPv6-enabled interface on a host (or router) is assigned a link-local address. This address can be manually assigned, or auto-configured.

The first field of a **link-local** IPv6 address will always begin ***FE8x (1111 1110 10).*** Link-local addresses are **unicasts,** and represent 1/1024th of the available IPv6 address space. A prefix of **/10** is used for link-local addresses.

FE80::1311:22FF:FE22:3333/10

There is no hierarchy to a link-local address:

- The first 10 bits are fixed (**FE8**), known as the **Format Prefix (FP).**
- The next 54 bits are set to **0.**
- The final 64 bits are used as the **interface ID**.

Site Local IPv6 Addresses

Site-local IPv6 addresses are the equivalent of "private" IPv4 addresses. Site-local addresses can be routed within a *site* or *organization*, but cannot be globally routed on the Internet. Multiple private subnets within a "site" are allowed.

The first field of a **site-local** IPv6 address will always begin ***FECx (1111 1110 11).*** Site-local addresses are **unicasts,** and represent 1/1024th of the available IPv6 address space.

FEC0::2731:E2FF:FE96:C283/64

Site-local addresses do adhere to a hierarchy:

- The first 10 bits are the fixed FP (**FEC**).
- The next 38 bits are set to **0.**
- The next 16 bits are used to identify the **private subnet ID**.
- The final 64 bits are used as the **interface ID**.

To identify two separate subnets (*1111* and *2222*):

FEC0::1111:2731:E2FF:FE96:C283/64
FEC0::2222:97A4:E2FF:FE1C:E2D1/64

* * *

Aggregate Global IPv6 Addresses

Aggregate Global IPv6 addresses are the equivalent of "public" IPv4 addresses. Aggregate global addresses can be routed publicly on the Internet. Any device or site that wishes to traverse the Internet must be uniquely identified with an aggregate global address.

Currently, the first field of an **aggregate global** IPv6 address will always begin **2*xxx* (001).** Aggregate global addresses are **unicasts,** and represent 1/8th of the available IPv6 address space.

2000::2731:E2FF:FE96:C283/64

Aggregate global addresses adhere to a very strict hierarchy:

- The first 3 bits are the fixed FP.
- The next 13 bits are the **top-level aggregation identifier (TLA ID).**
- The next 8 bits are **reserved** for future use.
- The next 24 bits are the **next-level aggregation identifier (NLA ID).**
- The next 16 bits are the **site-level aggregation identifier (SLA ID).**
- The final 64 bits are used as the **interface ID**.

By have multiple **levels**, a consistent, organized, and scalable hierarchy is maintained. High level registries are assigned ranges of TLA IDs. These can then be subdivided in the NLA ID field, and passed on to lower-tiered ISPs.

Such ISPs allocate these prefixes to their customers, which can further subdivide the prefix using the SLA ID field, to create whatever local hierarchy they wish. The 16-bit SLA field provides up to 65535 networks for an organization.

Note: Do not confuse the SLA ID field of a global address field, with a site-local address. Site-local addresses cannot be routed publicly, where as SLA ID's are just a subset of the publicly routable aggregate global address.

* * *

Multicast IPv6 Addresses

Multicast IPv6 addresses are the equivalent of IPv4 multicast addresses. Interfaces can belong to one or more multicast **groups**. Interfaces will accept a multicast packet only if they belong to that group. Multicasting provides a much more efficient mechanism than **broadcasting**, which requires that every host on a link accept and process each broadcast packet.

The first field of a **multicast** IPv6 address will always begin **FF*xx* (1111 1111).** The full multicast range is **FF00** through **FFFF**. **Multicasts** represent 1/256th of the available IPv6 address space.

FF01:0:0:0:0:0:0:1

Multicast addresses follow a specific format:

- The first 8 bits **identify the address** as a **multicast** (1111 1111)
- The next 4 bits are a **flag value**. If the flag is set to all zeroes (0000), the multicast address is considered *well-known*.
- The next 4 bits are a **scope value**:
  - 0000 (0) = Reserved
  - 0001 (1) = Node Local Scope
  - 0010 (2) = Link Local Scope
  - 0101 (5) = Site Local Scope
  - 1000 (8) = Organization Local Scope
  - 1110 (e) = Global Scope
  - 1111 (f) = Reserved
- The final 112 bits identify the actual **multicast group**.

IPv4 multicast addresses had no mechanism to support multiple "**scopes."** IPv6 scopes allow for a multicast hierarchy, a way to *contain* multicast traffic.

* * *

Common IPv6 Multicast Addresses

The following is a list of common, well-known IPv6 multicast addresses:

Node-Local Scope Multicast Addresses

- FF01::1 – All-nodes address
- FF01::2 – All-routers address

Link-Local Scope Multicast Addresses

- FF02::1 – All-nodes address
- FF02::2 – All-routers address
- FF02::5 – OSPFv3 (OSPF IPv6) All SPF Routers
- FF02::6 – OSPFv3 Designated Routers
- FF02::9 – RIPng Routers
- FF02::13 – PIM Routers

Site-Local Scope Multicast Addresses

- FF05::2 – All-routers address

All hosts must join the **all-nodes** multicast group, for both the node-local and link-local scopes. All routers must join the **all-routers** multicast group, for the node-local, link-local, and site-local scopes.

Every site-local and aggregate global address is assigned a **solicited-node multicast** address. This solicited-node address is created by appending the last 24 bits of the interface ID to the following prefix: FF02::1:FF/103.

Thus, if you have a site-local address of:

FEC0::1111:2731:E2FF:FE96:C283

The corresponding solicited-node multicast address would be:

FF02::1:FF96:C283

Solicited-node multicast addresses are most often used for neighbor discovery (covered in an upcoming section in this guide).

* * *

Required IPv6 Addresses

At a minimum, each IPv6 interface on a **host** must recognize the following IPv6 addresses:

- The loopback address
- A link-local address
- Any configured site-local or aggregate global addresses
- Any configured multicast groups
- The all-nodes multicast address (both node-local and link-local scopes)
- The solicited-node multicast address for any configured unicast addresses

In *addition* to the above addresses, each IPv6 interface on a **router** must recognize the following IPv6 addresses:

- The subnet-router anycast address
- Any configured multicast groups
- The all-routers multicast address (node-local, link-local, and site-local scopes)

IPv6 Addresses and URLs

IPv6 addresses can also be referenced in **URLs** (Uniform Resource Locator). URL's, however, use the colon to represent a specific TCP "port". This is not an issue with IPv4 addresses, which can easily be referenced using a URL:

http://192.168.1.1/index.html

Because IPv6 fields are separated by colons, the IPv6 address must be placed in brackets, to conform to the URL standard:

http://[FEC0::CC1E:2412:1111:2222:3333]/index.html

* * *

The IPv6 Header

The IPv6 header has **8 fields** and is **320 bits** long. It has been considerably streamlined compared to its IPv4 counterpart, which has **12 fields** and is **160 bits** long.

Field	*Length*	*Description*
Version	4 bits	*Version of IP (in this case, IPv6)*
Traffic Class	8 bits	*Classifies traffic for QoS*
Flow Label	20 bits	*Identifies a flow between a source and destination*
Payload Length	16 bits	*Length of data in packet*
Next Header	8 bits	*Specifies the next upper-layer or extension header*
Hop Limit	8 bits	*Decremented by each router traversed*
Source Address	128 bits	*Source IPv6 address*
Destination Address	128 bits	*Destination IPv6 address*

The *Next Header* field is of some importance. This field can identify either the next upper-layer header (for example, UDP, TCP or ICMP), or it can identify a special **Extension Header**, which placed in between the IPv6 and upper layer header.

Several such extension headers exist, and are usually processed in the following order:

- **Hop-by-Hop Options** – *specifies options that should be processed by every router in the path. Directly follows the IPv6 header.*
- **Destination Options –** *specifies options that should be processed by the destination device.*
- **Routing Header** – *specifies each router the packet must traverse to reach the destination (source routing)*
- **Fragment Header** – *used when a packet is larger than the MTU for the path*
- **Authentication Header –** *used to integrate IPSEC Authentication Header (AH) into the IPv6 packet*
- **ESP Header –** *used to integrate IPSEC Encapsulating Security Payload (ESP) into the IPv6 packet*

(Reference: http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t2/ipv6/ftipv6o.htm#1004285)

* * *

ICMPv6

ICMP Version 6 (**ICMPv6**) is a core component of IPv6. All devices employing IPv6 must also integrate ICMPv6.

ICMPv6 provides many services, including (but not limited to):

- Error Messages
- Informational messages (such as *echo* replies for IPv6 ping)
- MTU Path Discovery
- Neighbor Discovery

There are four key ICMPv6 error messages:

- **Destination Unreachable (**ICMP packet type 1**) –** *indicates that the packet cannot be forwarded to its destination. The node sending this message includes an explanatory code:*
  - **0 -** No route to destination
  - **1** - Access is administratively prohibited
  - **3** - Address unreachable
  - **4** - Port unreachable

- **Packet Too Big (**ICMP packet type 2**)** – *indicates the packet is larger than the MTU of the link. IPv6* ***routers do not fragment*** *packets. Instead, the Packet Too Big message is sent to the source (sending) device, which then reduces (or fragments) the size of the packet to the reported MTU. This message is used for* ***Path MTU Discovery (PMTUD)****.*

- **Time Exceeded (**ICMP packet type 3**)** – *indicates that the hop count limit has been reached, usually indicating a routing loop*

- **Parameter Problem (**ICMP packet type 4**) –** *indicates an error in the IPv6 header, or an IPv6 extension header. The node sending this message includes an explanatory code:*
  - **0 -** Erroneous header field
  - **1 -** Unrecognized next-header type
  - **2 -** Unrecognized IPv6 option

*(*Reference: *http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080113b1c.shtml)*

* * *

Neighbor Discovery Protocol (NDP) and ICMPv6

The **neighbor discovery protocol (NDP)** provides a multitude of services for IPv6 enabled devices, including:

- Automatic address configuration, and prefix discovery
- Duplicate address detection
- MTU discovery
- Router discovery
- Address resolution

NDP replaces many IPv4 specific protocols, such as DHCP and ARP. NDP utilizes **ICMPv6** to provide the above services.

Periodically, IPv6 routers send out **Router Advertisements (RA's)** to both announce their presence on a link, and to provide auto-configuration information for hosts. This **RA** (ICMP packet type 134) is sourced from the link-local address of the sending router, and sent to the link-scope all-nodes multicast group. The sending router sets a **hop limit** of **255** on a RA; however, the RA packet *must not* be forwarded outside the local link.

Hosts use RA's to configure themselves, and add the router to its local default router list. A host can request an RA by sending out a **Router Solicitation (RS,** ICMP packet type 133) to the link-local all-routers multicast address. A RS is usually sent when a host is *not* currently configured with an IP address.

The RA messages contain the following information for hosts:

- The **router's link-layer address** (to be added to the host's default router list)
- One or more **network prefixes**
- A **lifetime** (measured in seconds) for the prefix(es)
- The link **MTU**

Routers send **Redirect** messages to hosts, indicating a *better* route to a destination. Hosts can have multiple routers in its default router list, but one is chosen as the *true* default router. If this default router deems that another router has a better route to the destination, it forwards the Redirect message to the sending host.

* * *

Neighbor Discovery Protocol (NDP) and ICMPv6 (continued)

Neighbor Solicitations (NS's, ICMP packet type 135**)** are sent by hosts to identify the link-layer address of a neighbor, and ensure its reachability. A NS message's source address is the link-local address of the sending host, and the destination is the **solicited-node multicast** address of the destination host.

A neighbor will reply to a NS with a **Neighbor Advertisement (NA,** ICMP packet type 136). This process replaces the Address Resolution Protocol (ARP) used by IPv4, and provides a far more efficient means to learn neighbor address information.

Hosts additionally use the NS messages to **detect duplicate addresses**. Before a host assigns itself an IPv6 address, it sends out a NS to ensure no other host is configured with that address.

Autoconfiguration of Hosts

Hosts can be assigned IPv6 addresses one of two ways: manually, or using autoconfiguration. Hosts learn how to autoconfigure themselves from **Router Advertisements (RA's).**

Two types of autoconfiguration exist, **stateless** and **stateful**.

When using **Stateless Autoconfiguration**, a host first assigns itself a link-local IPv6 address. It accomplishes this by combining the link-local prefix (FE8) with its interface ID (MAC address in EUI-64 format).

The host then sends a **Router Solicitation** multicast to the all-routers multicast address, which provides one or more network prefixes. The host combines these prefixes with its interface ID to create its site-local (or aggregate global) IPv6 addresses.

Stateful Autoconfiguration is used in conjunction with stateless autoconfiguration. Stateful Autoconfiguration utilizes DHCPv6 to provide additional information to the host, such as DNS servers. DHCPv6 can also be used in the event that there is no router on the link, to provide stateless autoconfiguration.

* * *

Section 9
- Introduction to 802.11 Wireless -

802.11 Overview

In the mid 1990's, the IEEE LAN/MAN committee began developing a series of **Wireless Local Area Network (WLAN)** standards. Collectively, these wireless standards are identified as the **802.11 standard.**

Note: The 802.11 standard is occasionally referred to as **Wi-Fi**, though the term 'Wi-Fi' has been applied to other wireless standards as well.

Various **amendments** have been made to the 802.11 standard. These are identified by the letter appended to the standard, such as 802.11*a* or 802.11*g*. The 802.11 amendments will be covered in greater detail later in this guide.

Wireless devices communicate across a specific range of **RF frequencies** known as a **channel**, using an antenna off of a **radio card**. 802.11 antennas come in several forms:

- Omnidirectional
- Semi-directional
- Highly-directional

A group of communicating 802.11 wireless devices is known as a **service set.** A wireless client can connect point-to-point with another wireless client – this is referred to as an **ad-hoc** connection, or an **Independent Basic Service Set (IBSS).**

More commonly, wireless client are centrally connected via a **wireless access point (WAP).** This is referred to as an **infrastructure** connection, or a **Basic Service Set (BSS)**. Wireless clients must **associate** with a WAP before data can be forwarded. WAPs often serve as a *gateway* between the wired and wireless networks.

In environments where a single WAP does not provide sufficient coverage, multiple WAPs can be *linked* as part of an **Extended Service Set (ESS)**.

* * *

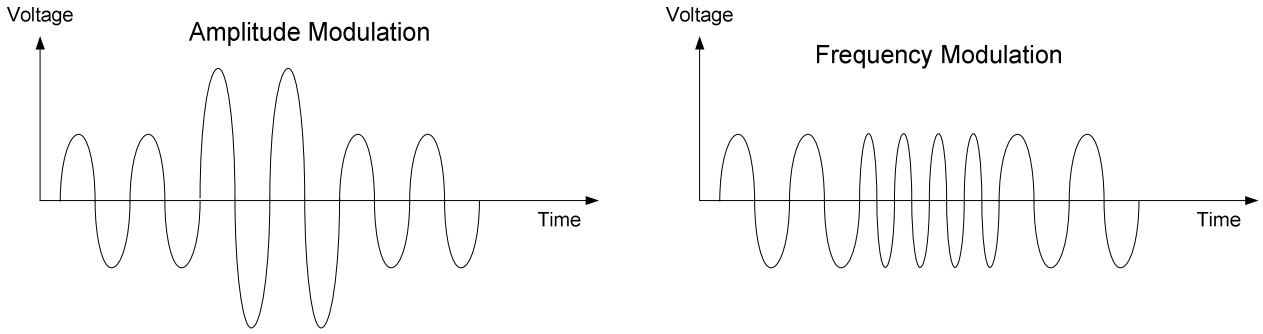
Radio Frequency Overview

Wireless communication is accomplished using **Radio Frequency (RF)** waves. **Frequency** is a measurement of the number of cycles completed per a given time period for an electromagnetic wave. The standard frequency measurement unit is the **hertz (Hz),** or one *cycle* per *second*.

Note: Ranges of frequencies are often identified by their specific use; these ranges are often referred to as **bands.**

Transmitting devices tune the signal to a specific frequency; receiving devices must tune to this frequency to receive the transmission. A signal at a specific frequency is referred to as the **carrier signal**. However, a carrier signal alone cannot contain data.

Modulation is the method of altering a signal to convey a message or data stream, usually by varying its **amplitude, frequency,** or **phase.**



802.11 devices employ multiple advanced modulation techniques, depending on the 802.11 amendment. This modulation requires that 802.11 devices communicate on a small subset of frequencies (referred to as a **channel)** varying around the carrier signal.

Each 802.11 amendment operates in either the **2.4-GHz** or **5-GHz** band:

- The **2.4-GHz band** provides the greatest range, but is *unregulated* and shared with appliances like microwaves and cordless phones. This can result in interference and degraded performance. The 2.4-GHz band is a subset of the industrial, scientific, and medical (ISM) band.
- The **5-GHz band** is *regulated* and thus generally free of interference. However, signals at this frequency suffer from poor range and are easily obstructed by intermediary objects. The 5-GHz band is referred to as the Unlicensed National Information Infrastructure (UNII) band.

* * *

RF Signal Strength

RF signals will attenuate in the open air. The power output of the RF antenna dictates the signal strength, and the usable distance of the signal.

RF power output is not usually measured in absolute terms (such as Watts). Instead, it is measured in **decibels (dB),** as a *ratio* of power to a *reference point*. The reference point is usually one Watt (W) or one milliWatt (mW).

The resulting power measurements are **Decibel Watts (dBw)** and **Decibel milliWatts (dBm)**. One milliWatt of power output is represented as **0 dBm.**

Decibel measurements are *logarithmic* in nature. The formula for calculating power output in decibel form is as follows:

$$dB = 10\log_{10}(P_{signal} / P_{reference})$$

The abbreviation *P* is short for power. Using the above formula, a signal transmitting at an *absolute* power of 20 mW would be represented as having a *relative* output power of 13 dBm.

Because decibels measure a *ratio* of power, it is possible to have a *negative* value. A negative value indicates that the amount of power is *less* than the reference point. For example, .25 mW of absolute power would be represented as -6 dBm. Conversely, a positive value indicates that the amount of power is *more* than the reference point.

(Reference: CCNP BCMSN Official Exam Certification Guide 4th Edition. David Hucaby. Pages 452-457; http://en.wikipedia.org/wiki/Decibel)

* * *

RF Interference and Obstruction

In addition to open-air attenuation, RF signals are susceptible to **interference**, degrading the performance and integrity of the communication. As stated previously, other devices operating in the **same frequency range** can interfere with a signal.

Physical objects can also obstruct or alter the trajectory of a RF signal:

- **Reflection –** occurs when a signal *bounces off* of a reflective material, altering its intended trajectory (and sometimes back towards the sender). Metal objects and water often cause reflection. If the signal is reflected in multiple directions, it is referred to as **scattering.**
- **Refraction –** occurs when the trajectory of a signal is *bent* as it *passes through* an object, such as a wall.
- **Absorption –** occurs when the energy of a signal is *absorbed* as it passes through an object, such as a wall or a tree. This loss of energy degrades the strength of the signal.
- **Diffraction –** occurs when a signal *bends around* a signal-absorbing object. For example, a sufficiently-strong signal can bend around an obstructing building, or around a corner within a building. However, this weakens and negatively affects the trajectory of a point-to-point signal.

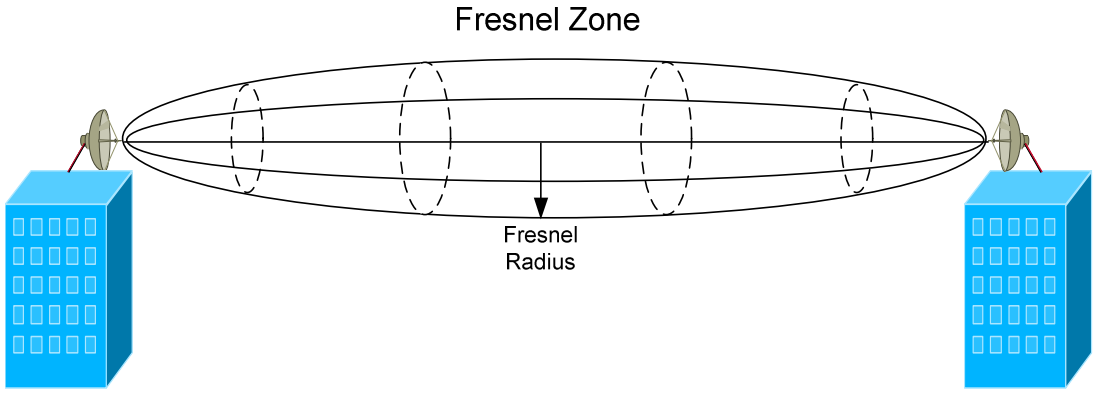
Because of diffraction, it is particularly important to maintain **line-of-sight** when employing a point-to-point wireless signal over a long distance. Buildings, trees, and even the curvature of the earth can obstruct the line-of-sight of the transmitting/receiving antennas.

(Reference: CCNP BCMSN Official Exam Certification Guide 4th Edition. David Hucaby. Pages 447-450, http://www.cisco.com/warp/public/102/wwan/quick-ref.pdf)

* * *

RF Fresnel Zones

Specifically, line-of-sight must be maintained within a signal's elliptical-shaped **Fresnel zone**.



If more than 40% of the lower radius of the Fresnel zone is obstructed, the signal will be negatively impacted from diffraction. Thus, it is imperative to maintain a *minimum* of 60% clearance in this radius.

Calculating the radius of a Fresnel zone requires a complex formula, which is beyond the scope of this guide. Various factors must be accounted for, such as atmospheric refraction, the curvature of the earth, frequency, and the relative heights of the two antennas.

A free Fresnel zone calculator is available online at:

http://www.afar.net/fresnel-zone-calculator/

(Reference: CCNP BCMSN Official Exam Certification Guide 4th Edition. David Hucaby. Pages 450-452)

* * *

802.11 Channels

Recall that all amendments to the 802.11 standard operate in one of two frequency bands:

- **2.4-GHz band** (specifically, 2.4000 to 2.4835GHz)
- **5.0-GHz band** (specifically, 5.150 to 5.825GHz)

The 2.4 GHz band supports a **total of 14 channels**, though the FCC limits this to **11 channels** in the United States. The *center* frequency of each channel is separated by only **5 MHz.**

Channel #	Center Frequency	Channel #	Center Frequency
1	2.412 MHz	8	2.447 MHz
2	2.417 MHz	9	2.452 MHz
3	2.422 MHz	10	2.457 MHz
4	2.427 MHz	11	2.462 MHz
5	2.432 MHz	12*	2.467 MHz
6	2.437 MHz	13*	2.472 MHz
7	2.442 MHz	14*	2.484 MHz

* Restricted in US

The 802.11 amendments that use the 2.4-GHz band (specifically, 802.11b and 802.11g) require a **22 MHz range** to modulate the signal. Thus, with each channel's center frequency separated by only 5 MHz, **channel overlap** will occur.

In fact, the 2.4-GHz band supports only **three non-overlapping channels**. Specifically, these are channels **1, 6,** and **11.** Devices competing on the same or adjacent channels will interfere with each other, degrading performance and reliability.

The less-often used 5-GHz band supports up to **12 non-overlapping channels** (in the U.S.), and is further separated into three sub-bands (with four channels each). The lower and middle bands are dedicated for *indoor* use, and the higher band is dedicated for *outdoor* use.

Remember that the 2.4-GHz band is *unregulated*, and the 5.0-GHz band is *regulated*.

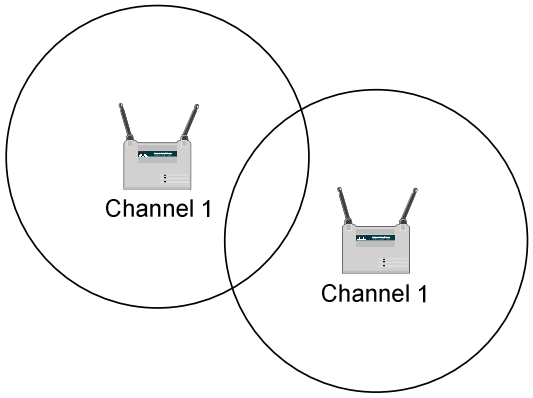
(Reference: http://en.wikipedia.org/wiki/List_of_WLAN_channels; http://www.cisco.com/en/US/docs/wireless/access_point/1200/vxworks/configuration/guide/bkscgaxa.html)

* * *

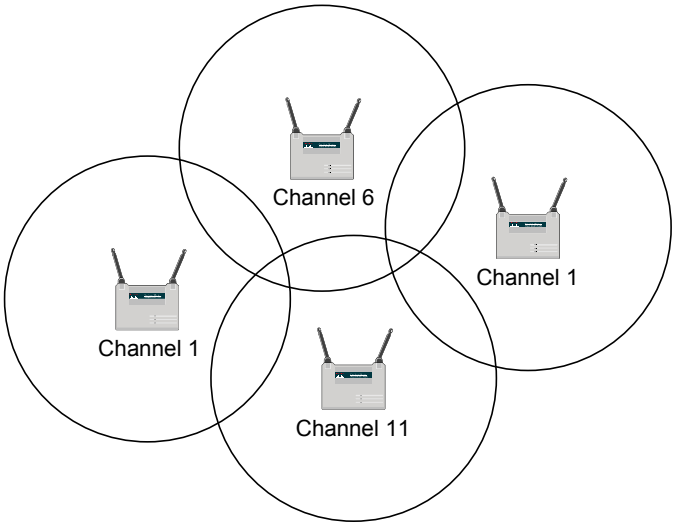
Preventing Channel Overlap

In large environments, a single WAP is often insufficient for full wireless coverage. Multiple WAPs can be linked together as part of an **Extended Service Set (ESS)**.

However, special considerations must be made when installing WAPs in close proximity to each other. Recall that only a limited number of non-overlapping channels are available in both the 2.4-GHz and 5.0-GHz bands.



Adjacent WAPs should *never* be configured on the same channel; the overlapping wireless fields will interfere with each other and *severely* degrade performance.



Providing full wireless coverage while preventing channel overlap can be challenging, especially if the environment has multiple floors. Performing a comprehensive **wireless site survey** is helpful in mapping out an accurate solution.

* * *

802.11 and Collisions

If two devices on a half-duplex Ethernet (802.3) network send a packet simultaneously, a **collision** will occur. Similarly, if two 802.11 wireless devices transmit simultaneously, their signals will mix resulting in unusable noise (essentially a *wireless collision*).

Half-duplex 802.3 Ethernet uses **Carrier Sense Multiple Access with Collision Detect (CSMA/CD)** to control media access. Devices monitor the physical link, and will only transmit a frame if the link is idle. When a collision is detected, both devices will wait a random amount of time before resending their respective packets.

All 802.11 connections are half-duplex. The only way to achieve full duplex is to send over one channel, and receive over another. The 802.11 standard currently has no such implementation.

802.11 devices have no method of *detecting* a collision, beyond the failure of the receiving device to send an **acknowledgement**. Instead, 802.11 devices attempt to avoid collisions using **Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA).** Devices will listen before attempting to transmit, and will *only* transmit if no other device is currently transmitting.

If another device *is* transmitting, other devices must wait until that transmission is finished, using a process called **Distributed Coordination Function (DCF)**. The currently transmitting device includes a **duration value** within the 802.11 header, informing other devices of the estimated time-length of its transmission.

Other 802.11 devices will not only wait out this duration value, but will wait an additional random amount of time (referred to as the **DCF interframe space (DIFS))**, before beginning their own transmissions. The random DIFS was implemented to prevent devices from transmitting simultaneously after waiting out another device's transmission duration. DIFS is often referred to as a **random back-off timer.**

(Reference: CCNP BCMSN Official Exam Certification Guide 4th Edition. David Hucaby. Pages 436-438, http://www.cisco.com/en/US/docs/voice_ip_comm/cuipph/7920/5_0/english/design/guide/wrlqos.html#wp1041341)

* * *

The 802.11 Amendments

The **original 802.11** standard was release in 1997, and utilized **direct-sequence spread spectrum (DSSS)** to modulate data onto an RF signal. The standard operated in the **2.4 GHz** frequency range, and had a maximum throughput of **2 Mbps**.

The original 802.11 standard never saw widespread adoption, and was quickly supplanted by the 802.11a and 802.11b **amendments**, which were developed concurrently and released in 1999.

802.11 wireless amendments that are currently in deployment include:

- **802.11a**
- **802.11b**
- **802.11g**
- **802.11n**

(Reference: http://en.wikipedia.org/wiki/IEEE_802.11_(legacy_mode))

802.11a

The **802.11a** amendment was released in 1999, and utilizes **orthogonal frequency-division multiplexing (OFDM)** for modulation. 802.11a operates in the **5.0-GHz** frequency band, and has a *maximum* throughput of **54 Mbps.** Specifically, 802.11a supports data rates of 6, 9, 12, 18, 24, 36, 48 and 54 Mbps, though the higher throughput is only available in close proximity to the wireless access point (WAP)/transmitter.

Because 802.11a operates in the regulated 5.0-GHz band, it is generally free of interference from other RF devices. However, the higher frequency reduces the effective distance of the signal, and is more susceptible to being absorbed by obstructing objects or walls.

802.11a is generally *not compatible* with other 802.11 amendments, as most of the other amendments operate in the 2.4-GHz band.

In the U.S., 802.11a supports a total of **12** non-overlapping channels, **4** of which can be used outdoors. Despite offering a large number of channels and good throughput, 802.11a did not see the same level of widespread deployment as the less expensive 802.11b and 802.11g amendments.

(Reference: http://en.wikipedia.org/wiki/IEEE_802.11a-1999)

* * *

802.11b

The **802.11b** amendment was also released in 1999, and utilizes **complementary code keying (CCK)** for modulation. 802.11b operates in the **2.4-GHz** frequency band, and has a *maximum* throughput of **11 Mbps.** Specifically, 802.11b supports data rates of 1, 2, 5.5, and 11 Mbps.

Because 802.11b operates in the unregulated 2.4-GHz band, it is susceptible to interference from other household RF devices.

In the U.S., 802.11b supports a total of **3** non-overlapping channels, specifically channels **1, 6,** and **11**.

(Reference: http://en.wikipedia.org/wiki/IEEE_802.11b-1999)

802.11g

The **802.11g** amendment was released in 2003, and utilizes **orthogonal frequency-division multiplexing (OFDM)** for modulation. 802.11g operates in the **2.4-GHz** frequency band, and has a *maximum* throughput of **54 Mbps.** Specifically, 802.11g supports data rates of 6, 9, 12, 18, 24, 36, 48, and 54 Mbps.

As with 802.11b, 802.11g operates in the unregulated 2.4-GHz band, and is susceptible to interference from other household RF devices.

In the U.S., 802.11g supports a total of **3** non-overlapping channels, specifically channels **1, 6,** and **11**.

802.11g is **backward-compatible** with 802.11b, as they both operate in the 2.4-GHz band. However, if an 802.11b device is present in an 802.11g environment, 802.11g will revert to CCK modulation, and will only support throughputs of 1, 2, 5.5, and 11 Mbps.

Neither 802.11b nor 802.11g are backward-compatible with 802.11a.

(Reference: http://en.wikipedia.org/wiki/IEEE_802.11g-2003)

* * *

802.11n

The **802.11n** amendment was officially released in 2009, though pre-release (or *draft*) equipment has been available since 2007. 802.11n supports significantly higher data rates than previous 802.11 amendments, through the use of wider channels (**40MHz** channels instead of 20MHz) and **Multiple-Input Multiple-Output (MIMO)**.

MIMO employs multiple antennas on both the transmitter and receiver. The resulting multiple data streams are then combined using **Spatial Division Multiplexing (SDM).** This, coupled with 40MHz channels, allows 802.11n to support throughput up to **600 Mbps.**

802.11n devices are identified by the number of *transmit* and *receive* antennas they support, with a format of $Antenna_{transmit}$ x $Antenna_{receive}$. For example, a WAP with four transmit and three receive antennas would be identified as a *4 x 3* MIMO WAP.

802.11n can operate in either the **2.4-GHz** *or* the **5.0-GHz** frequency bands, *or* both simultaneously. Thus, 802.11n is backwards compatible with 802.11a, 802.11b, and 802.11g. A pure 802.11n environment should operate in the 5.0-GHz band to maximize throughput and to limit interference.

Note also that the wider 40-MHz channel reduces the number of available non-overlapping channels in each band, which provides more incentive to use the 5.0-GHz band. 802.11n *does* support 20-MHz channels, though this will greatly reduce the maximum throughput.

(Reference: http://en.wikipedia.org/wiki/IEEE_802.11n-2009; http://www.airmagnet.com/assets/whitepaper/WP-802.11nPrimer.pdf; http://www.ciscosystems.sc/en/US/prod/collateral/wireless/ps5678/ps6973/ps8382/prod_white_paper0900aecd806b8ce7_ns767_Networking_Solutions_White_Paper.html)

The 802.11 Amendments – Quick Reference

	802.11	**802.11a**	**802.11b**	**802.11g**	**802.11n**
Max Throughput	2 Mbps	54Mbps	11Mbps	54Mbps	600Mbps
Modulation	DSSS	OFDM	CCK	OFDM	OFDM
Frequency Band	2.4GHz	5.0GHz	2.4GHz	2.4GHz	2.4/5.0GHz
Non-Overlapping Channels	-	12	3	3	Varies*
Released	1997	1999	1999	2003	2009

* Varies depending on the Frequency Band, and whether 20MHz or 40MHz channels are being utilized.

* * *

Associating with a Wireless Access Point (WAP)

Recall that a group of communicating 802.11 wireless devices is known as a **service set,** and that there are two *modes* of 802.11 communication:

- **Ad-hoc** or **Independent Basic Service Set (IBSS) –** where wireless clients communicate point-to-point with each other.
- **Infrastructure** or **Basic Service Set (BSS) –** where wireless clients communicate via a Wireless Access Point (WAP).

Wireless clients must **associate** with a WAP before data can be forwarded. Various parameters must match between the client and the WAP:

- **Service Set Identifier (SSID)**
- **Data Rate**
- **Authentication**
- **Encryption/Data Integrity**

The **SSID** is used to *identify* the wireless connection between a WAP (or WAPs) and clients. A wireless client must be configured with the WAP's SSID to associate with it. Otherwise, a client can also *request* (via a *probe*) the SSID if the WAP is configured to **broadcast** the SSID (via a *beacon*). As a best practice, broadcasting is usually disabled in secure environments.

The SSID is often mistaken as a security feature; however, the SSID does not authenticate users or encrypt data – it merely serves as an identifier for a wireless connection. The SSID also provides separation between multiple wireless LANs that might exist in an environment.

Wireless clients are often required to *authenticate* to a WAP. The original 802.11 standard provides for two methods of authentication:

- **Open Authentication -** authenticates *any* wireless client request.
- **Shared-Key Authentication –** requires a matching key to be configured on both the wireless client and WAP.

Open authentication (essentially, *no* authentication) is used for devices that cannot support a complex authentication process. Shared-key authentication employs **Wireless Equivalence Protocol (WEP)** keys for authenticating clients. WEP is covered in detail in the next section.

MAC-address filtering is an additional form of authentication, though not defined in the 802.11 standard. A list of allowed MAC addresses must be maintained on the WAP itself.

* * *

Wireless Equivalence Protocol (WEP)

The emergence of 802.11 technologies has introduced new security concerns, due to the open-air nature of wireless transmissions. Such transmissions are easily intercepted, which necessitates mechanisms to not only *authenticate* wireless clients, but also to *secure* data transfer (using **encryption)** and to ensure *data integrity* (using a **32-bit CRC**)**.**

Wireless Equivalence Protocol (WEP) was developed as part of the original 802.11 standard. WEP utilizes the **RC4 stream cipher** for encryption, which combines a **key** with a randomly-generated **initialization vector (IV)** to provide confidentiality.

WEP comes in two common forms:

- **64-bit WEP** – employs a *40-bit* key with a *24-bit* IV.
- **128-bit WEP –** employs a *104-bit* key with a *24-bit* IV.

The 128-bit WEP key is represented as a 26-digit hexadecimal string.

WEP can be used with both Open and Shared-Key authentication. With Open authentication, the WEP key is used only for encrypting data. With Shared-Key authentication, the WEP key used for *both* authenticating the wireless client and encrypting data. Regardless of the authentication method, the WEP key(s) **must be identical** on both the wireless client and the WAP.

WEP Shared-Key authentication employs a four-way handshake:

1. The client makes an authentication request to the WAP.
2. The WAP responds with a clear-text *challenge.*
3. The client *encrypts* the challenge with its WEP key.
4. The WAP *decrypts* the encrypted challenge and compares it to the original clear-text challenge.

The authentication process will only be successful if the WEP key is identical on both the WAP and the client. Surprisingly, Shared-Key authentication is less secure than Open authentication. A malicious attacker can intercept both the clear-text and encrypted challenges, and thus somewhat easily derive the encryption key.

WEP is no longer considered a viable security mechanism, as it is *easily* compromised. Additionally, WEP provides only one-way authentication; there is no mechanism within WEP for a client to authenticate the WAP.

(Reference: http://en.wikipedia.org/wiki/Wired_Equivalent_Privacy; http://www.wi-fiplanet.com/tutorials/article.php/1368661)

* * *

Wi-Fi Protected Access (WPA)

Wi-Fi Protected Access (WPA) was developed by the Wi-Fi Alliance to address the shortcomings of WEP. WPA incorporates some of the techniques and protocols that were eventually standardized as part of the **802.11i amendment**.

Temporal Key Integrity Protocol (TKIP) is the core component of WPA. Though TKIP employs a **RC4 stream cipher** like WEP, it offers several improvements, including:

- Per-Packet Key Hashing
- 64-bit Message Integrity Check (MIC)
- Broadcast Key Rotation
- Sequence Counting

Note: Cisco developed a proprietary implementation of TKIP that is *not compatible* with WPA TKIP. However, Cisco devices will often support both the standardized and propriety forms of TKIP.

WPA2, also developed by the Wi-Fi Alliance, incorporates *all* portions of the 802.11i amendment. It added support for **Advanced Encryption Standard (AES)** encryption with **Cipher Block Chaining Message Authentication Code Protocol (CCMP)**. AES-CCMP is considered *significantly* more secure than the RC4 stream cipher used by WEP/TKIP. WPA2 also added native support for Intrusion Detection Systems (IDS).

Both WPA and WPA2 support two *modes,* **Personal** and **Enterprise.**

WPA Personal employs pre-shared key (or passphrase) for authentication, and is often referred to as WPA-PSK (Pre-Shared Key). The WPA key can be represented as a 64-digit hexadecimal string, or an 8 to 63 character ASCII string. As with WEP, this key-string must be identical on both the client and the WAP.

WPA Enterprise employs an 802.1X/EAP server (such as a RADIUS server) for centralized authentication. An authentication server eliminates the need for static encryption/authentication keys to be configured on both the client and the WAP.

802.1X/EAP authentication is covered in detail in the next sections.

(Reference: http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access; http://www.cisco.com/en/US/prod/collateral/wireless/ps5678/ps430/ps4076/prod_brochure09186a00801f7d0b.html)

* * *

802.1X and Extensible Authentication Protocol (EAP)

The **802.1X standard** was developed by the IEEE to authenticate devices on a Layer-2 port basis. It was originally developed for Ethernet (802.3) bridges and switches, but was expanded to support the authentication of 802.11 wireless devices as well.

802.1X defines three *roles* in the authentication process:

- **Supplicant** – the device being authenticated. In an 802.11 environment, the supplicant would be the wireless client software.
- **Authenticator** – the device that is *requiring* the authentication. In an 802.11 environment, this is often the WAP.
- **Authentication Server** – the device that stores the user database, for validating authentication credentials. This is often an external RADIUS server, though some WAPs support a local user database.

802.1X provides the *encapsulation* of **Extensible Authentication Protocol (EAP)** traffic, which serves as the *framework* for authenticating clients. EAP is not an authentication mechanism in itself. Instead, EAP transports the authentication data between supplicants, authenticators, and authentication servers (all three of which must support 802.1X/EAP).

As a general framework, EAP supports a large number of *methods* for authentication, including (but not limited to):

- **Lightweight EAP (LEAP)**
- **EAP - Flexible Authentication via Secure Tunneling (EAP-FAST)**
- **EAP - Transport Layer Security (EAP-TLS)**
- **Protected EAP (PEAP)**

With any form of EAP, wireless clients *must* authenticate with a RADIUS server before any data traffic will be forwarded. Only EAP traffic is allowed between the client and WAP before authentication occurs.

Authenticating clients using 802.1X/EAP offers several advantages over Static-WEP and WPA-PSK, including:

- Centralized management of credentials
- Support for multiple encryption types
- Dynamic encryption keys

(Reference: http://en.wikipedia.org/wiki/IEEE_802.1X; http://en.wikipedia.org/wiki/Extensible_Authentication_Protocol; http://www.ieee802.org/1/files/public/docs2000/P8021XOverview.PDF)

* * *

Lightweight Extensible Authentication Protocol (LEAP)

Lightweight Extensible Authentication Protocol (LEAP) was developed by Cisco, and is supported by WPA/WPA2 as an 802.1X authentication method. LEAP employs a username/password for authentication via a RADIUS server, and does *not* require the use of certificates.

LEAP is supported by most operating system, including Mac OS, Linux, DOS, and most versions of Windows. LEAP additionally supports single sign-on in Windows environments, allowing clients to perform Active Directory (or NT Domain) and 802.1X authentication simultaneously.

LEAP authentication is a multi-step process:

1. The supplicant initiates the connection with a *Start* message.
2. The authenticator responds with a *Request/Identity* message.
3. The supplicant responds with an *Identity* message containing a username.
4. The authenticator then forwards the username to the authentication server with an *Access Request* message.
5. The supplicant and authentication server then authenticate *each other* using a challenge/response method. The authentication server sends a randomly-generated *challenge* to the supplicant. The supplicant then generates a hash value from the challenge and its password, using MD5. This hash value serves as the *response* back to the authentication server, and eliminates the need for the actual password to be transmitted between the two devices.
6. A *Success* message is generated if the supplicant and authentication server have successfully authenticated each other, which informs the authenticator that the supplicant can now pass data traffic.

Once authentication is completed, the supplicant and authentication server then generate a **pairwise master key (PMK).** The PMK is used to create the actual encryption keys for data transfer, via a four-way handshake.

LEAP was built on a variation of MS-CHAP, and is thus vulnerable to dictionary attacks. A strong password policy is extremely important when employing LEAP in a business environment. If strong passwords are not possible, Cisco recommends utilizing EAP-FAST instead of LEAP.

(Reference: http://www.ciscosistemi.com/en/US/prod/collateral/wireless/ps5678/ps430/prod_qas0900aecd801764f1.html; http://www.cisco.com/application/pdf/en/us/guest/netsol/ns386/c649/ccmigration_09186a0080871da5.pdf; *CCNP ONT Exam Certification Guide, Amir Ranjbar. Pages 262-264*)

* * *

EAP with Flexible Authentication via Secure Tunneling (EAP-FAST)

EAP-FAST was also developed by Cisco as an alternative to LEAP, and was standardized by the IETF. Like LEAP, it utilizes a username/password for authentication via a RADIUS server, and does *not* require the use of certificates. Unlike LEAP, EAP-FAST is not vulnerable to dictionary attacks, as it establishes a secure tunnel between the supplicant and authentication server.

EAP-FAST is supported by most versions of Windows, and supports Windows single sign-on in Active Directory/Domain environments.

EAP-FAST authentication is a three-phase process:

- **Phase 0** (*optional)* – the supplicant is assigned a **Protected Access Credential (PAC),** on a per-user basis**.** This phase is optional because the PAC can be manually configured on the supplicant.
- **Phase 1 –** the supplicant and authentication server establish a secure tunnel using the PAC.
- **Phase 2 –** the supplicant sends its username/password credentials to the authentication server, via the secure tunnel.

(Reference: http://www.cisco.com/en/US/prod/collateral/wireless/ps5679/ps5861/prod_qas09186a00802030dc_ps430_Products_Q_and_A_Item.html; *CCNP ONT Exam Certification Guide, Amir Ranjbar. Pages 264-266*)

EAP with Transport Layer Security (EAP-TLS)

EAP with **Transport Layer Security (EAP-TLS)** is an IETF standard protocol, and was the *first* EAP authentication method used with 802.11 wireless networks.

EAP-TLS utilizes Public Key Infrastructure (PKI) to authenticate supplicants using certificates. Both the supplicant *and* the authentication server must be assigned a certificate from a Certificate Authority (CA) server. Because of this, EAP-TLS is considered *extremely* secure, though the complexity of client-side certificates makes it somewhat unpopular.

EAP-TLS is natively supported on most versions of Windows (2000 and newer).

(Reference: http://www.cisco.com/en/US/tech/tk722/tk809/technologies_white_paper09186a008009256b.shtml; *CCNP ONT Exam Certification Guide, Amir Ranjbar. Pages 266-267*)

* * *

Protected EAP (PEAP)

Protected EAP (PEAP) was developed jointly by Cisco, Microsoft, and RSA Security, and was submitted to the IETF for standardization.

PEAP utilizes TLS to create a secure tunnel between the supplicant and authentication server. The key difference between PEAP and EAP-TLS is that only the *authentication server* requires a PKI certificate – no certificate is required on the supplicant.

PEAP authentication is a two-phase process:

1. The supplicant authenticates the authentication server by verifying the server-side PKI certificate. If successful, the supplicant and authentication server form the TLS tunnel.
2. The supplicant sends its username/password credentials to the authentication server, via the secure tunnel. This is accomplished using either EAP-MSCHAPv2 (for Windows-based authentication servers) or EAP-GTC (Generic Token Card, for LDAP-based authentication servers).

As with the other EAP-methods, a *Success* message is generated if the supplicant and authentication server have successfully authenticated each other, which informs the authenticator that the supplicant can pass traffic.

(Reference: http://www.cisco.com/en/US/products/ps6366/products_configuration_example09186a0080921f67.shtml; *CCNP ONT Exam Certification Guide, Amir Ranjbar. Pages 267-269*)

* * *

Part II

The Cisco IOS

---

* * *

Section 10
- Router Components -

Router Memory Components

Cisco routers (and switches) generally contain four types of memory:

- **ROM (Read-Only Memory)**
- **Flash**
- **NVRAM (Non-Volatile RAM)**
- **RAM (Random-Access Memory)**

ROM contains a **bootstrap** program called **ROM Monitor (**or **ROMmon)**. When a router is powered on, the bootstrap runs a hardware diagnostic called **POST (Power-On Self Test)**.

If POST completes successfully, the bootstrap then attempts to locate and load the Cisco **IOS (Internetwork Operating System)** stored in **Flash memory.** Flash memory *can* be erased or overwritten, thus making the Cisco IOS upgradeable. The Cisco IOS is covered in great detail in other guides.

If the bootstrap *cannot* find the IOS in Flash, a stripped-down version of the IOS that will be loaded from ROM instead. The contents of ROM cannot be altered or erased; the entire ROM chip must be replaced if an upgrade/repair is necessary.

If the bootstrap *does* find the IOS in Flash, it is loaded into RAM and attempts to find a **Startup Configuration** (**startup-config)** file in **NVRAM**. NVRAM is non-volatile, thus its contents will survive a power-cycle.

If the IOS *cannot* find a startup-config file in NVRAM, it will attempt to load a configuration file from a **TFTP** server (this request is **broadcasted** to 255.255.255.255). If no TFTP server responds, the IOS will enter **Initial Configuration Mode**, a series of interactive questions intended for quick configuration of the router.

If the IOS *does* find a startup-config file in NVRAM, this file is loaded into **RAM**, and becomes the **Running Configuration (running-config)**. RAM is a volatile memory, and thus its contents will be lost if the router is power-cycled.

* * *

Router Memory, Quick Reference

The following table details each of the basic types of **router memory**:

Memory	***Writable?***	***Volatile?***	***Function***
ROM	No	No	*Stores bootstrap*
Flash	Yes	No	*Stores IOS*
NVRAM	Yes	No	*Stores startup-config*
RAM	Yes	Yes	*Stores running-config*

The Router Boot-Process, Quick Reference

The following details the **router boot process**:

1. The router is powered on.
2. The bootstrap program (ROMmon) is loaded from ROM.
3. The bootstrap runs POST.
4. The bootstrap attempts to load the IOS from Flash.
    a. If the IOS *is not* found in Flash, the bootstrap loads into RAM the basic IOS stored in ROM.
    b. If the IOS *is* found in Flash, it is loaded into RAM.
5. The IOS attempts to load the startup-config file from NVRAM
    a. If the startup-config *is not* found in NVRAM, the IOS attempts to load a configuration file from TFTP.
    b. If no TFTP server responds, the router enters Initial Configuration Mode.
    c. If the startup-config *is* found in NVRAM, it is loaded into RAM.
6. The startup-config becomes the running-config in RAM.

Interfaces vs Lines

Cisco devices contain two distinctly different types of ports, **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 (but are not limited to):

- Serial interfaces
- Ethernet interfaces
- Fast Ethernet interfaces
- Token Ring interfaces
- ATM interfaces

Interfaces are identified by both the *type* of interface, and the **interface number** (which always begins at "0"). Thus, the first Ethernet interface on a router would be identified as *Ethernet0.*

Certain router families (such as the 3600 series) are modular, and have multiple "slots" for interfaces. Thus, interfaces on these routers are identified by both the **module number** *and* the interface number, formatted as: *module/interface.* Thus, the third Fast Ethernet interface on the first modular slot would be identified as *FastEthernet0/2*.

Lines identify ports that allow us to connect into, and then configure, Cisco devices. The most common examples of lines include:

- Console ports
- Auxiliary ports
- VTY (telnet) ports

Just like interfaces, lines are identified by both the *type* of line, and the line *number* (again, always begins at "0"). Thus, the first console port on a router would be identified as *Console0*.

* * *

Using Lines to Configure the IOS

As mentioned previously, three methods (or *lines*) exist to configure Cisco IOS devices:

- Console ports
- Auxiliary ports
- VTY (telnet) ports

Nearly every modern Cisco router or switch includes a **console port,** sometimes labeled on the device simply as *con***.** The console port is generally a RJ-45 connector, and requires a **rollover** cable to connect to. The opposite side of the rollover cable connects to a PC's serial port using a serial **terminal adapter**.

From the PC, software such as HyperTerminal is required to make a connection from the local serial port to the router console port. The following settings are necessary for a successful connection:

- Bits per second - *9600 baud*
- Data bits - *8*
- Parity - *None*
- Stop bits - *1*
- Flow Control - *Hardware*

Some Cisco devices include an **auxiliary port,** in addition to the console port. The auxiliary port can function similarly to a console port, and can be accessed using a rollover cable. Additionally, auxiliary ports support modem commands, thus providing dial-in access to Cisco devices.

Telnet, and now **SSH**, are the most common methods of remote access to routers and switches. The standard edition of the IOS supports up to **5 simultaneous** VTY connections. Enterprise editions of the IOS support up to **255 VTY** connections.

There are two requirements before a router/switch will accept a VTY connection:

- An **IP address** must be configured on an interface
- At least one VTY port must be configured with a **password**

Section 11
- Introduction to the Cisco IOS -

Cisco IOS

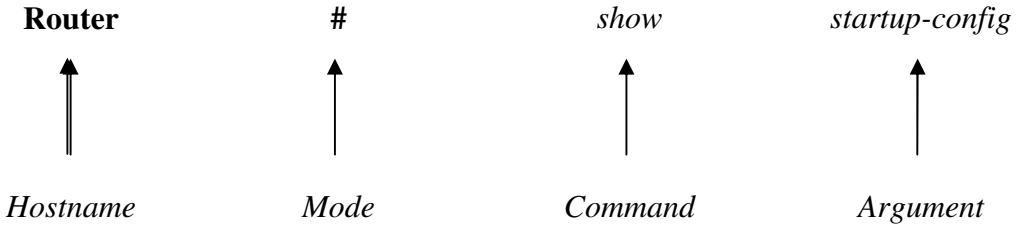
The **Cisco IOS (Internetwork Operating System)** is a command-line interface used by nearly all current Cisco routers and Catalyst switches. The IOS provides the mechanism to configure all Layer 2 and Layer 3 functions on Cisco devices.

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*)

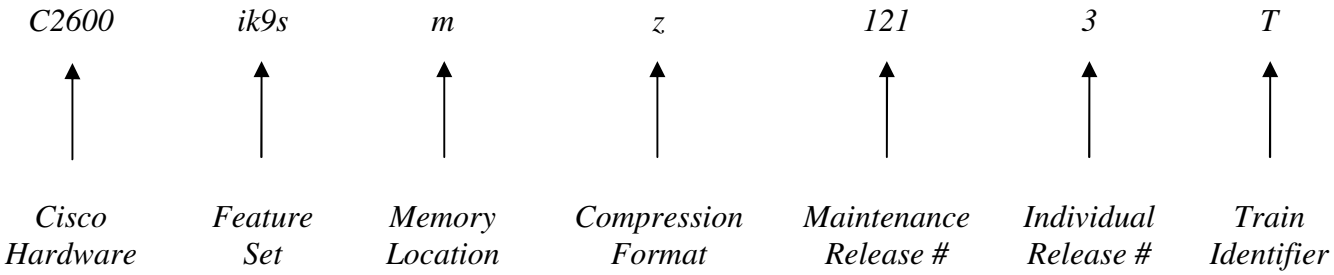
* * *

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 12
- Advanced IOS Functions -

The Configuration Register

The **configuration register** (**config-register**) is a hexadecimal value that controls various aspects of how a router boots, including:

- Baud Rate
- Boot Messages (enable/disable)
- Break (disable/ignore)
- Flash (read-only, read-write)
- NVRAM (use startup-config/bypass startup-config)

The **default** config-register is **0x2102.** To view your router's current config-register setting:

Router# *show version*

```
<snip>

32K bytes of non-volatile configuration memory.
16384K bytes of processor board System flash (Read ONLY)

Configuration register is 0x2102
```

Common config-register settings include:

Value	*Baud Rate*	*Boots Into?*	*Flash*	*Startup-Config*
0x2101	9600	IOS from ROM	Read/Write	Uses
0x2102	9600	IOS from Flash	Read/Only	Uses
0x2142	9600	IOS from Flash	Read/Only	Bypass

Remember, **0x2102** is the default config-register value on Cisco routers. In order to upgrade the Cisco IOS, the config-register must be changed to **0x2101**, so that the Flash memory becomes writeable.

To change the config-register from the IOS:

Router(config)# *config-register 0x2142*

This configuration change **does not take affect** until the **next reboot**.

(Reference: *http://www.cisco.com/en/US/products/hw/routers/ps133/products_tech_note09186a008022493f.shtml*)

* * *

Backing up and Restoring the Cisco IOS

The Cisco IOS is stored in flash. Multiple IOS files can be loaded into flash, assuming there is enough free space. You can view available free space, and the name of any file(s) in flash, by typing:

Router# *show flash*

```
System flash directory:
File   Length   Name/status
  1    4467254  c2500-ik9s-mz.122-4

[4467254 bytes used, 12309962 available, 16777216 total]
16384K bytes of processor board System flash (Read/Write)
```

To backup the IOS, a TFTP server is required. The TFTP server must have IP connectivity to the router.

To backup the IOS file from the router's flash to the TFTP server:

Router# *copy flash tftp*

You will be prompted for the following information:

- Address of remote host (the TFTP server)
- Source file name (the name of the file in flash)
- Destination file name

To load an IOS file from a TFTP server to the router's flash:

Router# *copy tftp flash*

The process is nearly identical to copy a startup-configuration file to or from a router's NVRAM:

Router# *copy startup tftp*
Router# *copy tftp startup*

* * *

CDP (Cisco Discovery Protocol)

CDP is a Cisco propriety protocol used to collect information about locally attached Cisco switches and routers. CDP is enabled by default on all IOS enabled routers and switches, and sends out updates every **60 seconds.**

CDP will provide the following information about directly connected neighbors:

- **Device ID –** hostname of remote router/switch
- **Local Interface –** interface that remote router/switch is directly connected to
- **Holdtime** – amount of time before remote device information is purged from CDP table
- **Capability** – Type of remote device (router, switch, host)
- **Platform** – Model of remote device

To view CDP timers and hold-down information:

Router# *show cdp*

To display neighbor information:

Router# *show cdp neighbors*

To display traffic statistics:

Router# *sh cdp traffic*

To display port and interface information:

Router# *sh cdp interface*

To disable CDP on an interface:

Router(config-if)# *no cdp enable*

To globally disable CDP:

Router(config)# *no cdp run*

* * *

Telnet

It is possible to telnet *from* a router into a remote device, using either the hostname or IP address of the remote device:

Router# *telnet Router2*
Router# *telnet 172.17.1.2*

To exit a telnet session:

Router2# *exit*

To return to the router you telnetted *from*, without exiting the session:

- Hold *CTRL+SHIFT+6* and then release
- Type the character *x*

To view all open telnet sessions:

Router# *show sessions*

Privilege Levels

IOS devices have a total of **16 privilege levels**, numbered 0 through 15. **User Exec** mode is privilege level 1. **Privileged Exec** mode is privilege level 15.

We can create a custom Privilege level, including the commands users are allowed to input at that mode:

Router(config)# *privilege exec all level 3 show interface*
Router(config)# *privilege exec all level 3 show ip route*
Router(config)# *privilege exec all level 3 show reload*

To then enter that privilege level from User Mode:

Router> *enable 3*

Observing Performance Statistics on Cisco Routers

To view the processor load on a Cisco Router:

Router# *show processes cpu*

Part III

Routing

---

* * *

Section 13
- The Routing Table -

Routing Table Basics

Routing is the process of sending a packet of information from one *network* to another *network.* Thus, routes are usually based on the destination network, and not the destination host (host routes *can* exist, but are used only in rare circumstances).

To route, routers build Routing Tables that contain the following:

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

The routing table is concerned with two types of protocols:

- A **routed** protocol is a layer 3 protocol that applies logical addresses to devices and routes data between networks. Examples would be IP and IPX.
- A **routing** protocol dynamically builds the network, topology, and next hop information in routing tables. Examples would be RIP, IGRP, OSPF, etc.

To determine the *best* route to a destination, a router considers three elements (in this order):

- **Prefix-Length**
- **Metric** (*within* a routing protocol)
- **Administrative Distance** *(between* separate routing protocols)

Prefix-length is the number of bits used to identify the network, and is used to determine the most *specific* route. A longer prefix-length indicates a more specific route. For example, assume we are trying to reach a host address of 10.1.5.2/24. If we had routes to the following networks in the routing table:

10.1.5.0/24
10.0.0.0/8

The router will do a bit-by-bit comparison to find the most *specific* route (i.e., longest matching prefix). Since the 10.1.5.0/24 network is more specific, that route will be used*,* ***regardless of metric or Administrative Distance.***

* * *

Administrative Distance vs. Metric

A "**metric**" allows a router to choose the best path *within* a routing protocol. Distance vector routing protocols use "**distance**" (usually hop-count) as their metric. Link state protocols utilize some sort of "**cost**" as their metric.

Only routes with the **best metric** are **added to the routing table**. Thus, even if a particular routing protocol (for example, RIP) has four routes to the same network, only the route with the *best* metric (hop-count in this example) would make it to the routing table. If multiple equal-metric routes exist to a particular network, most routing protocols will load-balance.

If your router is running multiple routing protocols, **Administrative Distance** is used to determine which routing protocol to *trust* the most. *Lowest* administrative distance wins.

Again: if a router receives two RIP routes to the same network, it will use the routes' **metric** to determine which path to use. If the metric is identical for both routes, the router will load balance between both paths.

If a router receives a RIP and an OSPF route to the same network, it will use **Administrative Distance** to determine which routing path to choose.

The Administrative Distance of common routing protocols (remember, lowest wins):

Connected	0
Static	1
EIGRP Summary	5
External BGP	20
Internal EIGRP	90
IGRP	100
OSPF	110
IS-IS	115
RIP	120
External EIGRP	170
Internal BGP	200
Unknown	255

A route with an "unknown" Administrative Distance will never be inserted into the routing table.

* * *

Viewing the routing table

The following command will allow you to view the routing table:

Router# *show ip route*

```
Gateway of last resort is 192.168.1.1 to network 0.0.0.0

C     192.168.1.0/24 is directly connected, Ethernet0
      150.50.0.0/24 is subnetted, 1 subnets
C     150.50.200.0 is directly connected, Loopback1
C     192.168.123.0 is directly connected, Serial0
C     192.168.111.0 is directly connected, Serial1
R     10.0.0.0 [120/1] via 192.168.123.1, 00:00:00, Serial0
               [120/1] via 192.168.111.2, 00:00:00, Serial1
S*    0.0.0.0/0 [1/0] via 192.168.1.1
```

Routes are labeled based on what protocol placed them in the table:

- C – Directly connected
- S – Static
- S* - Default route
- D - EIGRP
- R – RIP
- I – IGRP
- i – IS-IS
- O - OSPF

Notice the RIP routes contain the following field: **[120/1]**. This indicates both the administrative distance and the metric (the *120* is the AD, and the *1* is the hop-count metric).

To clear all routes from the routing table, and thus forcing any routing protocol to repopulate the table:

Router# *clear ip route **

* * *

Choosing the Best Route (Example)

Assume the following routes existed to the following host: 192.168.111.5/24

```
O     192.168.111.0/24 [110/58] via 192.168.131.1, 00:00:00, Serial3
R     192.168.111.0/24 [120/1] via 192.168.123.1, 00:00:00, Serial0
R     192.168.111.0/24 [120/5] via 192.168.5.2, 00:00:00, Serial1
S     192.168.0.0/16 [1/0] via 10.1.1.1
```

We have two RIP routes, an OSPF route, and a Static route to that destination. Which route will be chosen by the router?

Remember the three criteria the router considers:

- **Prefix-Length**
- **Metric**
- **Administrative Distance**

The static route has the lowest administrative distance (**1**) of any of the routes; however, its **prefix-length** is less specific. 192.168.111.0/24 is a more specific route than 192.168.0.0/16. Remember, prefix-length is *always* considered first.

The second RIP route **will not be inserted** into the routing table, because it has a higher metric (**5**) than the first RIP route (**1**). Thus, our routing table will actually look as follows:

```
O     192.168.111.0/24 [110/58] via 192.168.131.1, 00:00:00, Serial3
R     192.168.111.0/24 [120/1] via 192.168.123.1, 00:00:00, Serial0
S     192.168.0.0/16 [1/0] via 10.1.1.1
```

Thus, the true choice is between the OSPF route and the first RIP route. OSPF has the lowest administrative distance, and thus that route will be preferred.

PLEASE NOTE: Calculating the lowest metric route within a routing protocol occurs *before* administrative distance chooses the route it "trusts" the most. This is why the order of the above "criteria" is prefix-length, metric, and *then* administrative distance.

However, the route with the lowest administrative distance is *always* preferred, regardless of metric (assuming the prefix-length is equal). Thus, the metric is *calculated* first, but not *preferred* first over AD.

* * *

Section 14
- Static vs. Dynamic Routing -

Static vs. Dynamic Routing

There are two basic methods of building a routing table: **Statically** or **Dynamically**.

A **static** routing table is created, maintained, and updated by a network administrator, *manually*. A static route to *every* network must be configured on *every* router for full connectivity. This provides a granular level of control over routing, but quickly becomes impractical on large networks.

Routers will *not* share static routes with each other, thus reducing CPU/RAM overhead and saving bandwidth. However, static routing is *not fault-tolerant*, as any change to the routing infrastructure (such as a link going down, or a new network added) requires manual intervention. Routers operating in a purely static environment cannot seamlessly choose a better route if a link becomes unavailable.

Static routes have an Administrative Distance (AD) of **1**, and thus are always preferred over dynamic routes, unless the default AD is changed. A static route with an adjusted AD is called a **floating static route**, and is covered in greater detail in another guide.

A **dynamic** routing table is created, maintained, and updated by a *routing protocol* running on the router. Examples of routing protocols include **RIP** (Routing Information Protocol), **EIGRP** (Enhanced Interior Gateway Routing Protocol), and **OSPF** (Open Shortest Path First). Specific dynamic routing protocols are covered in great detail in other guides.

Routers *do* share dynamic routing information with each other, which increases CPU, RAM, and bandwidth usage. However, routing protocols are capable of dynamically choosing a different (or better) path when there is a change to the routing infrastructure.

Do not confuse *routing* protocols with *routed* protocols:

- A ***routed*** protocol is a Layer 3 protocol that applies logical addresses to devices and routes data between networks (such as IP)
- A ***routing*** protocol dynamically builds the network, topology, and next hop information in routing tables (such as RIP, EIGRP, etc.)

* * *

Static vs. Dynamic Routing (continued)

The following briefly outlines the advantages and disadvantages of *static* routing:

Advantages of Static Routing	• Minimal CPU/Memory overhead • No bandwidth overhead (updates are not shared between routers) • Granular control on how traffic is routed
Disadvantages of Static Routing	• Infrastructure changes must be manually adjusted • No "dynamic" fault tolerance if a link goes down • Impractical on large network

The following briefly outlines the advantages and disadvantages of *dynamic* routing:

Advantages of Dynamic Routing	• Simpler to configure on larger networks • Will dynamically choose a different (or better) route if a link goes down • Ability to load balance between multiple links
Disadvantages of Dynamic Routing	• Updates are shared between routers, thus consuming bandwidth • Routing protocols put additional load on router CPU/RAM • The choice of the "best route" is in the hands of the routing protocol, and not the network administrator

* * *

Dynamic Routing Categories

There are two distinct categories of dynamic routing protocols:

- **Distance-vector protocols**
- **Link-state protocols**

Examples of distance-vector protocols include **RIP** and **IGRP**. Examples of link-state protocols include **OSPF** and **IS-IS**.

EIGRP exhibits both distance-vector and link-state characteristics, and is considered a *hybrid* protocol.

Distance-vector Routing Protocols

All **distance-vector** routing protocols share several key characteristics:

- **Periodic** updates of the **full** routing table are sent to routing neighbors.
- Distance-vector protocols suffer from slow convergence, and are highly susceptible to loops.
- Some form of *distance* is used to calculate a route's metric.
- The **Bellman-Ford algorithm** is used to determine the shortest path.

A distance-vector routing protocol begins by advertising directly-connected networks to its neighbors. These updates are sent *regularly* (RIP – every 30 seconds; IGRP – every 90 seconds).

Neighbors will add the routes from these updates to their own routing tables. Each neighbor trusts this information *completely*, and will forward their full routing table (connected *and* learned routes) to every other neighbor. Thus, routers fully (and blindly) rely on neighbors for route information, a concept known as **routing by rumor**.

There are several disadvantages to this behavior. Because routing information is propagated from neighbor to neighbor via periodic updates, distance-vector protocols suffer from slow convergence. This, in addition to blind faith of neighbor updates, results in distance-vector protocols being highly susceptible to routing loops.

Distance-vector protocols utilize some form of **distance** to calculate a route's metric. RIP uses **hopcount** as its distance metric, and IGRP uses a composite of **bandwidth** and **delay**.

* * *

Link-State Routing Protocols

Link-state routing protocols were developed to alleviate the convergence and loop issues of distance-vector protocols. Link-state protocols maintain three separate tables:

- **Neighbor table** – contains a list of all neighbors, and the interface each neighbor is connected off of. Neighbors are formed by sending **Hello** packets.
- **Topology table** – otherwise known as the "link-state" table, contains a map of all links within an **area**, including each link's status.
- **Shortest-Path table** – contains the *best* routes to each particular destination (otherwise known as the "routing" table")

Link-state protocols do *not* "route by rumor." Instead, routers send updates advertising the *state* of their *links* (a **link** is a directly-connected network). All routers know the state of all existing links within their **area**, and store this information in a **topology** table. All routers within an area have *identical* topology tables.

The best route to each link (network) is stored in the **routing** (or **shortest-path**) table. If the state of a link changes, such as a router interface failing, an advertisement containing ***only* this link-state change** will be sent to all routers within that area. Each router will adjust its topology table accordingly, and will calculate a new *best* route if required.

By maintaining a consistent topology table among all routers within an area, link-state protocols can **converge very quickly** and are **immune to routing loops.**

Additionally, because updates are sent only during a link-state change, and contain *only* the change (and not the full table), link-state protocols are **less bandwidth intensive** than distance-vector protocols. However, the three link-state tables **utilize more RAM and CPU** on the router itself.

Link-state protocols utilize some form of **cost**, usually based on bandwidth**,** to calculate a route's metric. The **Dijkstra formula** is used to determine the shortest path.

* * *

Section 15
- Classful vs. Classless Routing -

Classful vs Classless routing protocols

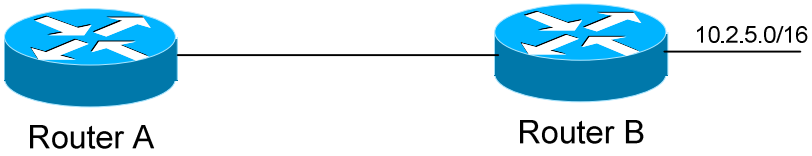
Classful routing protocols do not send subnet mask information with their routing updates. A router running a classful routing protocol will react in one of two ways when receiving a route:

- If the router has a directly connected interface belonging to the same **major network**, it will apply the same subnet mask as that interface.
- If the router *does not* have any interfaces belonging to the same major network, it will apply the *classful* subnet mask to the route.

Belonging to same "major network" simply indicates that they belong to the same "classful" network. For example:

- 10.3.1.0 and 10.5.5.0 belong to the same major network (10.0.0.0)
- 10.1.4.5 and 11.1.4.4 *do not* belong to the same major network
- 192.168.1.1 and 192.168.1.254 belong to the same major network (192.168.1.0)
- 192.168.1.5 and 192.167.2.5 *do not* belong to the same major network.

Take the following example (assume the routing protocol is classful):



If Router B sends a routing update to Router A, it will *not* include the subnet mask for the 10.2.0.0 network. Thus, Router A must make a decision.

If Router A has a directly connected interface that belongs to the same major network (10.0.0.0), it will use the subnet mask of that interface for the route. For example, if Router A has an interface on the 10.4.0.0/16 network, it will apply a subnet mask of /16 to the 10.2.0.0 network.

If Router A *does not* have a directly connected interfacing belonging to the same major network, it will apply the classful subnet mask of /8. This can obviously cause routing difficulties.

When using classful routing protocols, the subnet mask must remain consistent throughout your entire network.

* * *

Classful vs Classless routing protocols (continued)

Classless routing protocols *do* send the subnet mask with their updates. Thus, Variable Length Subnet Masks (VLSMs) are allowed when using classless routing protocols.

Examples of *classful* routing protocols include RIPv1 and IGRP.

Examples of *classless* routing protocols include RIPv2, EIGRP, OSPF, and IS-IS.

The IP Classless Command

The preceding section described how classful and classless protocols differ when sending *routing updates.* Additionally, the router itself can operate either "classfully" or "classlessly" when actually *routing data*.

When a "classful" router has an interface connected to a major network, *it believes it knows all routes connected to that major network.*

For example, a router may have an interface attached to the 10.1.5.0/24 network. It may also have routes from a routing protocol, also for the 10.x.x.x network.

However, if the classful router receives a packet destined for a 10.x.x.x subnet that is *not* in the routing table, it will *drop* that packet, ***even if there is a default route***.

Again, a classful router believes it knows all possible destinations in a major network.

To configure your router in "classful" mode:

Router(config)# *no ip classless*

To configure your router in "classless" mode (this is default in IOS 12.0 and greater):

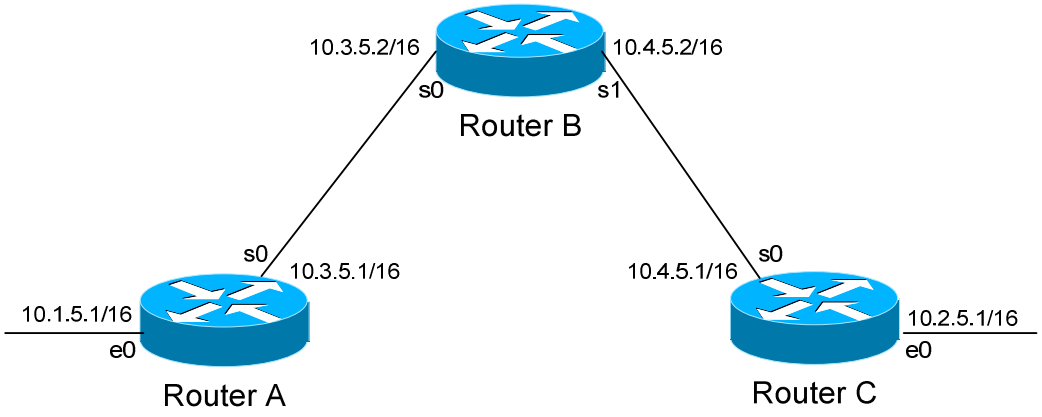
Router(config)# *ip classless*

(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094823.shtml)

* * *

Limitations of Classful Routing Example

The following section will illustrate the limitations of classful routing, using RIPv1 as an example. Consider the following diagram:



This particular scenario will work when using RIPv1, despite the fact that we've subnetted the major 10.0.0.0 network. Notice that the subnets are contiguous (that is, they belong to the same major network), and use the same subnet mask.

When Router A sends a RIPv1 update to Router B via Serial0, it will not include the subnet mask for the 10.1.0.0 network. However, because the 10.3.0.0 network is in the same major network as the 10.1.0.0 network, it will **not summarize** the address. The route entry in the update will simply state "10.1.0.0".

Router B will accept this routing update, and realize that the interface receiving the update (Serial0) belongs to the same major network as the route entry of 10.1.0.0. It will then apply the subnet mask of its Serial0 interface to this route entry.

Router C will similarly send an entry for the 10.2.0.0 network to Router B. Router B's routing table will thus look like:

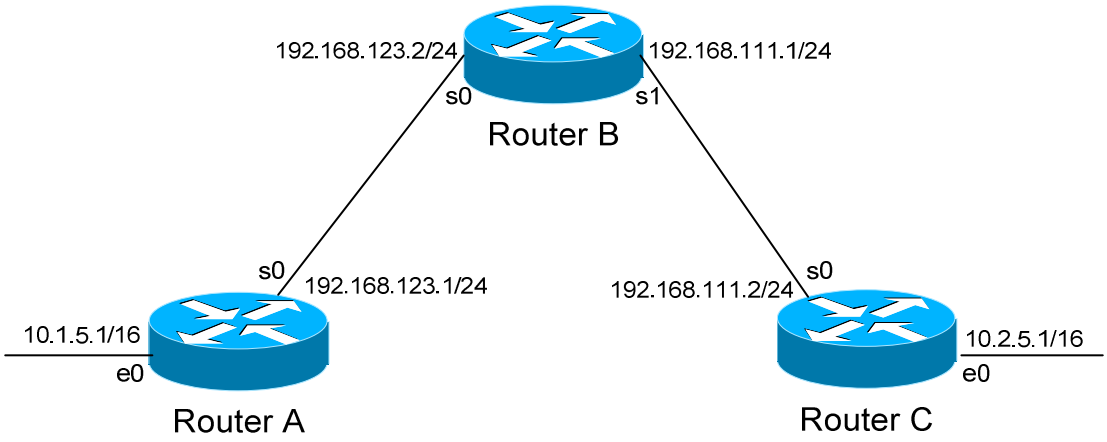
RouterB# *show ip route*

```
Gateway of last resort is not set

       10.0.0.0/16 is subnetted, 4 subnets
C      10.3.0.0 is directly connected, Serial0
C      10.4.0.0 is directly connected, Serial1
R      10.1.0.0 [120/1] via 10.3.5.1, 00:00:00, Serial0
R      10.2.0.0 [120/1] via 10.4.5.1, 00:00:00, Serial1
```

* * *

Limitations of Classful Routing Example

Consider the following, slightly altered, example:



We'll assume that RIPv1 is configured correctly on all routers. Notice that our networks are no longer contiguous. Both Router A and Router C contain *subnets* of the 10.0.0.0 major network (10.1.0.0 and 10.2.0.0 respectively).

Separating these networks now are two Class C subnets (192.168.123.0 and 192.168.111.0).

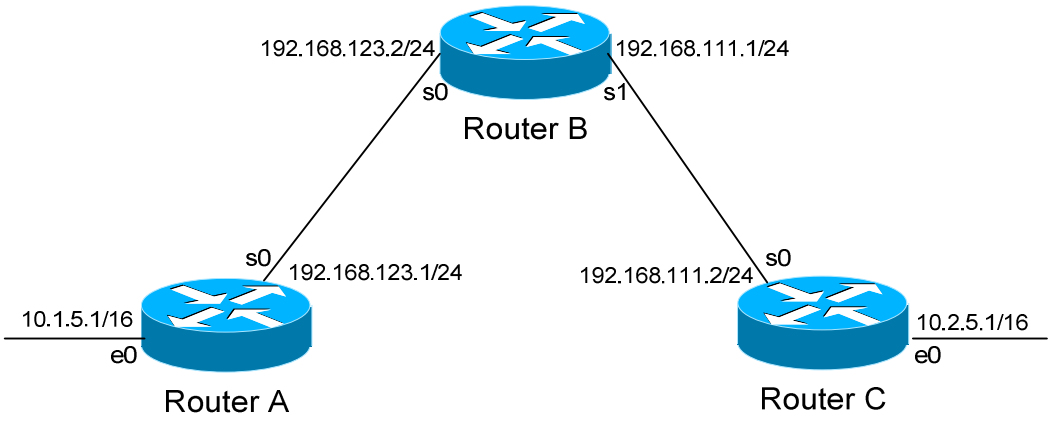
Why is this a problem? Again, when Router A sends a RIPv1 update to Router B via Serial, it will not include the subnet mask for the 10.1.0.0 network. Instead, Router A will consider itself a **border** router, as the 10.1.0.0 and 192.168.123.0 networks *do not* belong to the same major network. Router A will **summarize** the 10.1.0.0/16 network to its classful boundary of 10.0.0.0/8.

Router B will accept this routing update, and realize that it does not have a directly connected interface in the 10.x.x.x scheme. Thus, it has no subnet mask to apply to this route. Because of this, Router B will install the summarized 10.0.0.0 route into its routing table.

Router C, similarly, will consider itself a border router between networks 10.2.0.0 and 192.168.111.0. Thus, Router C will *also* send a summarized 10.0.0.0 route to Router B.

* * *

Limitations of Classful Routing Example



Router B's routing table will then look like:

RouterB# *show ip route*

```
Gateway of last resort is not set

C     192.168.123.0 is directly connected, Serial0
C     192.168.111.0 is directly connected, Serial1
R     10.0.0.0 [120/1] via 192.168.123.1, 00:00:00, Serial0
               [120/1] via 192.168.111.2, 00:00:00, Serial1
```

That's right, Router B now has two *equal* metric routes to get to the summarized 10.0.0.0 network, one through Router A and the other through Router C. Router B will now *load balance* all traffic to *any* 10.x.x.x network between routers A and C. Suffice to say, this is not a good thing. ☺

It gets better. Router B then tries to send routing updates to Router A and Router C, including the summary route of 10.0.0.0/8. Router A's routing table looks like:

RouterA# *show ip route*

```
Gateway of last resort is not set

C     192.168.123.0 is directly connected, Serial0
       10.0.0.0/16 is subnetted, 1 subnet
C     10.1.0.0 is directly connected, Ethernet0
```

Router A will receive the summarized 10.0.0.0/8 route from Router B, and will reject it. This is because it already has the summary network of 10.0.0.0 in its routing table, and it's directly connected. Router C will respond exactly the same, and the 10.1.0.0/16 and 10.2.0.0/16 networks will never be able to communicate.

* * *

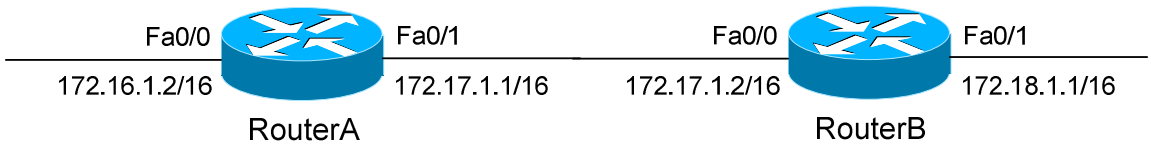
Section 16
- Configuring Static Routes -

Configuring Static Routes

The basic syntax for a static route is as follows:

Router(config)# *ip route [destination_network] [subnet_mask] [next-hop]*

Consider the following example:



RouterA will have the 172.16.0.0/16 and 172.17.0.0/16 networks in its routing table as directly-connected routes. To add a static route on RouterA, pointing to the *172.18.0.0/16* network off of RouterB:

RouterA(config)# *ip route 172.18.0.0 255.255.0.0 172.17.1.2*

Notice that we point to the IP address on RouterB's fa0/0 interface as the *next-hop* address. Likewise, to add a static route on RouterB, pointing to the *172.16.0.0/16* network off of RouterA:

RouterB(config)# *ip route 172.16.0.0 255.255.0.0 172.17.1.1*

To remove a static route, simply type *no* in front of it:

RouterA(config)# *no ip route 172.18.0.0 255.255.0.0 172.17.1.2*

On point-to-point links, an **exit-interface** can be specified instead of a next-hop address. Still using the previous diagram as an example:

RouterA(config)# *ip route 172.18.0.0 255.255.0.0 fa0/1*

RouterB(config)# *ip route 172.16.0.0 255.255.0.0 fa0/0*

A static route using an exit-interface has an Administrative Distance of **0,** as opposed to the default AD of **1** for static routes. An exit-interface is only functional on a point-to-point link, as there is only one possible next-hop device.

* * *

Advanced Static Routes Parameters

The Administrative Distance of a static route can be changed to form a **floating static route**, which will only be used if there are no other routes with a lesser AD in the routing table. A floating static route is often used as a *backup* route to a dynamic routing protocol.

To change the Administrative Distance of a static route to *250*:

RouterA(config)# *ip route 172.18.0.0 255.255.0.0 172.17.1.2 250*

Static routes will only remain in the routing table as long as the interface connecting to the next-hop router is up. To ensure a static route remains *permant*ly in the routing table, even if the next-hop interface is down:

RouterA(config)# *ip route 172.18.0.0 255.255.0.0 172.17.1.2 permanent*

Static routes can additionally be used to *discard* traffic to specific networks, by directing that traffic to a virtual *null* interface:

RouterA(config)# *ip route 10.0.0.0 255.0.0.0 null0*

Default Routes

Normally, if a specific route to a particular network does not exist, a router will drop all traffic destined to that network.

A **default route,** or **gateway of last resort**, allows traffic to be forwarded, even without a specific route to a particular network.

The default route is identified by all zeros in both the network and subnet mask (*0.0.0.0 0.0.0.0).* It is the *least* specific route possible, and thus will *only* be used if a more specific route does not exist (hence "gateway of last resort").

To configure a default route:

RouterA(config)# *ip route 0.0.0.0 0.0.0.0 172.17.1.2*

Advanced default routing is covered in great detail in another guide.

* * *

Section 17
- Routing Information Protocol -

RIP (Routing Information Protocol)

RIP is a standardized Distance Vector protocol, designed for use on smaller networks. RIP was one of the first true Distance Vector routing protocols, and is supported on a wide variety of systems.

RIP adheres to the following Distance Vector characteristics:

- RIP sends out periodic routing updates (every **30 seconds**)
- RIP sends out the full routing table every periodic update
- RIP uses a form of distance as its metric (in this case, **hopcount**)
- RIP uses the Bellman-Ford Distance Vector algorithm to determine the best "path" to a particular destination

Other characteristics of RIP include:

- RIP supports IP and IPX routing.
- RIP utilizes UDP port 520
- RIP routes have an administrative distance of **120**.
- RIP has a maximum hopcount of **15 hops.**

Any network that is 16 hops away or more is considered unreachable to RIP, thus the maximum diameter of the network is 15 hops. A metric of 16 hops in RIP is considered a **poison route** or **infinity metric.**

If multiple paths exist to a particular destination, RIP will load balance between those paths (by default, up to **4**) only if the metric (hopcount) is **equal**. RIP uses a round-robin system of load-balancing between equal metric routes, which can lead to **pinhole congestion**.

For example, two paths might exist to a particular destination, one going through a 9600 baud link, the other via a T1. If the metric (hopcount) is equal, RIP will load-balance, sending an equal amount of traffic down the 9600 baud link and the T1. This will (obviously) cause the slower link to become congested.

RIP Versions

RIP has two versions, **Version 1** (**RIPv1**) and **Version 2** (**RIPv2**).

RIPv1 (RFC 1058) is **classful**, and thus does not include the subnet mask with its routing table updates. Because of this, RIPv1 does not support **Variable Length Subnet Masks (VLSMs**). When using RIPv1, networks must be contiguous, and subnets of a major network must be configured with identical subnet masks. Otherwise, route table inconsistencies (or worse) will occur.

RIPv1 sends updates as **broadcasts** to address 255.255.255.255.

RIPv2 (RFC 2543) is **classless**, and thus *does* include the subnet mask with its routing table updates. RIPv2 fully supports VLSMs, allowing discontiguous networks and varying subnet masks to exist.

Other enhancements offered by RIPv2 include:

- Routing updates are sent via **multicast**, using address **224.0.0.9**
- Encrypted authentication can be configured between RIPv2 routers
- Route tagging is supported (explained in a later section)

RIPv2 can interoperate with RIPv1. By default:

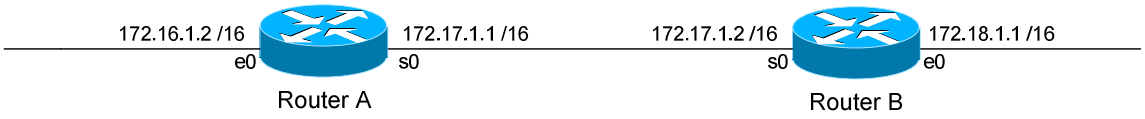
- RIPv1 routers will sent only Version 1 packets
- RIPv1 routers will receive both Version 1 and 2 updates
- RIPv2 routers will both send and receive only Version 2 updates

We can control the version of RIP a particular interface will "send" or "receive."

Unless RIPv2 is manually specified, a Cisco will default to RIPv1 when configuring RIP.

* * *

RIPv1 Basic Configuration



Routing protocol configuration occurs in Global Configuration mode. On Router A, to configure RIP, we would type:

Router(config)# *router rip*
Router(config-router)# *network 172.16.0.0*
Router(config-router)# *network 172.17.0.0*

The first command, *router rip*, enables the RIP process.

The *network* statements tell RIP which networks you wish to advertise to other RIP routers. We simply list the networks that are directly connected to our router. Notice that we specify the networks at their classful boundaries, and we do not specify a subnet mask.

To configure Router B:

Router(config)# *router rip*
Router(config-router)# *network 172.17.0.0*
Router(config-router)# *network 172.18.0.0*

The routing table on Router A will look like:

RouterA# *show ip route*

```
<eliminated irrelevant header>

Gateway of last resort is not set

C     172.16.0.0 is directly connected, Ethernet0
C     172.17.0.0 is directly connected, Serial0
R     172.18.0.0 [120/1] via 172.17.1.2, 00:00:00, Serial0
```

The routing table on Router B will look like:

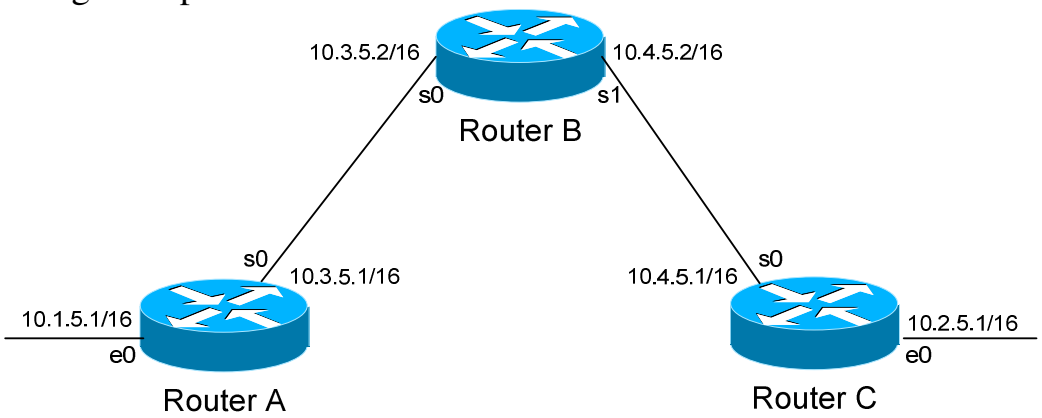
RouterB# *show ip route*

```
<eliminated irrelevant header>

Gateway of last resort is not set

C     172.17.0.0 is directly connected, Serial0
C     172.18.0.0 is directly connected, Ethernet0
R     172.16.0.0 [120/1] via 172.17.1.1, 00:00:00, Serial0
```

* * *

Limitations of RIPv1

The example on the previous page works fine with RIPv1, because the networks are contiguous and the subnet masks are consistent. Consider the following example:



This particular scenario will *still* work when using RIPv1, despite the fact that we've subnetted the major 10.0.0.0 network. Notice that the subnets are contiguous (that is, they belong to the same major network), and use the same subnet mask.

When Router A sends a RIPv1 update to Router B via Serial0, it will not include the subnet mask for the 10.1.0.0 network. However, because the 10.3.0.0 network is in the same major network as the 10.1.0.0 network, it will **not summarize** the address. The route entry in the update will simply state "10.1.0.0".

Router B will accept this routing update, and realize that the interface receiving the update (Serial0) belongs to the same major network as the route entry of 10.1.0.0. It will then apply the subnet mask of its Serial0 interface to this route entry.

Router C will similarly send an entry for the 10.2.0.0 network to Router B. Router B's routing table will thus look like:

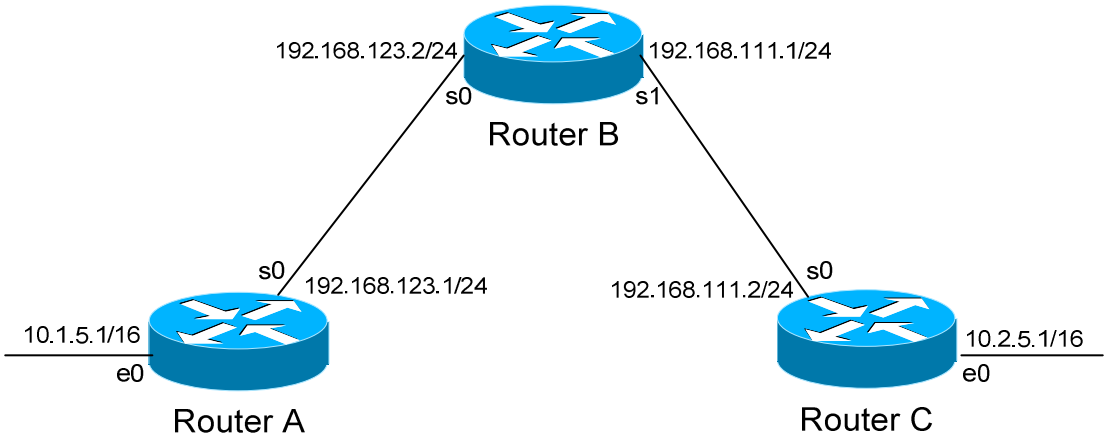
RouterB# *show ip route*

```
Gateway of last resort is not set

       10.0.0.0/16 is subnetted, 4 subnets
C      10.3.0.0 is directly connected, Serial0
C      10.4.0.0 is directly connected, Serial1
R      10.1.0.0 [120/1] via 10.3.5.1, 00:00:00, Serial0
R      10.2.0.0 [120/1] via 10.4.5.1, 00:00:00, Serial1
```

* * *

Limitations of RIPv1 (continued)

Consider the following, slightly altered, example:



We'll assume that RIPv1 is configured correctly on all routers. Notice that our networks are no longer contiguous. Both Router A and Router C contain *subnets* of the 10.0.0.0 major network (10.1.0.0 and 10.2.0.0 respectively).

Separating these networks now are two Class C subnets (192.168.123.0 and 192.168.111.0).

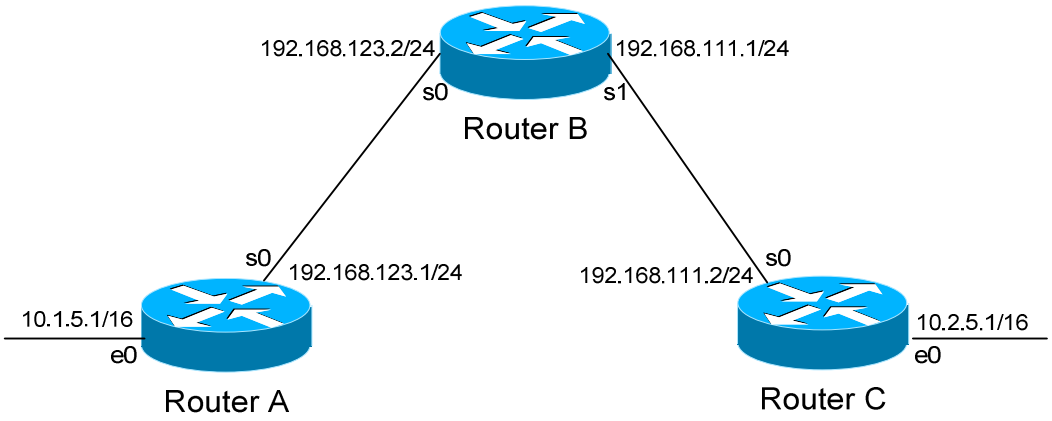
Why is this a problem? Again, when Router A sends a RIPv1 update to Router B via Serial, it will not include the subnet mask for the 10.1.0.0 network. Instead, Router A will consider itself a **border** router, as the 10.1.0.0 and 192.168.123.0 networks *do not* belong to the same major network. Router A will **summarize** the 10.1.0.0/16 network to its classful boundary of 10.0.0.0/8.

Router B will accept this routing update, and realize that it does not have a directly connected interface in the 10.x.x.x scheme. Thus, it has no subnet mask to apply to this route. Because of this, Router B will install the summarized 10.0.0.0 route into its routing table.

Router C, similarly, will consider itself a border router between networks 10.2.0.0 and 192.168.111.0. Thus, Router C will *also* send a summarized 10.0.0.0 route to Router B.

* * *

Limitations of RIPv1 (continued)



Router B's routing table will then look like:

RouterB# *show ip route*

```
Gateway of last resort is not set

C     192.168.123.0 is directly connected, Serial0
C     192.168.111.0 is directly connected, Serial1
R     10.0.0.0 [120/1] via 192.168.123.1, 00:00:00, Serial0
               [120/1] via 192.168.111.2, 00:00:00, Serial1
```

That's right, Router B now has two *equal* metric routes to get to the summarized 10.0.0.0 network, one through Router A and the other through Router C. Router B will now *load balance* all traffic to *any* 10.x.x.x network between routers A and C. Suffice to say, this is not a good thing. ☺

It gets better. Router B then tries to send routing updates to Router A and Router C, including the summary route of 10.0.0.0/8. Router A's routing table looks like:

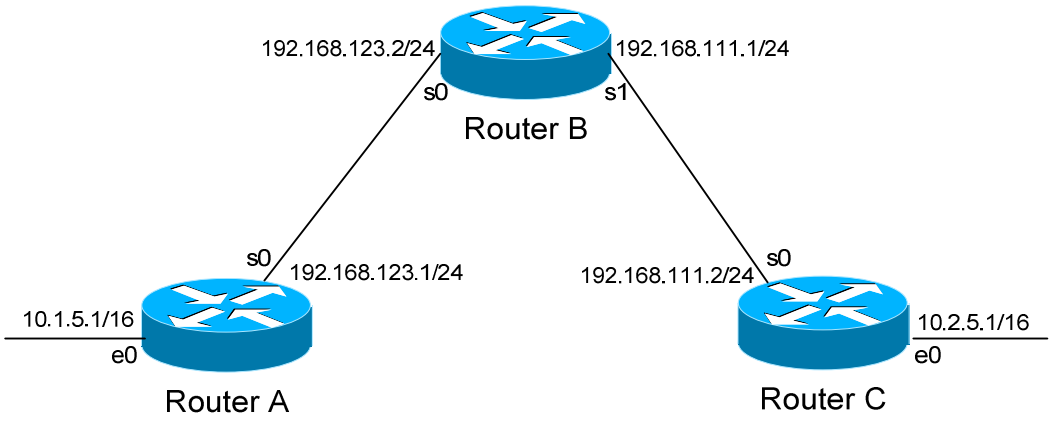
RouterA# *show ip route*

```
Gateway of last resort is not set

C     192.168.123.0 is directly connected, Serial0
       10.0.0.0/16 is subnetted, 1 subnet
C     10.1.0.0 is directly connected, Ethernet0
```

Router A will receive the summarized 10.0.0.0/8 route from Router B, and will reject it. This is because it already has the summary network of 10.0.0.0 in its routing table, and it's directly connected. Router C will respond exactly the same, and the 10.1.0.0/16 and 10.2.0.0/16 networks will *never* be able to communicate.

* * *

RIPv2 Configuration



RIPv2 overcomes the limitations of RIPv1 by including the subnet mask in its routing updates. By default, Cisco routers will use RIPv1. To change to Version 2, you must type:

Router(config)# *router rip*
Router(config-router)# *version 2*

Thus, the configuration of Router A would be:

RouterA(config)# *router rip*
RouterA(config-router)# *version 2*
RouterA(config-router)# *network 10.0.0.0*
RouterA(config-router)# *network 192.168.123.0*

Despite the fact that RIPv2 is a classless routing protocol, we still specify networks at their classful boundaries, without a subnet mask.

However, when Router A sends a **RIPv2** update to Router B via Serial0, by default it will still **summarize** the 10.1.0.0/16 network to 10.0.0.0/8. Again, this is because the 10.1.0.0 and 192.168.123.0 networks *do not* belong to the same major network. Thus, RIPv2 acts like RIPv1 in this circumstance…

…unless you disable **auto summarization:**

RouterA(config)# *router rip*
RouterA(config-router)# *version 2*
RouterA(config-router)# *no auto-summary*

The *no auto-summary* command will prevent Router A from summarizing the 10.1.0.0 network. Instead, Router A will send an update that includes both the subnetted network (10.1.0.0) and its subnet mask (255.255.0.0).

* * *

RIP Timers

RIP has four basic timers:

Update Timer (default **30 seconds**) – indicates how often the router will send out a routing table update.

Invalid Timer (default **180 seconds**) – indicates how long a route will remain in a routing table before being marked as invalid, if no new updates are heard about this route. The invalid timer will be reset if an update is received for that particular route *before* the timer expires.

A route marked as invalid is *not* immediately removed from the routing table. Instead, the route is marked (and advertised) with a metric of 16, indicating it is unreachable, and placed in a **hold-down** state.

Hold-down Timer (default **180 seconds**) – indicates how long RIP will "suppress" a route that it has placed in a **hold-down** state. RIP will not accept any new updates for routes in a hold-down state, until the hold-down timer expires.

A route will enter a hold-down state for one of three reasons:

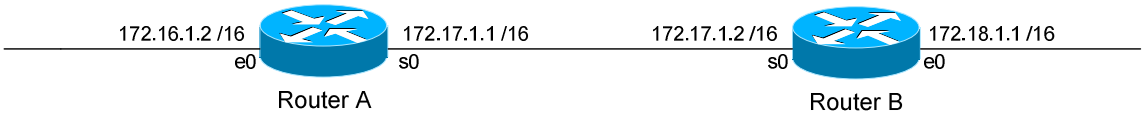
- The invalid timer has expired.
- An update has been received from another router, marking that route with a metric of 16 (or *unreachable*).
- An update has been received from another router, marking that route with a *higher* metric than what is currently in the routing table. This is to prevent loops.

Flush Timer (default **240 seconds**) **–** indicates how long a route can remain in a routing table before being flushed, if no new updates are heard about this route. The flush timer runs **concurrently with the invalid timer**, and thus will flush out a route 60 seconds after it has been marked invalid.

RIP timers must be identical on **all** routers on the RIP network, otherwise massive instability will occur.

* * *

RIP Timers Configuration and Example



Consider the above example. Router A receives a RIP update from Router B that includes network 172.18.0.0. Router A adds this network to its routing table:

RouterA# *show ip route*

```
Gateway of last resort is not set

C     172.16.0.0 is directly connected, Ethernet0
C     172.17.0.0 is directly connected, Serial0
R     172.18.0.0 [120/1] via 172.17.1.2, 00:00:00, Serial0
```

Immediately, Router A sets an **invalid** timer of 180 seconds and **flush** timer of 240 seconds to this route, which run concurrently. If no update for this route is heard for 180 seconds, several things will occur:

- The route is marked as invalid in the routing table.
- The route enters a **hold-down** state (triggering the hold-down timer).
- The route is advertised to all other routers as unreachable.

The hold-down timer runs for 180 seconds *after* the route is marked as invalid. The router will not accept any new updates for this route until this hold-down period expires.

If no update is heard *at all*, the route will be removed from the routing table once the flush timer expires, which is 60 seconds after the route is marked as invalid. Remember that the invalid and flush timers run concurrently.

To configure the RIP timers:

Router(config)# *router rip*
Router(config-router)# *timers basic 20 120 120 160*

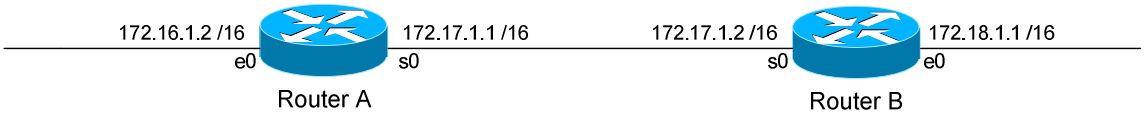
The *timers basic* command allows us to change the update (*20*), invalid (*120*), hold-down (*120)*, and flush *(240)* timers. To return the timers back to their defaults:

Router(config-router)# *no timers basic*

* * *

RIP Loop Avoidance Mechanisms

RIP, as a Distance Vector routing protocol, is susceptible to loops.



Let's assume no loop avoidance mechanisms are configured on either router. If the 172.18.0.0 network fails, Router B will send out an update to Router A within 30 seconds (whenever its update timer expires) stating that route is unreachable (metric = 16).

But what if an update from Router A reaches Router B *before* this can happen? Router A believes it can reach the 172.18.0.0 network in one hop (through Router B). This will cause Router B to believe it can reach the failed 172.18.0.0 network in **two hops**, through Router A. Both routers will continue to increment the metric for the network until they reach a hop count of **16**, which is unreachable. This behavior is known as **counting to infinity**.

How can we prevent this from happening? There are several loop avoidance mechanisms:

Split-Horizon – Prevents a routing update from being sent out the interface it was received on. In our above example, this would prevent Router A from sending an update for the 172.18.0.0 network *back* to Router B, as it originally learned the route from Router B. Split-horizon is **enabled** by default on Cisco Routers.

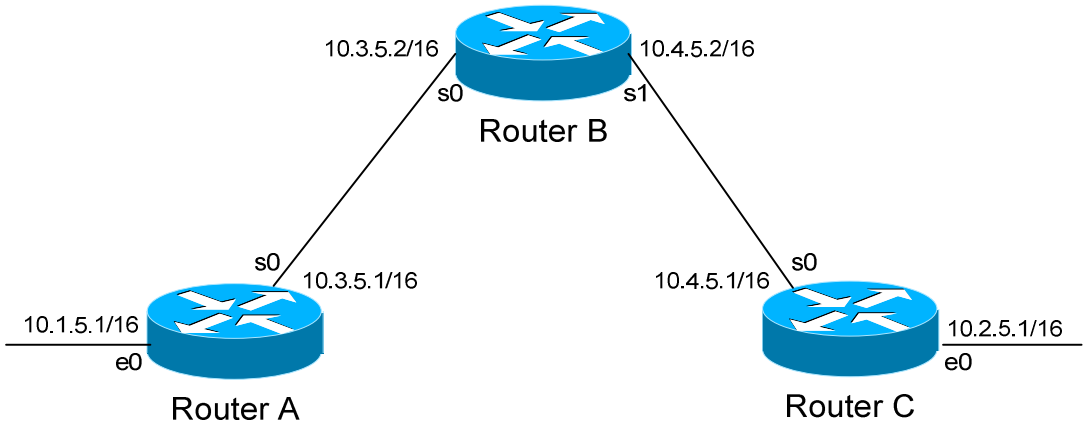
Route-Poisoning – Works in conjunction with split-horizon, by **triggering** an automatic update for the failed network, without waiting for the update timer to expire. This update is sent out all interfaces with an infinity metric for that network.

Hold-Down Timers – Prevents RIP from accepting any new updates for routes in a hold-down state, until the hold-down timer expires. If Router A sends an update to Router B with a *higher* metric than what is currently in Router B's routing table, that route will be placed in a hold-down state. (Router A's metric for the 172.18.0.0 network is 1; while Router B's metric is 0).

* * *

RIP Passive Interfaces

It is possible to control which router interfaces will participate in the RIP process.



Consider the following scenario. Router C does not want to participate in the RIP domain. However, it still wants to *listen* to updates being sent from Router B, just not *send* any updates back to Router B:

RouterC(config)# *router rip*
RouterC(config-router)# *network 10.4.0.0*
RouterC(config-router)# *network 10.2.0.0*
RouterC(config-router)# *passive-interface s0*

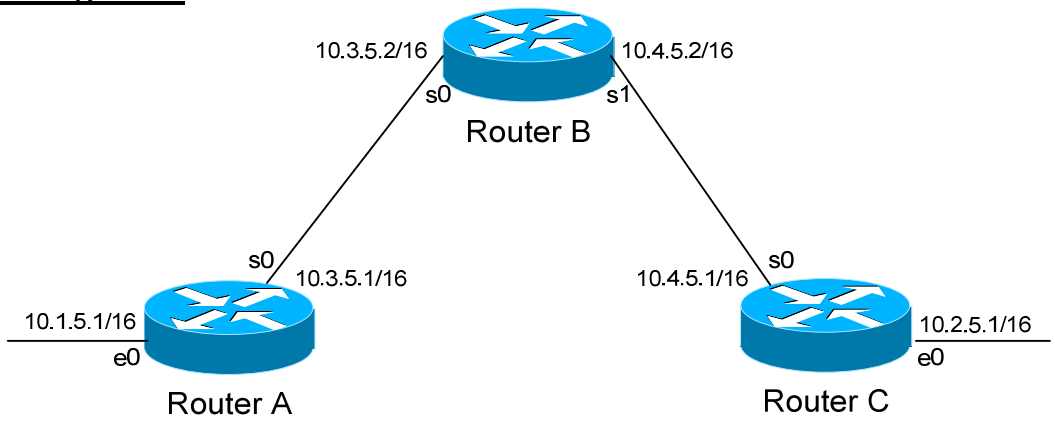
The *passive-interface* command will prevent updates from being **sent** out of the Serial0 interface, but Router C will still **receive** updates on this interface.

We can configure **all** interfaces to be passive using the *passive-interface default* command, and then individually use the *no passive-interface* command on the interfaces we **do** want updates to be sent out:

RouterC(config)# *router rip*
RouterC(config-router)# *network 10.4.0.0*
RouterC(config-router)# *network 10.2.0.0*
RouterC(config-router)# *passive-interface default*
RouterC(config-router)# *no passive-interface e0*

* * *

RIP Neighbors



Recall that RIPv1 sends out its updates as **broadcasts**, whereas RIPv2 sends out its updates as **multicasts** to the 224.0.0.9 address. We can configure specific RIP *neighbor* commands, which will allow us to **unicast** routing updates to those neighbors.

On Router B:

RouterB(config)# *router rip*
RouterB(config-router)# *network 10.3.0.0*
RouterB(config-router)# *network 10.4.0.0*
RouterB(config-router)# *neighbor 10.3.5.1*
RouterB(config-router)# *neighbor 10.4.5.1*

Router B will now unicast RIP updates to Router A and Router C.

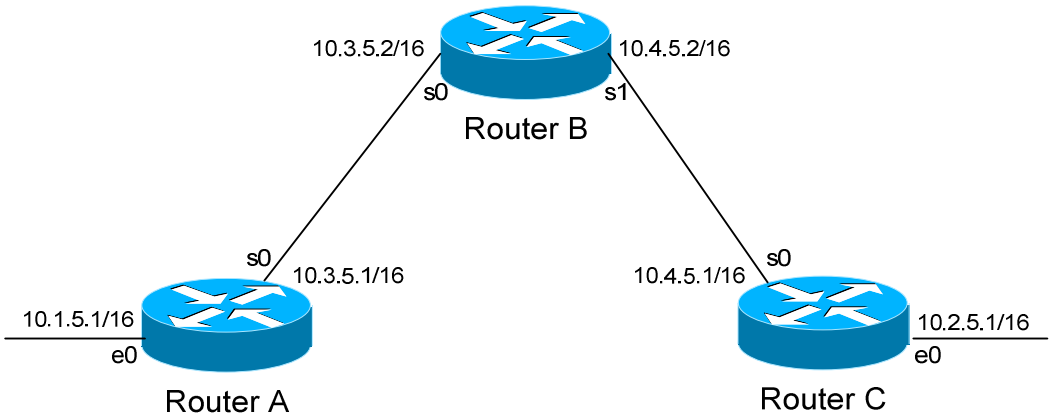
However, Router B will still broadcast (if RIPv1) or multicast (if RIPv2) its updates, in addition to sending unicast updates to its neighbors. In order to prevent broadcast/multicast updates, we must also use **passive interfaces**:

RouterB(config)# *router rip*
RouterB(config-router)# *passive-interface s0*
RouterB(config-router)# *passive-interface s1*
RouterB(config-router)# *neighbor 10.3.5.1*
RouterB(config-router)# *neighbor 10.4.5.1*

The *passive-interface* commands prevent the updates from being broadcasted or multicasted. The *neighbor* commands still allow unicast updates to those specific neighbors.

* * *

Interoperating between RIPv1 and RIPv2



Recall that, with some configuration, RIPv1 and RIPv2 can interoperate. By default:

- RIPv1 routers will sent only Version 1 packets
- RIPv1 routers will receive both Version 1 and 2 updates
- RIPv2 routers will both send and receive only Version 2 updates

If Router A is running RIP v1, and Router B is running RIP v2, some additional configuration is necessary.

Either we must configure Router A to send Version 2 updates:

RouterA(config)# *interface s0*
RouterA(config-if)# *ip rip send version 2*

Or configure Router B to accept Version 1 updates.

RouterB(config)# *interface s0*
RouterB(config-if)# *ip rip receive version 1*

Notice that this is configured on an interface. Essentially, we're configuring the version of RIP on a per-interface basis.

We can also have an interface send or receive both versions simultaneously:

RouterB(config)# *interface s0*
RouterB(config-if)# *ip rip receive version 1 2*

We can further for RIPv2 to send broadcast updates, instead of multicasts:

RouterB(config)# *interface s0*
RouterB(config)# *ip rip v2-broadcast*

* * *

Triggering RIP Updates

On point-to-point interfaces, we can actually force RIP to only send routing updates if there is a change:

RouterB(config)# *interface s0.150 point-to-point*
RouterB(config-if)# *ip rip triggered*

Again, this is only applicable to **point-to-point** links. We cannot configure RIP triggered updates on an Ethernet network.

Troubleshooting RIP

Various troubleshooting commands exist for RIP.

To view the IP routing table:

Router# *show ip route*

```
<eliminated irrelevant header>

Gateway of last resort is not set

C     172.16.0.0 is directly connected, Ethernet0
C     172.17.0.0 is directly connected, Serial0
R     172.18.0.0 [120/1] via 172.17.1.2, 00:00:15, Serial0
R     192.168.123.0 [120/1] via 172.16.1.1, 00:00:00, Ethernet0
```

To view a specific route within the IP routing table:

Router# *show ip route 172.18.0.0*

```
Routing entry for 172.18.0.0/16
      Known via "rip", distance 120, metric 1
      Last update from 172.17.1.2 on Serial 0, 00:00:15 ago
```

To debug RIP in real time:

Router# *debug ip rip*

* * *

Troubleshooting RIP (continued)

To view information specific to the RIP protocol:

Router# *show ip protocols*

```
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 20 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Incoming routes will have 4 added to metric if on list 1
  Redistributing: connected, static, rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Ethernet0             1     1 2
    Serial0               1 2   1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.16.0.0
    172.17.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.17.1.2           120      00:00:17
  Distance: (default is 120)
```

This command provides us with information on RIP timers, on the RIP versions configured on each interface, and the specific networks RIP is advertising.

To view *all* routes in the RIP database, and not just the entries added to the routing table:

Router# *show ip rip database*

```
7.0.0.0/8    auto-summary
7.0.0.0/8
    [5] via 172.16.1.1, 00:00:06, Ethernet0
172.16.0.0/16    directly connected, Ethernet0
172.17.0.0/16    directly connected, Serial0
```

* * *

Section 18
- Interior Gateway Routing Protocol -

IGRP (Interior Gateway Routing Protocol)

IGRP is a Cisco-proprietary Distance-Vector protocol, designed to be more scalable than RIP, its standardized counterpart.

IGRP adheres to the following Distance-Vector characteristics:

- IGRP sends out periodic routing updates (every **90 seconds**).
- IGRP sends out the full routing table every periodic update.
- IGRP uses a form of distance as its metric (in this case, a composite of **bandwidth** and **delay**).
- IGRP uses the Bellman-Ford Distance Vector algorithm to determine the best "path" to a particular destination.

Other characteristics of IGRP include:

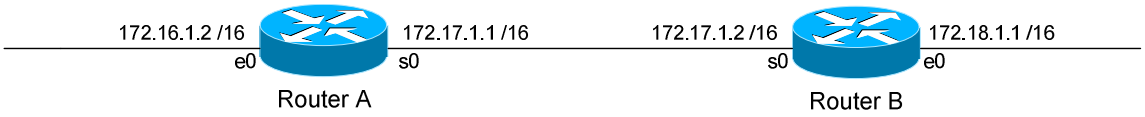
- IGRP supports **only IP routing**.
- IGRP utilizes **IP protocol 9**.
- IGRP routes have an administrative distance of **100**.
- IGRP, by default, supports a maximum of **100 hops.** This value can be adjusted to a maximum of **255 hops**.
- IGRP is a **classful** routing protocol.

IGRP uses **Bandwidth** and **Delay of the Line**, by default, to calculate its distance metric. **Reliability**, **Load**, and **MTU** are optional attributes that can be used to calculate the distance metric.

IGRP requires that you include an **Autonomous System (AS)** number in its configuration. Only routers in the same Autonomous system will send updates between each other.

* * *

Configuring IGRP



Routing protocol configuration occurs in Global Configuration mode. On Router A, to configure IGRP, we would type:

Router(config)# *router igrp 10*
Router(config-router)# *network 172.16.0.0*
Router(config-router)# *network 172.17.0.0*

The first command, *router igrp 10*, enables the IGRP process. The "*10*" indicates the Autonomous System number that we are using. Only other IGRP routers in Autonomous System *10* will share updates with this router.

The *network* statements tell IGRP which networks you wish to advertise to other RIP routers. We simply list the networks that are directly connected to our router. Notice that we specify the networks at their classful boundaries, and we do not specify a subnet mask.

To configure Router B:

Router(config)# *router igrp 10*
Router(config-router)# *network 172.17.0.0*
Router(config-router)# *network 172.18.0.0*

The routing table on Router A will look like:

RouterA# *show ip route*

```
Gateway of last resort is not set

C     172.16.0.0 is directly connected, Ethernet0
C     172.17.0.0 is directly connected, Serial0
I     172.18.0.0 [120/1] via 172.17.1.2, 00:00:00, Serial0
```

The routing table on Router B will look like:

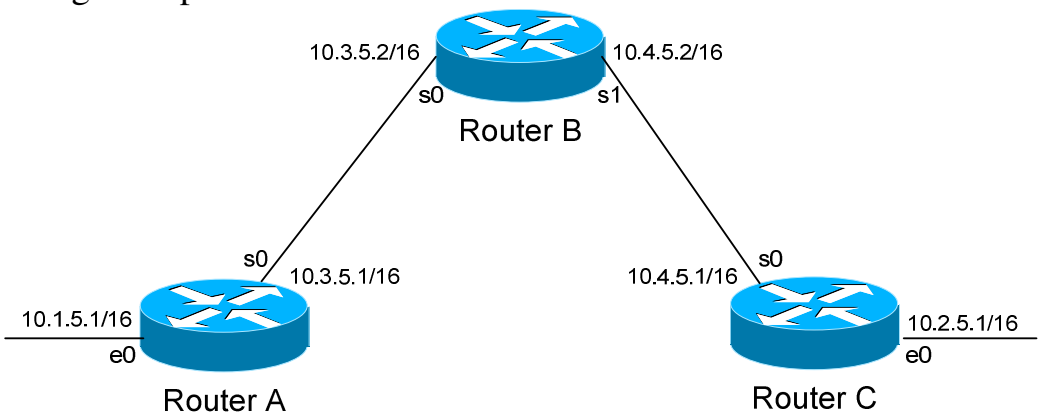
RouterB# *show ip route*

```
Gateway of last resort is not set

C     172.17.0.0 is directly connected, Serial0
C     172.18.0.0 is directly connected, Ethernet0
I     172.16.0.0 [120/1] via 172.17.1.1, 00:00:00, Serial0
```

* * *

Limitations of IGRP

The example on the previous page works fine with IGRP, because the networks are contiguous and the subnet masks are consistent. Consider the following example:



This particular scenario will *still* work when using IGRP, despite the fact that we've subnetted the major 10.0.0.0 network. Notice that the subnets are contiguous (that is, they belong to the same major network), and use the same subnet mask.

When Router A sends an IGRP update to Router B via Serial0, it will not include the subnet mask for the 10.1.0.0 network. However, because the 10.3.0.0 network is in the same major network as the 10.1.0.0 network, it will **not summarize** the address. The route entry in the update will simply state "10.1.0.0".

Router B will accept this routing update, and realize that the interface receiving the update (Serial0) belongs to the same major network as the route entry of 10.1.0.0. It will then apply the subnet mask of its Serial0 interface to this route entry.

Router C will similarly send an entry for the 10.2.0.0 network to Router B. Router B's routing table will thus look like:

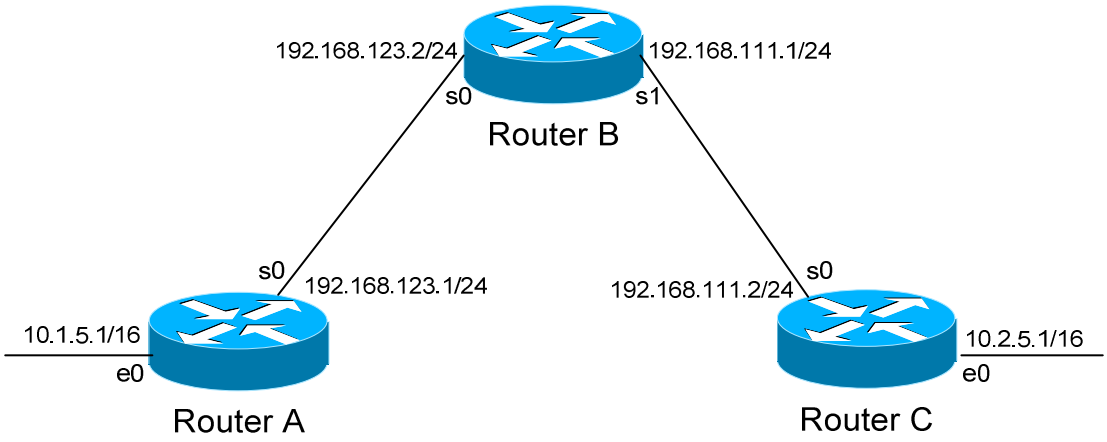
RouterB# *show ip route*

```
Gateway of last resort is not set

       10.0.0.0/16 is subnetted, 4 subnets
C      10.3.0.0 is directly connected, Serial0
C      10.4.0.0 is directly connected, Serial1
I      10.1.0.0 [120/1] via 10.3.5.1, 00:00:00, Serial0
I      10.2.0.0 [120/1] via 10.4.5.1, 00:00:00, Serial1
```

* * *

Limitations of IGRP (continued)

Consider the following, slightly altered, example:



We'll assume that IGRP is configured correctly on all routers. Notice that our networks are no longer contiguous. Both Router A and Router C contain *subnets* of the 10.0.0.0 major network (10.1.0.0 and 10.2.0.0 respectively).

Separating these networks now are two Class C subnets (192.168.123.0 and 192.168.111.0).

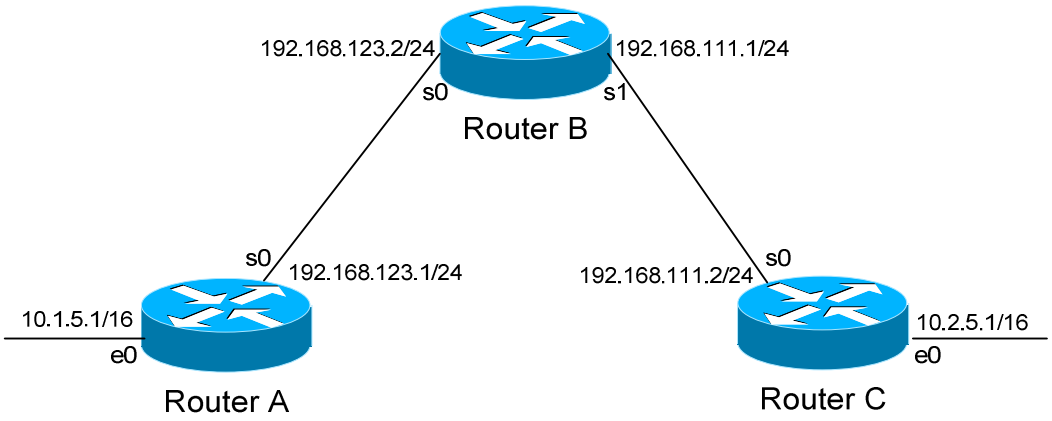
Why is this a problem? Again, when Router A sends an IGRP update to Router B via Serial, it will not include the subnet mask for the 10.1.0.0 network. Instead, Router A will consider itself a **border** router, as the 10.1.0.0 and 192.168.123.0 networks *do not* belong to the same major network. Router A will **summarize** the 10.1.0.0/16 network to its classful boundary of 10.0.0.0/8.

Router B will accept this routing update, and realize that it does not have a directly connected interface in the 10.x.x.x scheme. Thus, it has no subnet mask to apply to this route. Because of this, Router B will install the summarized 10.0.0.0 route into its routing table.

Router C, similarly, will consider itself a border router between networks 10.2.0.0 and 192.168.111.0. Thus, Router C will *also* send a summarized 10.0.0.0 route to Router B.

* * *

Limitations of IGRP (continued)



Router B's routing table will then look like:

RouterB# *show ip route*

```
Gateway of last resort is not set

C     192.168.123.0 is directly connected, Serial0
C     192.168.111.0 is directly connected, Serial1
I     10.0.0.0 [120/1] via 192.168.123.1, 00:00:00, Serial0
               [120/1] via 192.168.111.2, 00:00:00, Serial1
```

That's right, Router B now has two *equal* metric routes to get to the summarized 10.0.0.0 network, one through Router A and the other through Router C. Router B will now *load balance* all traffic to *any* 10.x.x.x network between routers A and C. Suffice to say, this is not a good thing. ☺

It gets better. Router B then tries to send routing updates to Router A and Router C, including the summary route of 10.0.0.0/8. Router A's routing table looks like:

RouterA# *show ip route*

```
Gateway of last resort is not set

C     192.168.123.0 is directly connected, Serial0
       10.0.0.0/16 is subnetted, 1 subnet
C     10.1.0.0 is directly connected, Ethernet0
```

Router A will receive the summarized 10.0.0.0/8 route from Router B, and will reject it. This is because it already has the summary network of 10.0.0.0 in its routing table, and it's directly connected. Router C will respond exactly the same, and the 10.1.0.0/16 and 10.2.0.0/16 networks will *never* be able to communicate.

* * *

IGRP Timers

IGRP has four basic timers:

Update Timer (default **90 seconds**) – indicates how often the router will send out a routing table update.

Invalid Timer (default **270 seconds**) – indicates how long a route will remain in a routing table before being marked as invalid, if no new updates are heard about this route. The invalid timer will be reset if an update is received for that particular route *before* the timer expires.

A route marked as invalid is *not* immediately removed from the routing table. Instead, the route is marked (and advertised) with a metric of 101 (remember, 100 maximum hops is default), indicating it is unreachable, and placed in a **hold-down** state.

Hold-down Timer (default **280 seconds**) – indicates how long IGRP will "suppress" a route that it has placed in a **hold-down** state. IGRP will not accept any new updates for routes in a hold-down state, until the hold-down timer expires.

A route will enter a hold-down state for one of three reasons:

- The invalid timer has expired.
- An update has been received from another router, marking that route with a metric of 101 (unreachable).
- An update has been received from another router, marking that route with a *higher* metric than what is currently in the routing table (this is to prevent loops).

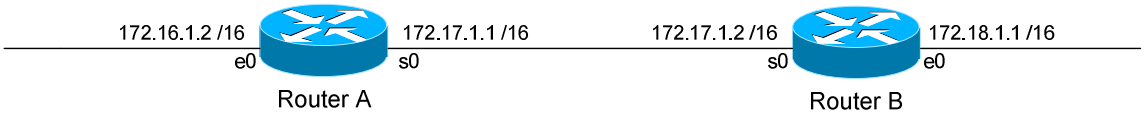
Flush Timer (default **630 seconds**) **–** indicates how long a route can remain in a routing table before being flushed, if no new updates are heard about this route. The flush timer runs **concurrently with the invalid timer**, and thus will flush out a route 360 seconds after it has been marked invalid.

IGRP timers must be identical on **all** routers on the IGRP network, otherwise massive instability will occur.

* * *

IGRP Loop Avoidance Mechanisms

IGRP, as a Distance Vector routing protocol, is susceptible to loops.



Let's assume no loop avoidance mechanisms are configured on either router. If the 172.18.0.0 network fails, Router B will send out an update to Router A within 30 seconds (whenever its update timer expires) stating that route is unreachable.

But what if an update from Router A reaches Router B *before* this can happen? Router A believes it can reach the 172.18.0.0 network in one hop (through Router B). This will cause Router B to believe it can reach the failed 172.18.0.0 network in **two hops**, through Router A. Both routers will continue to increment the metric for the network until they reach an infinity hop count (by default, 101). This behavior is known as **counting to infinity**.

How can we prevent this from happening? There are several loop avoidance mechanisms:

Split-Horizon – Prevents a routing update from being sent out the interface it was received on. In our above example, this would prevent Router A from sending an update for the 172.18.0.0 network *back* to Router B, as it originally learned the route from Router B. Split-horizon is **enabled** by default on Cisco Routers.

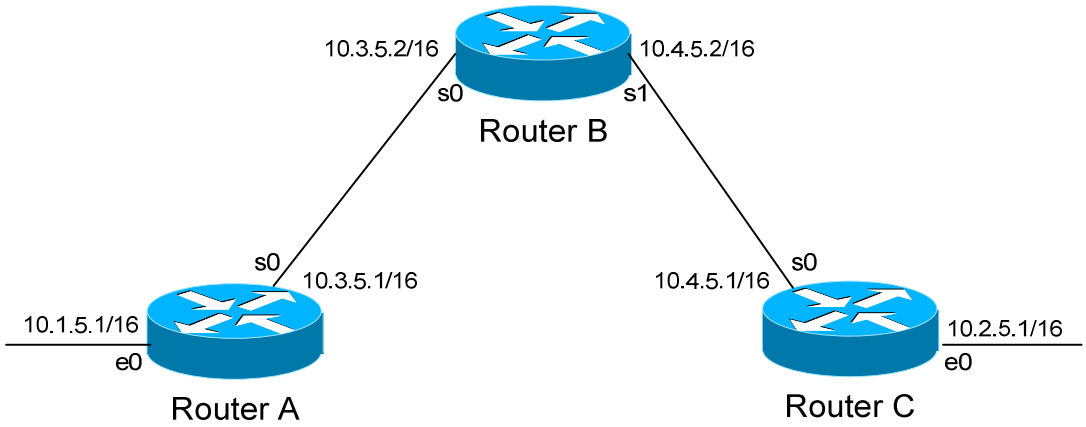
Route-Poisoning – Works in conjunction with split-horizon, by **triggering** an automatic update for the failed network, without waiting for the update timer to expire. This update is sent out all interfaces with an infinity metric for that network.

Hold-Down Timers – Prevents IGRP from accepting any new updates for routes in a hold-down state, until the hold-down timer expires. If Router A sends an update to Router B with a *higher* metric than what is currently in Router B's routing table, that route will be placed in a hold-down state.

* * *

IGRP Passive Interfaces

It is possible to control which router interfaces will participate in the IGRP process.



Consider the following scenario. Router C does not want to participate in the IGRP domain. However, it still wants to *listen* to updates being sent from Router B, just not *send* any updates back to Router B:

RouterC(config)# *router igrp 10*
RouterC(config-router)# *network 10.4.0.0*
RouterC(config-router)# *network 10.2.0.0*
RouterC(config-router)# *passive-interface s0*

The *passive-interface* command will prevent updates from being **sent** out of the Serial0 interface, but Router C will still **receive** updates on this interface.

We can configure **all** interfaces to be passive using the *passive-interface default* command, and then individually use the *no passive-interface* command on the interfaces we **do** want updates to be sent out:

RouterC(config)# *router igrp 10*
RouterC(config-router)# *network 10.4.0.0*
RouterC(config-router)# *network 10.2.0.0*
RouterC(config-router)# *passive-interface default*
RouterC(config-router)# *no passive-interface e0*

* * *

Advanced IGRP Configuration

To change the maximum hop-count to 255 for IGRP:

Router(config)# *router igrp 10*
Router(config-router)# *metric maximum-hops 255*

Section 19
- Enhanced Interior Gateway Routing Protocol -

EIGRP (Enhanced Interior Gateway Routing Protocol)

EIGRP is a Cisco-proprietary Hybrid routing protocol, incorporating features of both Distance-Vector and Link-State routing protocols.

EIGRP adheres to the following Hybrid characteristics:

- EIGRP uses **Diffusing Update Algorithm (DUAL)** to determine the best path among all "feasible" paths. DUAL also helps ensure a loop-free routing environment.
- EIGRP will form **neighbor** relationships with adjacent routers in the same **Autonomous System (AS)**.
- EIGRP traffic is either sent as unicasts, or as multicasts on address **224.0.0.10**, depending on the EIGRP packet type.
- Reliable Transport Protocol (RTP) is used to ensure delivery of most EIGRP packets.
- EIGRP routers **do not** send periodic, full-table routing updates. Updates are sent when a change occurs, and include *only* the change.
- EIGRP is a classless protocol, and thus supports VLSMs.

Other characteristics of EIGRP include:

- EIGRP supports IP, IPX, and Appletalk routing.
- EIGRP applies an Administrative Distance of **90** for routes originating *within* the local Autonomous System.
- EIGRP applies an Administrative Distance of **170** for external routes coming from *outside* the local Autonomous System
- EIGRP uses **Bandwidth** and **Delay of the Line,** by default, to calculate its distance metric. It also supports three other parameters to calculate its metric: **Reliability**, **Load**, and **MTU.**
- EIGRP has a maximum hop-count of **224**, though the default maximum hop-count is set to **100**.

EIGRP, much like OSPF, builds three separate tables:

- **Neighbor table** – list of all neighboring routers. Neighbors must belong to the same **Autonomous System**
- **Topology table** – list of *all* routes in the Autonomous System
- **Routing table** – contains the *best* route for each known network

* * *

EIGRP Neighbors

EIGRP forms neighbor relationships, called **adjacencies,** with other routers in the same AS by exchanging **Hello** packets. Only after an adjacency is formed can routers share routing information. Hello packets are sent as multicasts to address 224.0.0.10.

By default, on LAN and high-speed WAN interfaces, EIGRP Hellos are sent every **5 seconds**. On slower WAN links (T1 speed or slower), EIGRP Hellos are sent every **60 seconds** by default.

The EIGRP Hello timer can be adjusted on a *per interface* basis:

Router(config-if)# *ip hello-interval eigrp 10 7*

The above command allows us to change the *hello* timer to *7* seconds for Autonomous System *10*.

In addition to the Hello timer, EIGRP neighbors are stamped with a **Hold timer.** The Hold timer indicates how long a router should wait before marking a neighbor inactive, if it stops receiving hello packets from that neighbor.

By default, the Hold timer is **three times** the Hello timer. Thus, on high-speed links the timer is set to **15 seconds**, and on slower WAN links the timer is set to **180 seconds**.

The Hold timer can also be adjusted on a *per interface* basis:

Router(config-if)# *ip hold-interval eigrp 10 21*

The above command allows us to change the *hold* timer to *21* seconds for Autonomous System *10*.

Changing the Hello timer **does not** automatically change the Hold timer. Additionally, Hello and Hold timers ***do not*** need to match between routers for an EIGRP neighbor relationship to form.

(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080093f07.shtml#eigrp_work)

* * *

EIGRP Neighbors (continued)

A **neighbor table** is constructed from the EIGRP Hello packets, which includes the following information:

- The IP address of the neighboring router.
- The local interface that received the neighbor's Hello packet.
- The Hold timer.
- A sequence number indicating the order neighbors were learned.

Adjacencies will not form unless the **primary IP addresses** on connecting interfaces are on the same subnet. Neighbors *cannot* be formed on secondary addresses.

If connecting interfaces are on *different* subnets, an EIGRP router will log the following error to console when a multicast Hello is received:

```
00:11:22: IP-EIGRP: Neighbor 172.16.1.1 not on common
subnet for Serial0
```

Always ensure that primary IP addresses belong to the same subnet between EIGRP neighbors.

To log all neighbor messages and errors to console, use the following two commands:

Router(config)# *router eigrp 10*
Router(config-router)# *eigrp log-neighbor-changes*
Router(config-router)# *eigrp log-neighbor-warnings*

(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_configuration_example09186a0080093f09.shtml)

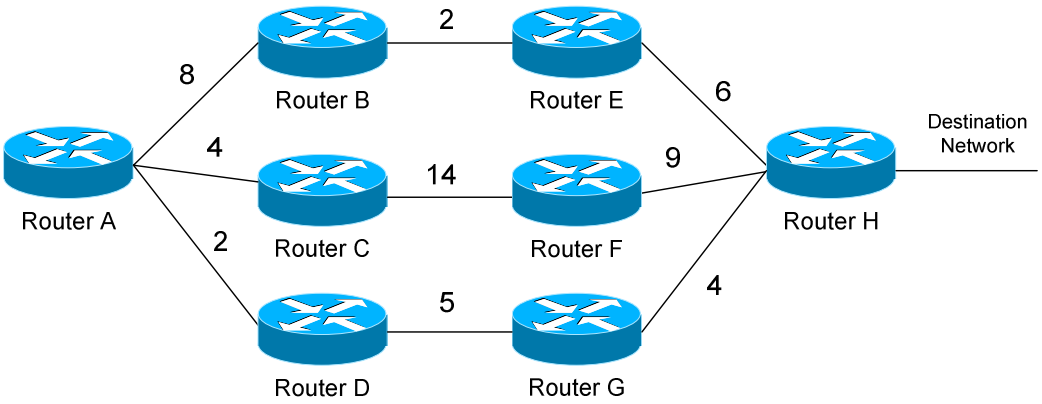
The EIGRP Topology Table

Once EIGRP neighbors form adjacencies, they will begin to share routing information. Each router's update contains a list of all routes known by that router, and the respective metrics for those routes.

All such routes are added to an EIGRP router's topology table. The route with the lowest metric to each network will become the **Feasible Distance (FD)**. The Feasible Distance for each network will be installed into the routing table.

The Feasible Distance is derived from the **Advertised Distance** of the router *sending* the update, and the local router's metric to the advertising router.

Confused? Consider the following example:



Router A has three separate paths to the Destination Network, either through Router B, C, or D. If we add up the metrics to form a "distance" (the metrics are greatly simplified in this example), we can determine the following:

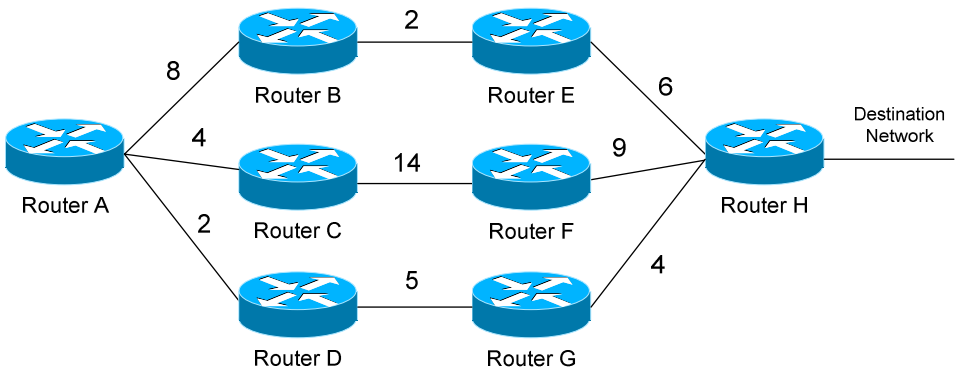
- Router B's Feasible Distance to the Destination Network is 8.
- Router C's Feasible Distance to the Destination Network is 23.
- Router D's Feasible Distance to the Destination Network is 9.

Router B sends an update to Router A, it will provide an **Advertised Distance** of 8 to the Destination Network. Router C will provide an AD of 23, and D will provide an AD of 9.

Router A calculates the total distance to the Destination network by adding the AD of the advertising router, with its own distance to *reach* that advertising router. For example, Router A's metric to Router B is 8; thus, the *total* distance will be 16 to reach the Destination Network through Router B.

* * *

The EIGRP Topology Table (continued)



Remember, however, that Router A's Feasible Distance must be the route with the lowest metric. If we add the Advertised Distance with the local metric between each router, we would see that:

- The route through Router B has a distance of **16** to the destination
- The route through Router C has a distance of **27** to the destination
- The route through Router D has a distance of **11** to the destination

Thus, the route through Router D (metric of 11) would become the **Feasible Distance** for Router A, and is added to the routing table as the *best* route. This route is identified as the **Successor**.

To allow convergence to occur quickly if a link fails, EIGRP includes backup routes in the topology table called **Feasible Successors (FS).** A route will only become a Successor if its Advertised Distance is *less* than the current Feasible Distance**.** This is known as a **Feasible Condition (FC)**.

For example, we determined that Router A's Feasible Distance to the destination is 11, through Router D. Router C's Advertised Distance is 23, and thus *would not* become a Feasible Successor, as it has a higher metric than Router A's current Feasible Distance. Routes that are not Feasible Successors become route **Possibilities.**

Router B's Advertised Distance is 8, which is less than Router A's current Feasible Distance. Thus, the route through Router B to the Destination Network *would* become a Feasible Successor.

Feasible Successors provide EIGRP with redundancy, without forcing routers to re-converge (thus stopping the flow of traffic) when a topology change occurs. If no Feasible Successor exists and a link fails, a route will enter an **Active** (converging) state until an alternate route is found.

* * *

EIGRP Packet Types

EIGRP employs five packet types:

- **Hello packets -** *multicast*
- **Update packets –** *unicast* or *multicast*
- **Query packets** – *multicast*
- **Reply packets** – *unicast*
- **Acknowledgement packets -** *unicast*

Hello packets are used to form neighbor relationships, and were explained in detail previously. Hello packets are always multicast to address 224.0.0.10.

Update packets are sent between neighbors to build the topology and routing tables. Updates sent to *new* neighbors are sent as unicasts. However, if a route's metric is changed, the update is sent out as a multicast to address 224.0.0.10.

Query packets are sent by a router when a Successor route fails, and there are no Feasible Successors in the topology table. The router places the route in an **Active state**, and queries its neighbors for an alternative route. Query packets are sent as a multicast to address 224.0.0.10.

Reply packets are sent in response to Query packets, assuming the responding router has an alternative route (feasible successor). Reply packets are sent as a unicast to the querying router.

Recall that EIGRP utilizes the **Reliable Transport Protocol** (**RTP**) to ensure reliable delivery of most EIGRP packets. Delivery is guaranteed by having packets *acknowledged* using…..**Acknowledgment packets!**

Acknowledgment packets (also known as **ACK's**) are simply Hello packets with no data, other than an acknowledgment number. ACK's are always sent as unicasts. The following packet types employ RTP to ensure reliable delivery via ACK's:

- Update Packets
- Query Packets
- Reply Packets

Hello and Acknowledgments (ha!) packets do not utilize RTP, and thus do not require acknowledgement.

* * *

EIGRP Route States

An EIGRP route can exist in one of two states, in the topology table:

- **Active state**
- **Passive State**

A **Passive state** indicates that a route is reachable, and that EIGRP is fully converged. A stable EIGRP network will have all routes in a Passive state.

A route is placed in an **Active state** when the Successor and any Feasible Successors fail, forcing the EIGRP to send out Query packets and re-converge. Multiple routes in an Active state indicate an unstable EIGRP network. If a Feasible Successor exists, a route should *never* enter an Active state.

Routes will become **Stuck-in-Active (SIA)** when a router sends out a Query packet, but does not receive a Reply packet within **three minutes**. In other words, a route will become SIA if EIGRP fails to re-converge.

To view the current state of routes in the EIGRP topology table:

Router# *show ip eigrp topology*

```
IP-EIGRP Topology Table for AS(10)/ID(172.19.1.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
            r - reply Status, s - sia Status

P 10.3.0.0/16, 1 successors, FD is 2297856
         via 172.16.1.2 (2297856/128256), Serial0
P 172.19.0.0/16, 1 successors, FD is 281600
         via Connected, Serial 1
```

To view only *active* routes in the topology table:

Router# *show ip eigrp topology active*

```
IP-EIGRP Topology Table for AS(10)/ID(172.19.1.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
            r - Reply status

A 172.19.0.0/16, 1 successors, FD is 23456056 1 replies,
            active 0:00:38, query-origin: Multiple Origins
```

(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a008010f016.shtml)

* * *

EIGRP Metrics

EIGRP can utilize 5 separate metrics to determine the best route to a destination:

- **Bandwidth** (K1) – Slowest link in the route path, measured in kilobits
- **Load** (K2) – Cumulative load of all outgoing interfaces in the path, given as a fraction of 255
- **Delay of the Line** (K3) – Cumulative delay of all outgoing interfaces in the path in tens of microseconds
- **Reliability** (K4) – Average reliability of all outgoing interfaces in the path, given as a fraction of 255
- **MTU** (K5) – The smallest Maximum Transmission Unit in the path. The MTU value is actually *never* used to calculate the metric

By default, only **Bandwidth** and **Delay of the Line** are used. This is identical to IGRP, except that EIGRP provides a more granular metric by multiplying the bandwidth and delay by 256. Bandwidth and delay are determined by the interfaces that lead towards the destination network.

By default, the full formula for determining the EIGRP metric is:

[10000000/bandwidth + delay] * 256

The bandwidth value represents the link with the *lowest* bandwidth in the path, in kilobits. The delay is the total delay of all outgoing interfaces in the path.

As indicated above, each metric is symbolized with a "K" and then a number. When configuring EIGRP metrics, we actually identify which metrics we want EIGRP to consider. Again, by default, only Bandwidth and Delay are considered. Thus, using on/off logic:

K1 = 1, K2 = 0, K3 = 1, K4 = 0, K5 = 0

If all metrics were set to "on," the full formula for determining the EIGRP metric would be:

[K1 * bandwidth * 256 + (K2 * bandwidth) / (256 - load)
+ K3 * delay * 256] * [K5 / (reliability + K4)]

Remember, the "K" value is either set to on ("1") or off ("0").

* * *

Configuring EIGRP Metrics

EIGRP allows us to identify which metrics the protocol should consider, using the following commands:

Router(config)# *router eigrp 10*
Router(config-router)# *metric weights 0 1 1 1 0 0*

The first command enables the *EIGRP* process for Autonomous System *10*. The second actually identifies which EIGRP metrics to use. The first number (*0*) is for Type of Service, and should always be zero. The next numbers, in order, are K1 (*1*), K2 (*1*), K3 (*1*), K4 *(0)*, and K5 (*0)*. Thus, we are instructing EIGRP to use bandwidth, load, and delay to calculate the total metric, but not reliability or MTU.

Our formula would thus be:

[K1 * bandwidth * 256 + (K2 * bandwidth) / (256 - load)
+ K3 * delay * 256]

The actual values of our metrics (such as bandwidth, delay, etc.) must be configured indirectly. We can adjust the bandwidth of an interface:

Router(config)# *int s0/0*
Router(config-if)# *bandwidth 64000*
Router(config-if)# *ip bandwidth-percent eigrp 10 30*

However, this command does not actually dictate the *physical* speed of the interface. It merely controls how EIGRP *considers* this interface. Best practice is to set the bandwidth to the actual physical speed of the interface.

The *ip bandwidth-percent eigrp* command limits the percentage of bandwidth EIGRP can use on an interface. The percentage is based on the configured *bandwidth* value. By default, EIGRP will use **up to 50%** of the bandwidth of an interface. The above command adjusts this to *30%* for Autonomous System *10*.

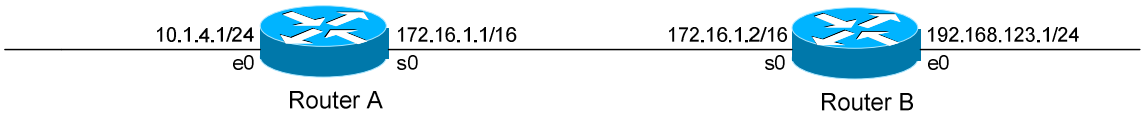
If adjustments to the EIGRP metric need to be made, the delay metric (in tens of microseconds) on an interface should be used:

Router(config)# *int s0/0*
Router(config-if)# *delay 10000*

Metric settings must be **identical** on the connecting interfaces of two routers; otherwise they will not form a neighbor relationship.

* * *

Configuring Basic EIGRP



Routing protocol configuration occurs in Global Configuration mode. On Router A, to configure EIGRP, we would type:

```
RouterA(config)# router eigrp 10
RouterA(config-router)# network 172.16.0.0
RouterA(config-router)# network 10.0.0.0
```

The first command, *router eigrp 10*, enables the EIGRP process. The "*10*" indicates the Autonomous System number that we are using. The Autonomous System number can range from 1 to 65535.

Only other EIGRP routers in Autonomous System *10* will form neighbor adjacencies and share updates with this router.

The *network* statements serve two purposes in EIGRP:

- First, they identify which *networks* you wish to advertise to other EIGRP routers (similar to RIP).
- Second, they identify which *interfaces* on the local router to attempt to form neighbor relationships out of (similar to OSPF).

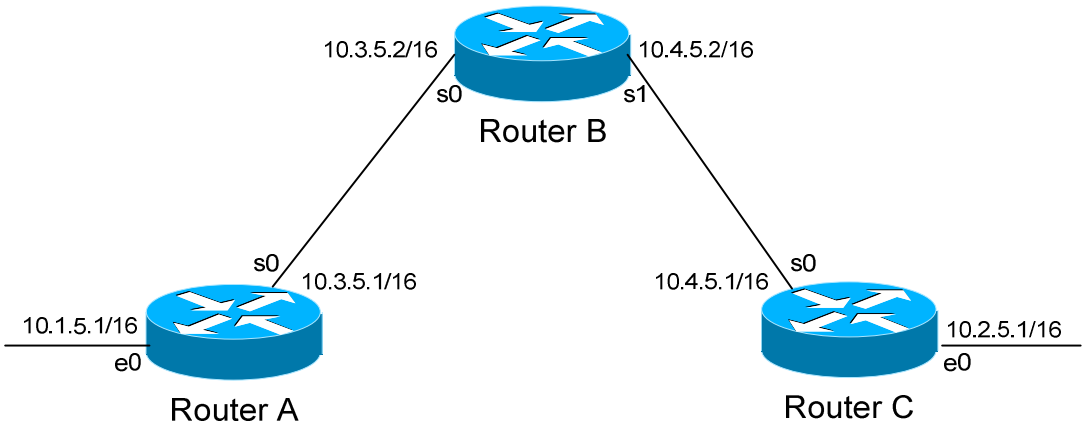
Prior to IOS version 12.0(4), the *network* statements were classful, despite the fact that EIGRP is a classless routing protocol. For example, the above *network 10.0.0.0* command would advertise the networks of directly-connected interfaces belonging to the 10.0.0.0/8 network and its subnets. It would further attempt to form neighbor relationships out of these interfaces.

IOS version **12.0(4) and later** provided us with more granular control of our *network* statements. It introduced a *wildcard mask* parameter, which allows us to choose the networks to advertise in a classless fashion:

```
RouterA(config)# router eigrp 10
RouterA(config-router)# network 172.16.0.0 0.0.255.255
RouterA(config-router)# network 10.1.4.0 0.0.0.255
```

* * *

EIGRP Passive Interfaces



It is possible to control which router interfaces will participate in the EIGRP process. Just as with RIP, we can use the *passive-interface* command.

However, please note that the *passive-interface* command works differently with EIGRP than with RIP or IGRP. EIGRP will no longer form neighbor relationships out of a "passive" interface, thus this command prevents updates from being *sent* or *received* out of this interface:

RouterC(config)# *router eigrp 10*
RouterC(config-router)# *network 10.4.0.0*
RouterC(config-router)# *network 10.2.0.0*
RouterC(config-router)# *passive-interface s0*

Router C will not form a neighbor adjacency with Router B.

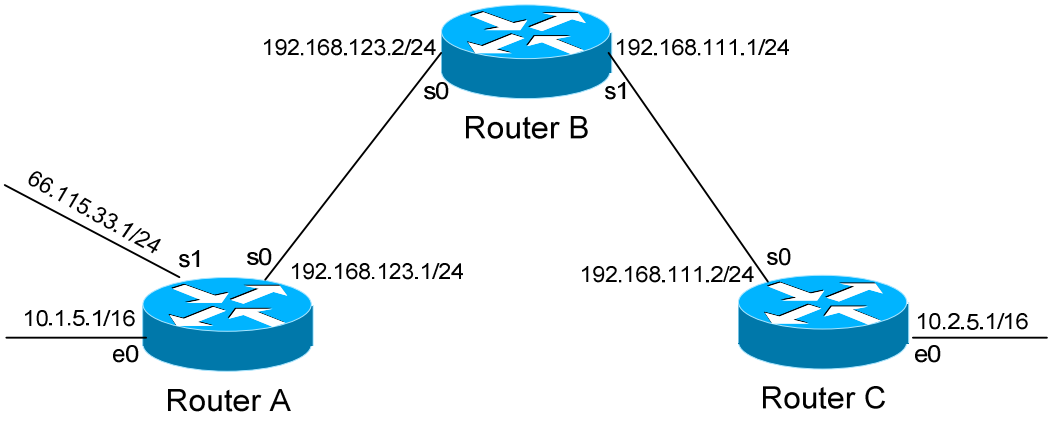
We can configure **all** interfaces to be passive using the *passive-interface default* command, and then individually use the *no passive-interface* command on the interfaces we **do** want neighbors to be formed on:

RouterC(config)# *router eigrp 10*
RouterC(config-router)# *network 10.4.0.0*
RouterC(config-router)# *network 10.2.0.0*
RouterC(config-router)# *passive-interface default*
RouterC(config-router)# *no passive-interface e0*

Always remember, that the *passive-interface* command will prevent EIGRP (and OSPF) from forming neighbor relationships out of that interface. N**o** routing updates are passed in either direction.

* * *

EIGRP Auto-Summarization



EIGRP is a classless routing protocol that supports Variable Length Subnet Masks (VLSMs). The above example would pose no problem for EIGRP.

However, EIGRP will still **automatically summarize** when crossing **major network boundaries**.

For example, when Router A sends an EIGRP update to Router B via Serial0, by default it will still **summarize** the 10.1.0.0/16 network to 10.0.0.0/8. This is because the 10.1.0.0/16 and 192.168.123.0/24 networks *do not* belong to the same major network. Likewise, the 66.115.33.0/24 network will be summarized to 66.0.0.0/8.

An auto-summary route will be advertised as a normal *internal* EIGRP route. The *best* metric from among the summarized routes will be applied to this summary route.

The router that *performed* the auto-summarization will also add the summary route to its routing table, with a next hop of the *Null0* interface. This is to prevent routing loops.

This **auto-summarization** can be disabled:

RouterA(config)# *router eigrp 10*
RouterA(config-router)# *no auto-summary*

The *no auto-summary* command will prevent Router A from summarizing the 10.1.0.0/16 and 66.115.33.0/24 networks.

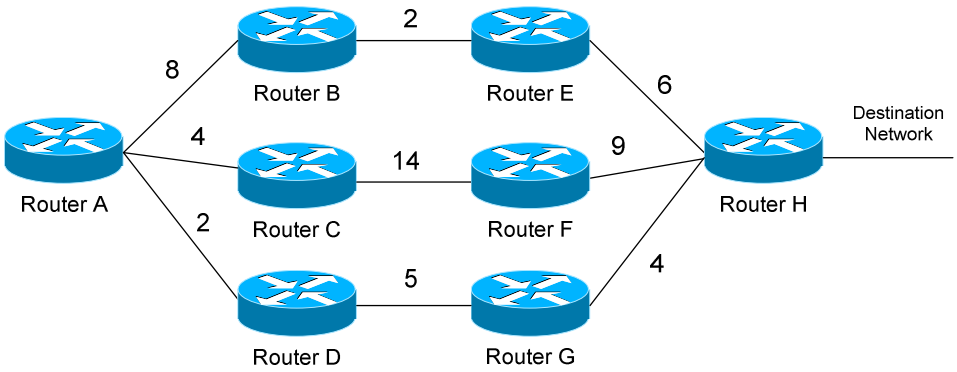
(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_white_paper09186a0080094cb7.shtml#summarization)

* * *

EIGRP Load-Balancing

By default, EIGRP will automatically load-balance across equal-metric routes (four by default, six maximum). EIGRP also supports load-balancing across routes with an *unequal* metric.

Consider the following example:



Earlier in this section, we established that Router A would choose the route through Router D as its **Feasible Distance** to the destination network. The route through Router B became a **Feasible Successor.**

By default, EIGRP will *not* load-balance between these two routes, as their metrics are different (11 through Router D, 16 through Router B). We must use the *variance* command to tell EIGRP to load-balance across these unequal-metric links:

RouterA(config)# *router eigrp 10*
RouterA(config-router)# *variance 2*
RouterA(config-router)# *maximum-paths 6*

The *variance* command assigns a "multiplier," in this instance of *2*. We multiply this *variance* value by the metric of our Feasible Distance (2 x 11 = 22). Thus, any Feasible Successors with a metric within twice that of our Feasible Distance (i.e. 12 through 22) will now be used for load balancing by EIGRP.

Remember, only Feasible Successors can be used for load balancing, not Possibilities (such as the route through Router C).

The *maximum-paths* command adjusts the number of links EIGRP can load-balance across.

* * *

Troubleshooting EIGRP

To view the EIGRP Neighbor Table:

Router# *show ip eigrp neighbor*

IP-EIGRP neighbors for process 10

H	Address	Interface	Hold	Uptime	SRTT (sec)	RTO (ms)	Q Cnt	Seq Num	Type
0	172.16.1.2	S0	13	00:00:53	32	200	0	2	
0	172.18.1.2	S2	11	00:00:59	32	200	0	3	

To view the EIGRP Topology Table, containing all EIGRP route information:

Router# *show ip eigrp topology*

```
IP-EIGRP Topology Table for AS(10)/ID(172.19.1.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
            r - reply Status, s - sia Status

P 10.3.0.0/16, 1 successors, FD is 2297856
         via 172.16.1.2 (2297856/128256), Serial0
P 172.19.0.0/16, 1 successors, FD is 281600
         via Connected, Serial 1
P 172.18.0.0/16, 1 successors, FD is 128256
         via Connected, Serial 2
P 172.16.0.0/16, 1 successors, FD is 2169856
         via Connected, Serial0
```

To view information on EIGRP traffic sent and received on a router:

Router# *show ip eigrp traffic*

```
IP-EIGRP Traffic Statistics for process 10
  Hellos sent/received: 685/429
  Updates sent/received: 4/3
  Queries sent/received: 0/0
  Replies sent/received: 0/0
  Acks sent/received: 1/2
  Input queue high water mark 1, 0 drops
  SIA-Queries sent/received: 0/0
  SIA-Replies sent/received: 0/0
```

* * *

Troubleshooting EIGRP (continued)

To view the bandwidth, delay, load, reliability and MTU values of an interface:

Router# *show interface s0*

```
Serial0 is up, line protocol is up
  Hardware is HD64570
  Internet address is 172.16.1.1/16
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
     reliability 255/255, txload 1/255, rxload 1/255

<irrelevant output removed>
```

To view information specific to the EIGRP protocol:

Router# *show ip protocols*

```
Routing Protocol is "eigrp 10"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: eigrp 10
  Automatic network summarization is not in effect
  Maximum path: 4
  Routing for Networks:
    172.16.0.0
    172.18.0.0
    172.19.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    (this router)         90      00:26:11
    172.16.1.2            90      00:23:49
  Distance: internal 90 external 170
```

This command provides us with information on EIGRP timers, EIGRP metrics, summarization, and the specific networks RIP is advertising.

* * *

Troubleshooting EIGRP (continued)

To view the IP routing table:

Router# *show ip route*

```
Gateway of last resort is not set

C     172.16.0.0 is directly connected, Serial0
C     172.19.0.0 is directly connected, Serial1
D     10.3.0.0 [90/2297856] via 172.16.1.2, 00:00:15, Serial0
```

To view a specific route within the IP routing table:

Router# *show ip route 10.3.0.0*

```
Routing entry for 10.3.0.0/16
  Known via "eigrp 10", distance 90, metric 2297856 type internal
  Last update from 172.16.1.2 on Serial 0, 00:00:15 ago
```

To debug EIGRP in realtime:

Router# *debug eigrp neighbors*
Router# *debug eigrp packet*
Router# *debug eigrp route*
Router# *debug eigrp summary*

* * *

Section 20
- Open Shortest Path First -

OSPF (Open Shortest Path First)

OSPF is a standardized Link-State routing protocol, designed to scale efficiently to support larger networks.

OSPF adheres to the following Link State characteristics:

- OSPF employs a hierarchical network design using **Areas.**
- OSPF will form **neighbor** relationships with adjacent routers in the same **Area.**
- Instead of advertising the *distance* to connected networks, OSPF advertises the *status* of directly connected **links** using **Link-State Advertisements (LSAs)**.
- OSPF sends updates (LSAs) when there is a change to one of its links, and will *only* send the change in the update. LSAs are additionally refreshed every **30 minutes**.
- OSPF traffic is multicast either to address **224.0.0.5** (all OSPF routers) or **224.0.0.6** (all Designated Routers).
- OSPF uses the **Dijkstra Shortest Path First** algorithm to determine the shortest path.
- OSPF is a classless protocol, and thus supports VLSMs.

Other characteristics of OSPF include:

- OSPF supports only IP routing.
- OSPF routes have an administrative distance is **110**.
- OSPF uses **cost** as its metric, which is computed based on the bandwidth of the link. OSPF has no hop-count limit.

The OSPF process builds and maintains three separate tables:

- A **neighbor table** – contains a list of all neighboring routers.
- A **topology table** – contains a list of *all* possible routes to all known networks within an area.
- A **routing table** – contains the *best* route for each known network.

* * *

OSPF Neighbors

OSPF forms neighbor relationships, called **adjacencies,** with other routers in the same **Area** by exchanging **Hello** packets to multicast address **224.0.0.5**. Only after an adjacency is formed can routers share routing information.

Each OSPF router is identified by a unique **Router ID.** The Router ID can be determined in one of three ways:

- The Router ID can be **manually** specified.
- If not manually specified, the highest IP address configured on any **Loopback interface** on the router will become the Router ID.
- If no loopback interface exists, the highest IP address configured on any **Physical interface** will become the Router ID.

By default, Hello packets are sent out OSPF-enabled interfaces every **10 seconds** for broadcast and point-to-point interfaces, and **30 seconds** for non-broadcast and point-to-multipoint interfaces.

OSPF also has a **Dead Interval**, which indicates how long a router will wait without hearing any hellos before announcing a neighbor as "down." Default for the Dead Interval is **40 seconds** for broadcast and point-to-point interfaces, and **120** seconds for non-broadcast and point-to-multipoint interfaces. Notice that, by default, the dead interval timer is four times the Hello interval.

These timers can be adjusted on a *per interface* basis:

Router(config-if)# *ip ospf hello-interval 15*
Router(config-if)# *ip ospf dead-interval 60*

* * *

OSPF Neighbors (continued)

OSPF routers will only become neighbors if the following parameters within a Hello packet are identical on each router:

- Area ID
- Area Type (stub, NSSA, etc.)
- Prefix
- Subnet Mask
- Hello Interval
- Dead Interval
- Network Type (broadcast, point-to-point, etc.)
- Authentication

The Hello packets also serve as **keepalives** to allow routers to quickly discover if a neighbor is down. Hello packets also contain a **neighbor field** that lists the Router IDs of all neighbors the router is connected to.

A **neighbor table** is constructed from the OSPF Hello packets, which includes the following information:

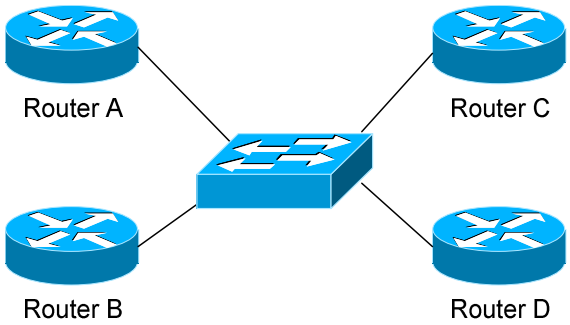
- The **Router ID** of each neighboring router
- The current "state" of each neighboring router
- The interface directly connecting to each neighbor
- The IP address of the remote interface of each neighbor

(Reference: http://www.cisco.com/warp/public/104/29.html)

OSPF Designated Routers

In multi-access networks such as Ethernet, there is the possibility of *many* neighbor relationships on the same physical segment. In the above example, four routers are connected into the same multi-access segment. Using the following formula (where "n" is the number of routers):

n(n-1)/2



…..it is apparent that **6** separate adjacencies are needed for a fully meshed network. Increase the number of routers to five, and **10** separate adjacencies would be required. This leads to a considerable amount of unnecessary Link State Advertisement (LSA) traffic.

If a link off of Router A were to fail, it would flood this information to all neighbors. Each neighbor, in turn, would then flood that same information to all other neighbors. This is a waste of bandwidth and processor load.

To prevent this, OSPF will elect a **Designated Router (DR)** for each multi-access networks, accessed via multicast address **224.0.0.6**. For redundancy purposes, a **Backup Designated Router (BDR)** is also elected.

OSPF routers will form adjacencies with the DR and BDR. If a change occurs to a link, the update is forwarded only to the DR, which then forwards it to all other routers. This greatly reduces the flooding of LSAs.

DR and BDR elections are determined by a router's **OSPF priority**, which is configured on a per-interface basis (a router can have interfaces in multiple multi-access networks). The router with the **highest priority** becomes the DR; second highest becomes the BDR. If there is a tie in priority, whichever router has the **highest Router ID** will become the DR. To change the priority on an interface:

Router(config-if)# *ip ospf priority 125*

Default priority on Cisco routers is **1**. A priority of **0** will prevent the router from being elected DR or BDR. **Note:** The DR election process is ***not preemptive***. Thus, if a router with a higher priority is added to the network, it will *not* automatically supplant an existing DR. Thus, a router that should never become the DR should always have its priority set to 0.

* * *

OSPF Neighbor States

Neighbor adjacencies will progress through several **states**, including:

Down – indicates that no Hellos have been heard from the neighboring router.

Init – indicates a Hello packet has been heard from the neighbor, but two-way communication has not yet been initialized.

2-Way – indicates that bidirectional communication has been established. Recall that Hello packets contain a *neighbor* field. Thus, communication is considered 2-Way once a router sees its own Router ID in its neighbor's Hello Packet. **Designated** and **Backup Designated Routers** are elected at this stage.

ExStart – indicates that the routers are preparing to share link state information. Master/slave relationships are formed between routers to determine who will begin the exchange.

Exchange – indicates that the routers are exchanging **Database Descriptors (DBDs).** DBDs contain a description of the router's Topology Database. A router will examine a neighbor's DBD to determine if it has information to share.

Loading – indicates the routers are finally exchanging **Link State Advertisements,** containing information about all links connected to each router. Essentially, routers are sharing their topology tables with each other.

Full – indicates that the routers are fully synchronized. The topology table of all routers in the area should now be identical. Depending on the "role" of the neighbor, the state may appear as:

- **Full/DR** – indicating that the neighbor is a Designated Router (DR)
- **Full/BDR** – indicating that the neighbor is a Backup Designated Router (BDR)
- **Full/DROther –** indicating that the neighbor is neither the DR or BDR

On a multi-access network, OSPF routers will *only* form Full adjacencies with DRs and BDRs. Non-DRs and non-BDRs will still form adjacencies, but will remain in a **2-Way State**. This is normal OSPF behavior.

* * *

OSPF Network Types

OSPF's functionality is different across several different network topology types. OSPF's interaction with Frame Relay will be explained in another section

Broadcast Multi-Access – indicates a topology where broadcast occurs.

- Examples include Ethernet, Token Ring, and ATM.
- OSPF *will* elect DRs and BDRs.
- Traffic to DRs and BDRs is multicast to 224.0.0.6. Traffic from DRs and BDRs to other routers is multicast to 224.0.0.5.
- Neighbors *do not* need to be manually specified.

Point-to-Point – indicates a topology where two routers are directly connected.

- An example would be a point-to-point T1.
- OSPF *will not* elect DRs and BDRs.
- All OSPF traffic is multicast to 224.0.0.5.
- Neighbors *do not* need to be manually specified.

Point-to-Multipoint – indicates a topology where one interface can connect to multiple destinations. Each connection between a source and destination is treated as a point-to-point link.

- An example would be Point-to-Multipoint Frame Relay.
- OSPF *will not* elect DRs and BDRs.
- All OSPF traffic is multicast to 224.0.0.5.
- Neighbors *do not* need to be manually specified.

Non-broadcast Multi-access Network (NBMA) – indicates a topology where one interface can connect to multiple destinations; however, broadcasts cannot be sent across a NBMA network.

- An example would be Frame Relay.
- OSPF *will* elect DRs and BDRs.
- OSPF neighbors must be *manually* defined, thus All OSPF traffic is unicast instead of multicast.

Remember: on *non-broadcast* networks, neighbors must be **manually specified**, as multicast Hello's are not allowed.

* * *

Configuring OSPF Network Types

The default OSPF network type for basic Frame Relay is **Non-broadcast Multi-access Network (NBMA).** To configure manually:

```
Router(config)# interface s0
Router(config-if)# encapsulation frame-relay
Router(config-if)# frame-relay map ip 10.1.1.1 101
Router(config-if)# ip ospf network non-broadcast

Router(config)# router ospf 1
Router(config-router)# neighbor 10.1.1.1
```

Notice that the *neighbor* was manually specified, as multicasting is not allowed on an NBMA. However, the Frame-Relay network can be tricked into allowing broadcasts, eliminating the need to manually specify neighbors:

```
Router(config)# interface s0
Router(config-if)# encapsulation frame-relay
Router(config-if)# frame-relay map ip 10.1.1.1 101 broadcast
Router(config-if)# ip ospf network broadcast
```

Notice that the *ospf network* type has been changed to *broadcast,* and the *broadcast* parameter was added to the *frame-relay map* command. The neighbor no longer needs to be specified, as multicasts will be allowed out this map.

The default OSPF network type for Ethernet and Token Ring is **Broadcast Multi-Access.** To configure manually:

```
Router(config)# interface e0
Router(config-if)# ip ospf network broadcast
```

The default OSPF network type for T1's (HDLC or PPP) and Point-to-Point Frame Relay is **Point-to-Point.** To configure manually:

```
Router(config)# interface s0
Router(config-if)# encapsulation frame-relay

Router(config)# interface s0.1 point-to-point
Router(config-if)# frame-relay map ip 10.1.1.1 101 broadcast
Router(config-if)# ip ospf network point-to-point
```

* * *

Configuring OSPF Network Types (continued)

The default OSPF network type for Point-to-Multipoint Frame Relay is *still* **Non-broadcast Multi-access Network (NBMA).** However, OSPF supports an additional network type called **Point-to-Multipoint,** which will allow neighbor discovery to occur automatically**.** To configure:

Router(config)# *interface s0*
Router(config-if)# *encapsulation frame-relay*

Router(config)# *interface s0.2 multipoint*
Router(config-if)# *frame-relay map ip 10.1.1.1 101 broadcast*
Router(config-if)# *ip ospf network point-to-multipoint*

Additionally, a *non-broadcast* parameter can be added to the *ip ospf network* command when specifying *point-to-multipoint.*

Router(config)# *interface s0*
Router(config-if)# *encapsulation frame-relay*

Router(config)# *interface s0.2 multipoint*
Router(config-if)# *frame-relay map ip 10.1.1.1 101*
Router(config-if)# *ip ospf network point-to-multipoint non-broadcast*

Router(config)# *router ospf 1*
Router(config-router)# *neighbor 10.1.1.1*

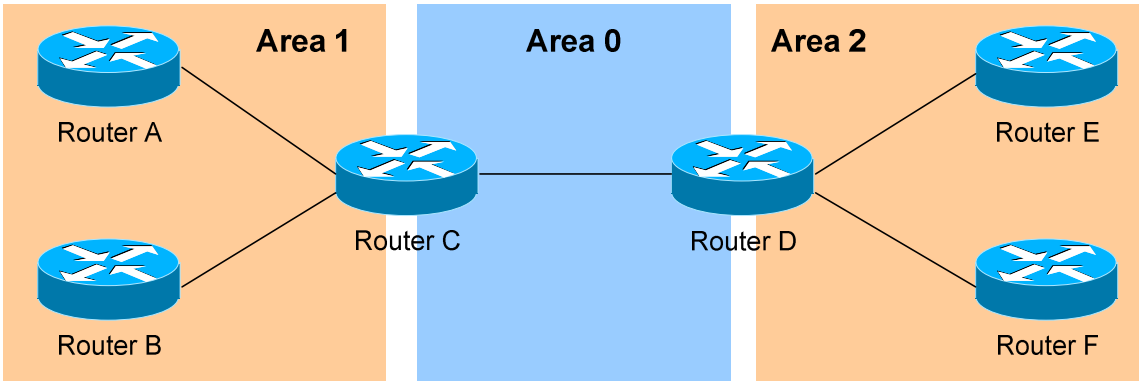
Notice the different in configuration. The *frame-relay map* command no longer has the *broadcast* parameter, as broadcasts and multicasts are not allowed on a non-broadcast network.

Thus, in the OSPF router configuration, neighbors must again be manually specified. Traffic to those neighbors will be **unicast** instead of multicast.

OSPF network types must be set identically on two "neighboring" routers, otherwise they will never form an adjacency.

* * *

The OSPF Hierarchy



OSPF is a hierarchical system that separates an Autonomous System into individual **areas**. OSPF traffic can either be **intra-area** (within one area), **inter-area** (between separate areas), or **external** (from another AS).

OSPF routers build a **Topology Database** of all **links** within their area, and all routers within an area will have an *identical* topology database. Routing updates between these routers will *only* contain information about links local to their area. Limiting the topology database to include only the local area conserves bandwidth and reduces CPU loads.

Area 0 is required for OSPF to function, and is considered the "**Backbone**" area. As a rule, all other areas must have a connection into Area 0, though this rule can be bypassed using **virtual links** (explained shortly). Area 0 is often referred to as the *transit* area to connect all other areas.

OSPF routers can belong to multiple areas, and will thus contain separate Topology databases for each area. These routers are known as **Area Border Routers (ABRs)**.

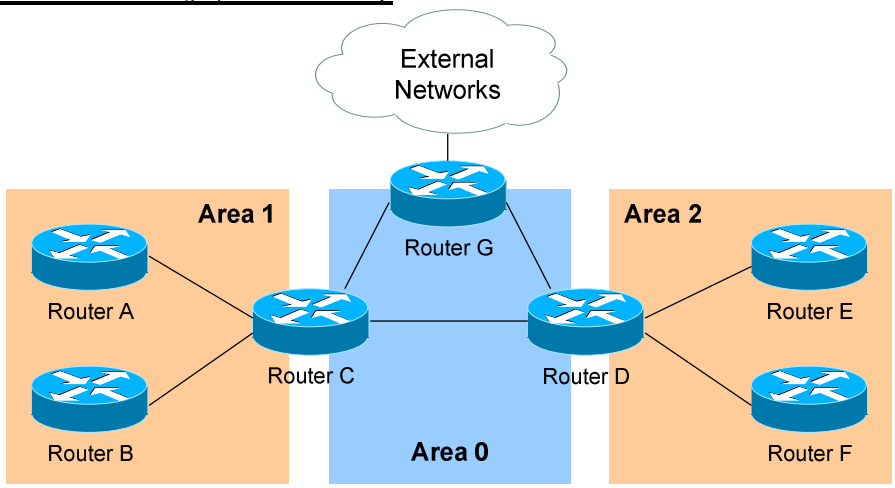
Consider the above example. Three areas exist: Area 0, Area 1, and Area 2. Area 0, again, is the backbone area for this Autonomous System. Both Area 1 and Area 2 must directly connect to Area 0.

Routers A and B belong fully to Area 1, while Routers E and F belong fully to Area 2. These are known as **Internal Routers**.

Router C belongs to both Area 0 and Area 1. Thus, it is an **ABR**. Because it has an interface in Area 0, it can also be considered a **Backbone Router**. The same can be said for Router D, as it belongs to both Area 0 and Area 2.

* * *

The OSPF Hierarchy (continued)



Now consider the above example. Router G has been added, which belongs to Area 0. However, Router G also has a connection to the Internet, which is outside this Autonomous System.

This makes Router G an **Autonomous System Border Router** (**ASBR**). A router can become an ASBR in one of two ways:

- By connecting to a separate Autonomous System, such as the Internet
- By redistributing another routing protocol into the OSPF process.

ASBRs provide access to *external* networks. OSPF defines two "types" of external routes:

- **Type 2 (E2)** – Includes only the external cost to the destination network. External cost is the metric being advertised from outside the OSPF domain. This is the default type assigned to external routes.
- **Type 1 (E1)** – Includes both the external cost, and the internal cost to reach the ASBR, to determine the total metric to reach the destination network. Type 1 routes are always *preferred* over Type 2 routes to the same destination.

Thus, the four separate OSPF router types are as follows:

- **Internal Routers –** all router interfaces belong to only one Area.
- **Area Border Routers (ABRs) –** contains interfaces in at least two separate areas
- **Backbone Routers –** contain at least one interface in Area 0
- **Autonomous System Border Routers (ASBRs) –** contain a connection to a separate Autonomous System

* * *

LSAs and the OSPF Topology Database

OSPF, as a link-state routing protocol, does not rely on *routing-by-rumor* as RIP and IGRP do.

Instead, OSPF routers keep track of the status of **links** within their respective areas. A link is simply a router interface. From these lists of links and their respective statuses, the topology database is created. OSPF routers forward **link-state advertisements (LSAs)** to ensure the topology database is consistent on each router within an area.

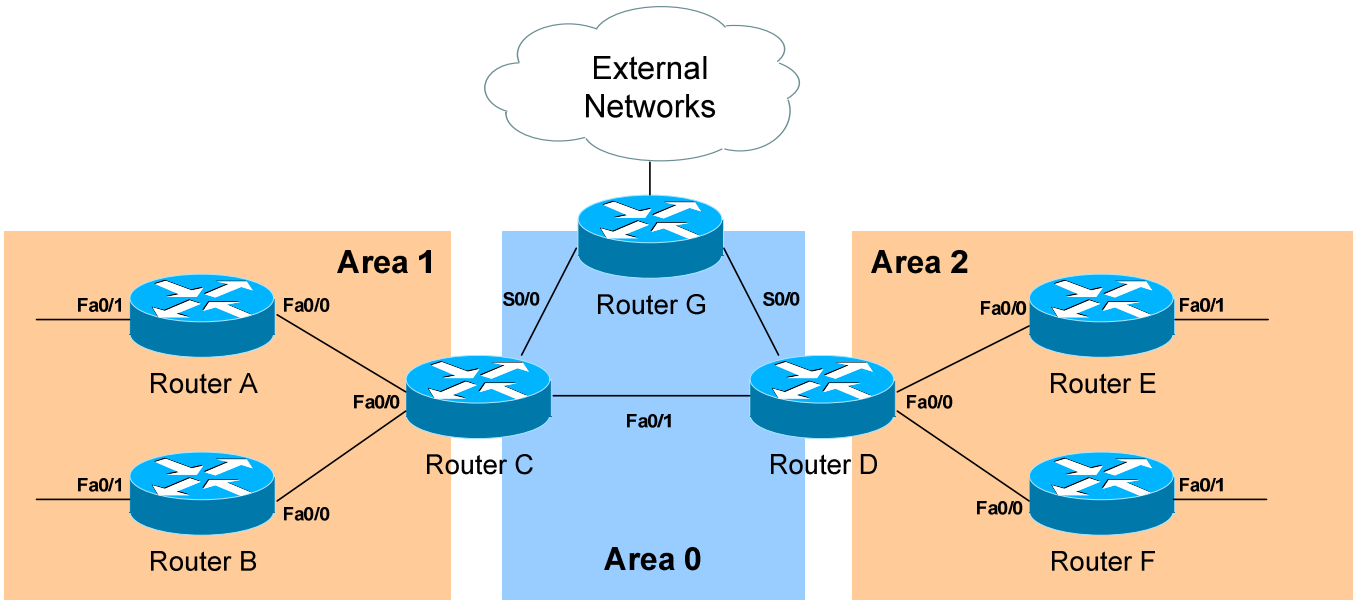
Several LSA types exist:

- **Router LSA (Type 1) –** Contains a list of all links local to the router, and the status and "cost" of those links. Type 1 LSAs are generated by all routers in OSPF, and are flooded to all other routers within the local area.
- **Network LSA (Type 2) –** Generated by all Designated Routers in OSPF, and contains a list of all routers attached to the Designated Router.
- **Network Summary LSA (Type 3)** – Generated by all ABRs in OSPF, and contains a list of all destination networks within an area. Type 3 LSAs are sent between areas to allow inter-area communication to occur.
- **ASBR Summary LSA (Type 4)** – Generated by ABRs in OSPF, and contains a *route* to any ASBRs in the OSPF system. Type 4 LSAs are sent from an ABR into its local area, so that Internal routers know how to exit the Autonomous System.
- **External LSA (Type 5)** – Generated by ASBRs in OSPF, and contain routes to destination networks *outside* the local Autonomous System. Type 5 LSAs can also take the form of a **default route** to all networks outside the local AS. Type 5 LSAs are flooded to all areas in the OSPF system.

Multicast OSPF (MOSPF) utilizes a Type 6 LSA, but that goes beyond the scope of this guide.

Later in this section, **Type 7 NSSA External LSAs** will be described in detail.

LSAs and the OSPF Topology Database (continued)



From the above example, the following can be determined:

- Routers A, B, E, and F are **Internal Routers.**
- Routers C and D are **ABRs.**
- Router G is an **ASBR.**

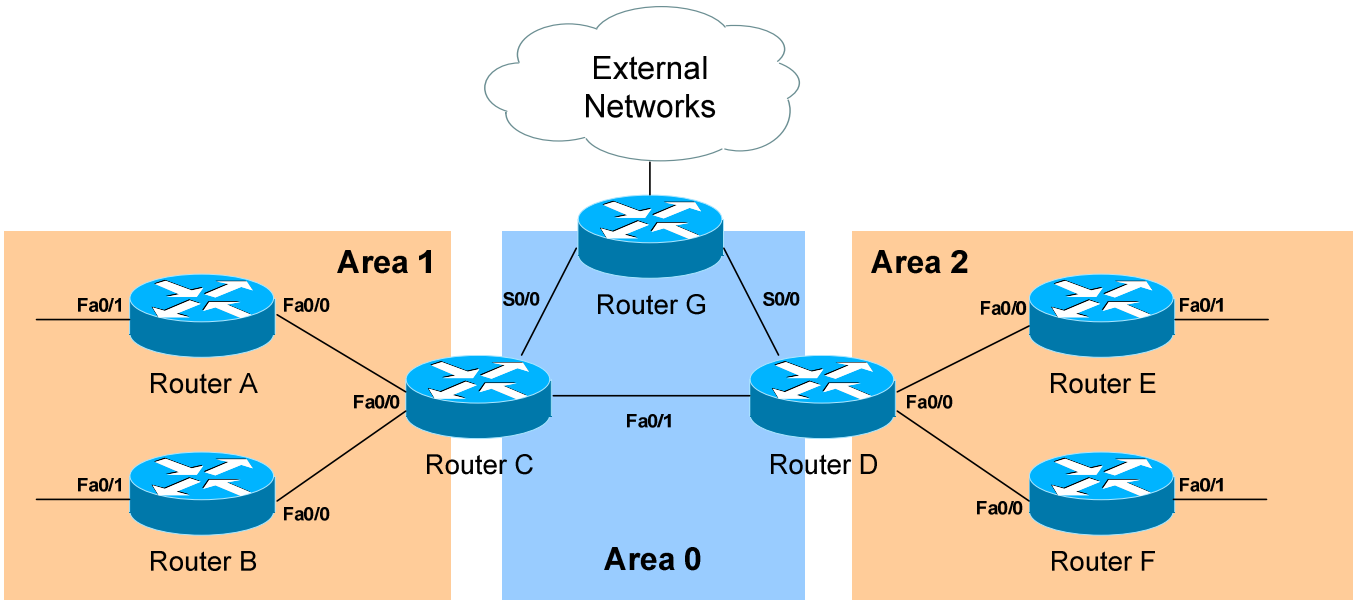
All routers will generate **Router (Type 1) LSAs**. For example, Router A will generate a Type 1 LSA that contains the status of links FastEthernet 0/0 and FastEthernet 0/1. This LSA will be flooded to all other routers in Area 1.

Designated Routers will generate **Network (Type 2) LSAs.** For example, if Router C was elected the DR for the multi-access network in Area 1, it would generate a Type 2 LSA containing a list of all routers attached to it.

Area Border Routers (ABRs) will generate **Network Summary (Type 3) LSAs**. For example, Router C is an ABR between Area 0 and Area 1. It will thus send Type 3 LSAs into *both* areas. Type 3 LSAs sent into Area 0 will contain a list of networks within Area 1, including **costs** to reach those networks. Type 3 LSAs sent into Area 1 will contain a list of networks within Area 0, *and* all other areas connected to Area 0. This allows Area 1 to reach any other area, and all other areas to reach Area 1.

* * *

LSAs and the OSPF Topology Database (continued)



ABRs will also generate **ASBR Summary (Type 4) LSAs.** For example, Router C will send Type 4 LSAs into Area 1 containing a route to the ASBR, thus providing routers in Area 1 with the path out of the Autonomous System.

ASBRs will generate **External (Type 5) LSAs.** For example, Router G will generate Type 5 LSAs that contain routes to network outside the AS. These Type 5 LSAs will be flooded to routers of all areas.

Each type of LSA is propagated under three circumstances:

- When a new adjacency is formed.
- When a change occurs to the topology table.
- When an LSA reaches its maximum age (every **30 minutes,** by default).

Thus, though OSPF is typically recognized to only send updates when a change occurs, LSA's are still periodically refreshed every 30 minutes.

The OSPF Metric

OSPF determines the best (or *shortest)* path to a destination network using a **cost** metric, which is based on the bandwidth of interfaces. The *total* cost of a route is the sum of all *outgoing* interface costs. Lowest cost is preferred.

Cisco applies default costs to specific interface types:

Type	***Cost***
Serial (56K)	*1785*
Serial (64K)	*1562*
T1 (1.544Mbps)	*64*
Token Ring (4Mbps)	*25*
Ethernet (10 Mbps)	*10*
Token Ring (16 Mbps)	*6*
Fast Ethernet	*1*

On Serial interfaces, OSPF will use the configured *bandwidth* (measured in Kbps) to determine the cost:

Router(config)# *interface s0*
Router(config-if)# *bandwidth 64*

The default cost of an interface can be superseded:

Router(config)# *interface e0*
Router(config-if)# *ip ospf cost 5*

Changing the cost of an interface can alter which path OSPF deems the "shortest," and thus should be used with great care.

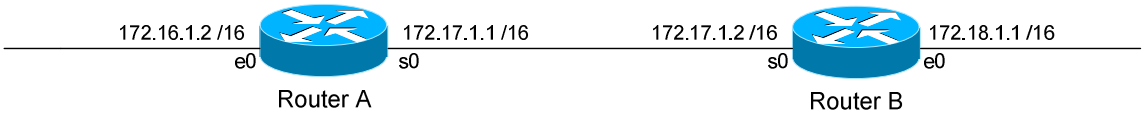
To alter how OSPF calculates its default metrics for interfaces:

Router(config)# *router ospf 1*
Router(config-router)# *ospf auto-cost reference-bandwidth 100*

The above *ospf auto-cost* command has a value of *100* configured, which is actually the default. This indicates that a 100Mbps link will have a cost of 1 (because 100/100 is 1). All other costs are based off of this. For example, the cost of 4 Mbps Token Ring is 25 because 100/4 = 25.

* * *

Configuring Basic OSPF



Routing protocol configuration occurs in Global Configuration mode. On Router A, to configure OSPF:

```
RouterA(config)# router ospf 1
RouterA(config-router)# router-id 1.1.1.1
RouterA(config-router)# network 172.16.0.0 0.0.255.255 area 1
RouterA(config-router)# network 172.17.0.0 0.0.255.255 area 0
```

The first command, *router ospf 1*, enables the OSPF process. The "*1*" indicates the OSPF process ID, and can be unique on each router. The process ID allows multiple OSPF processes to run on the same router. The *router-id* command assigns a unique OSPF ID of *1.1.1.1* for this router.

Note the use of a wildcard mask instead of a subnet mask in the *network* statement. With OSPF, we're *not* telling the router what networks to advertise; we're telling the router to place certain interfaces into specific areas, so those routers can form neighbor relationships. The wildcard mask *0.0.255.255* tells us that the last two octets can match any number.

The first *network* statement places interface E0 on Router A into Area 1. Likewise, the second *network* statement places interface S0 on Router A into Area 0. The network statement could have been written more specifically:

```
RouterA(config)# router ospf 1
RouterA(config-router)# network 172.16.1.2 0.0.0.0 area 1
RouterA(config-router)# network 172.17.1.1 0.0.0.0 area 0
```

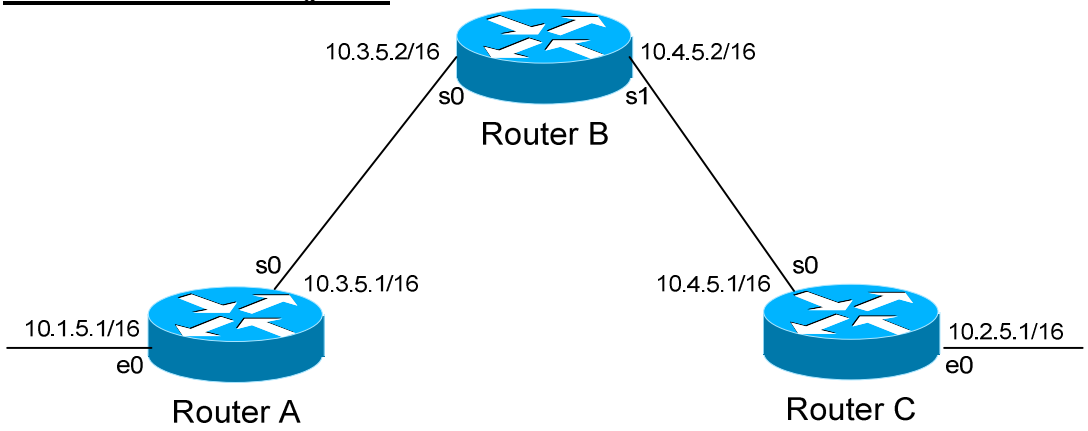
In order for Router B to form a neighbor relationship with Router A, its connecting interface must be put in the same Area as Router A:

```
RouterB(config)# router ospf 1
RouterA(config-router)# router-id 2.2.2.2
RouterB(config-router)# network 172.17.1.2 0.0.0.0 area 0
RouterB(config-router)# network 172.18.1.1 0.0.0.0 area 2
```

If Router B's S0 interface was placed in a different area than Router A's S0 interface, the two routers would never form a neighbor relationship, and never share routing updates.

* * *

OSPF Passive-Interfaces



It is possible to control which router interfaces will participate in the OSPF process. Just as with EIGRP and RIP, we can use the *passive-interface* command.

However, please note that the *passive-interface* command works differently with OSPF than with RIP or IGRP. OSPF will no longer form neighbor relationships out of a "passive" interface, thus this command prevents updates from being *sent* or *received* out of this interface:

RouterC(config)# *router ospf 1*
RouterC(config-router)# *network 10.4.0.0 0.0.255.255 area 0*
RouterC(config-router)# *network 10.2.0.0 0.0.255.255 area 0*
RouterC(config-router)# *passive-interface s0*

Router C will not form a neighbor adjacency with Router B.

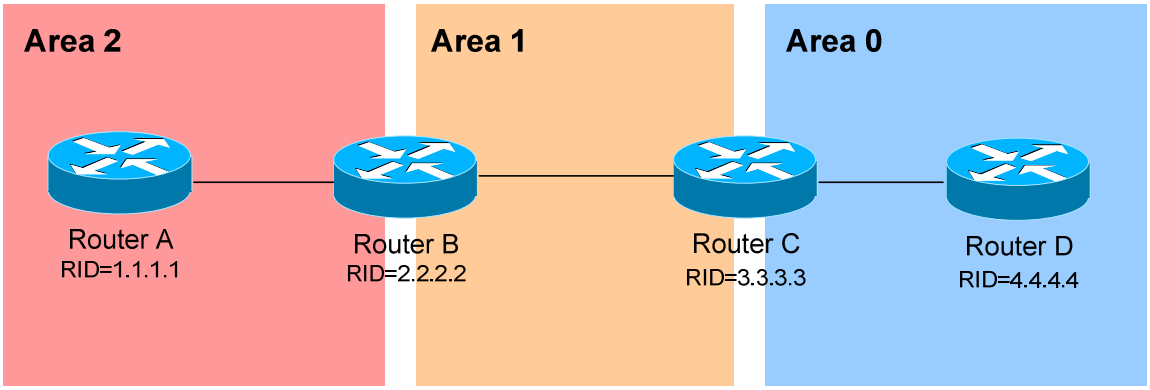
It is possible to configure **all** interfaces to be passive using the *passive-interface default* command, and then individually use the *no passive-interface* command on the interfaces that neighbors **should** be formed on:

RouterC(config)# *router ospf 1*
RouterC(config-router)# *network 10.4.0.0 0.0.255.255 area 0*
RouterC(config-router)# *network 10.2.0.0 0.0.255.255 area 0*
RouterC(config-router)# *passive-interface default*
RouterC(config-router)# *no passive-interface e0*

Always remember, that the *passive-interface* command will prevent OSPF (and EIGRP) from forming neighbor relationships out of that interface. **No** routing updates are passed in either direction.

* * *

OSPF Virtual Links



Earlier in this guide, it was stated that all areas must directly connect into Area 0, as a rule. In the above example, Area 2 has no direct connection to Area 0, but must transit through Area 1 to reach the backbone area. In normal OSPF operation, this shouldn't be possible.

There may be certain circumstances that may prevent an area from directly connecting into Area 0. **Virtual links** can be used as a workaround, to *logically* connect separated areas to Area 0. In the above example, a virtual link would essentially create a tunnel from Area 2 to Area 0, using Area 1 a **transit area**. One end of the Virtual Link *must* be connected to Area 0.

Configuration occurs on the **Area Border Routers (ABRs)** connecting Area 1 to Area 2 (Router B), and Area 1 to Area 0 (Router C). Configuration on Router B would be as follows:

```
RouterB(config)# router ospf 1
RouterB(config-router)# router-id 2.2.2.2
RouterB(config-router)# area 1 virtual-link 3.3.3.3
```

The first command enables the *ospf* process. The second command manually sets the *router-id* for Router B to *2.2.2.2*.

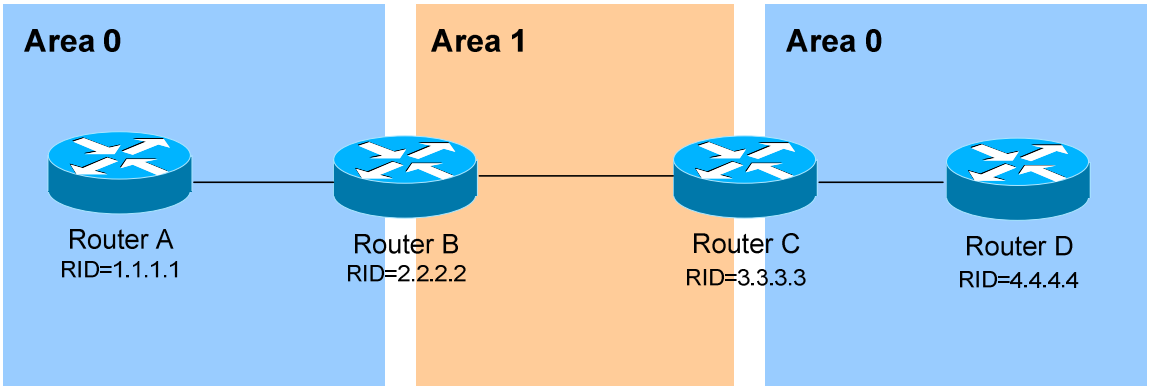
The third command actually creates the *virtual-link*. Notice that it specifies *area 1*, which is the **transit area**. Finally, the command points to the remote ABR's Router ID of *3.3.3.3*.

Configuration on Router C would be as follows:

```
RouterC(config)# router ospf 1
RouterC(config-router)# router-id 3.3.3.3
RouterC(config-router)# area 1 virtual-link 2.2.2.2
```

* * *

OSPF Virtual Links (continued)



It is also possible to have two separated (or discontiguous) Area 0's. In order for OSPF to function properly, the two Area 0's must be connected using a **virtual link**.

Again, configuration occurs on the transit area's ABRs:

RouterB(config)# *router ospf 1*
RouterB(config-router)# *router-id 2.2.2.2*
RouterB(config-router)# *area 1 virtual-link 3.3.3.3*

RouterC(config)# *router ospf 1*
RouterC(config-router)# *router-id 3.3.3.3*
RouterC(config-router)# *area 1 virtual-link 2.2.2.2*

Always remember: the area specified in the *virtual-link* command is the **transit** area. Additionally, the transit area ***cannot*** be a stub area.

As stated earlier, if authentication is enabled for Area 0, the same authentication must be configured on Virtual Links, as they are "extensions" of Area 0:

RouterB(config)# *router ospf 1*
RouterB(config-router)# *area 1 virtual-link 3.3.3.3 message-digest-key 1 md5 MYKEY*

RouterC(config)# *router ospf 1*
RouterC(config-router)# *area 1 virtual-link 2.2.2.2 message-digest-key 1 md5 MYKEY*

* * *

Troubleshooting OSPF

To view the OSPF Neighbor Table:

Router# *show ip ospf neighbor*

Neighbor ID	Pri	State	Dead Time	Address	Interface
7.7.7.7	1	FULL/ -	00:00:36	150.50.17.2	Serial0
6.6.6.6	1	FULL/DR	00:00:11	150.50.18.1	Ethernet0

The Neighbor Table provides the following information about each neighbor:

- The ***Router ID*** of the remote neighbor.
- The OSPF ***priority*** of the remote neighbor (used for DR/BDR elections).
- The current neighbor ***state***.
- The ***dead interval*** timer.
- The connecting ***IP address*** of the remote neighbor.
- The local ***interface*** connecting to the remote neighbor.

To view the OSPF topology table:

Router# *show ip ospf database*

OSPF Router with ID (9.9.9.9) (Process ID 10)

Router Link States (Area 0)

Link ID	ADV Router	Age	Seq#	Checksum	Link count
7.7.7.7	7.7.7.7	329	0x80000007	0x42A0	2
8.8.8.8	8.8.8.8	291	0x80000007	0x9FFC	1

Summary Net Link States (Area 0)

Link ID	ADV Router	Age	Seq#	Checksum
192.168.12.0	7.7.7.7	103	0x80000005	0x13E4
192.168.34.0	7.7.7.7	105	0x80000003	0x345A

The Topology Table provides the following information:

- The actual ***link*** (or *route*).
- The ***advertising*** Router ID.
- The link-state ***age*** timer.
- The ***sequence number*** and ***checksum*** for each entry.

(Reference: http://www.cisco.com/en/US/products/sw/iosswrel/ps5187/products_command_reference_chapter09186a008017d02e.html)

* * *

Troubleshooting OSPF (continued)

To view the specific information about an OSPF process:

Router# *show ip ospf 1*

```
Routing Process "ospf 1" with ID 9.9.9.9
 Supports only single TOS(TOS0) routes
 Supports opaque LSA
 SPF schedule delay 5 secs, Hold time between two SPFs 10 secs
 Minimum LSA interval 5 secs. Minimum LSA arrival 1 secs
 Number of external LSA 0. Checksum Sum 0x0
 Number of opaque AS LSA 0. Checksum Sum 0x0
 Number of DCbitless external and opaque AS LSA 0
 Number of DoNotAge external and opaque AS LSA 0
 Number of areas in this router is 1. 1 normal 0 stub 0 nssa
 External flood list length 0
   Area BACKBONE(0)
      Number of interfaces in this area is 1
      Area has no authentication
      SPF algorithm executed 3 times
      Area ranges are
      Number of LSA 2. Checksum Sum 0xDDEC
      Number of opaque link LSA 0. Checksum Sum 0x0
      Number of DCbitless LSA 0
      Number of indication LSA 0
      Number of DoNotAge LSA 0
      Flood list length 0
```

The *show ip ospf* command provides the following information:

- The local ***Router ID.***
- ***SPF Scheduling*** information, and various ***SPF timers.***
- The number of ***interfaces*** in specific ***areas***, including the ***type*** of area.
- The link-state ***age*** timer.
- The ***sequence number*** and ***checksum*** for each entry.

Troubleshooting OSPF (continued)

To view OSPF-specific information on an interface:

Router# *show ip ospf interface s0*

```
Serial0 is up, line protocol is up
  Internet Address 192.168.79.2/24, Area 0
  Process ID 10, Router ID 9.9.9.9, Network Type POINT_TO_POINT, Cost: 64
  Transmit Delay is 1 sec, State POINT_TO_POINT,
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:04
  Index 1/1, flood queue length 0
  Next 0x0(0)/0x0(0)
  Last flood scan length is 1, maximum is 1
  Last flood scan time is 0 msec, maximum is 0 msec
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 7.7.7.7
  Suppress hello for 0 neighbor(s)
```

The *show ip ospf interface* command provides the following information:

- The local ***Router ID***.
- The interface ***network type***.
- The OSPF ***cost*** for the interface.
- The interface ***Hello*** and ***Dead*** timers.
- A list of neighbor ***adjacencies***.

To view routing protocol specific information for OSPF:

Router# *show ip protocols*

```
Routing Protocol is "ospf 10"
  Invalid after 0 seconds, hold down 0, flushed after 0
  Outgoing update filter list for all interfaces is
  Incoming update filter list for all interfaces is
  Routing for Networks:
    192.168.79.0  0.0.0.255 area 0
    192.168.109.0 0.0.0.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
    7.7.7.7          110            00:01:05
  Distance: (default is 110)
```

The *show ip protocols* command provides the following information:

- Locally originated ***networks*** that are being advertised.
- Neighboring ***sources*** for routing information
- The ***administrative distance*** of neighboring sources.

* * *

Troubleshooting OSPF (continued)

To reset an OSPF process, including neighbor adjacencies:

Router# *clear ip ospf process*

To display information about OSPF virtual-links:

Router# *show ip ospf virtual-links*

To display routes to both ABRs and ASBRs:

Router# *show ip ospf border-routers*

To debug OSPF in realtime:

Router# *debug ip ospf adj*
Router# *debug ip ospf events*
Router# *debug ip ospf hello*

* * *

Part IV

VLANs, Access-Lists, and Services

Section 21
- 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**.

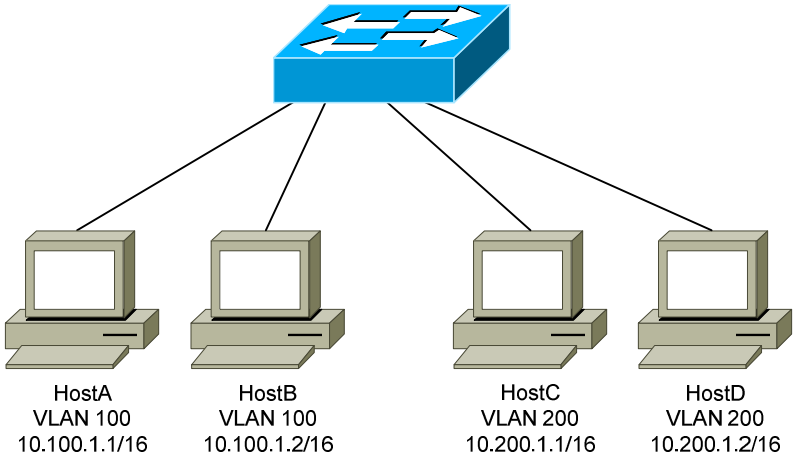
Thus, a switch that supports VLANs is not necessarily a Layer-3 switch. However, a purely Layer-2 switch cannot route between VLANs.

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.

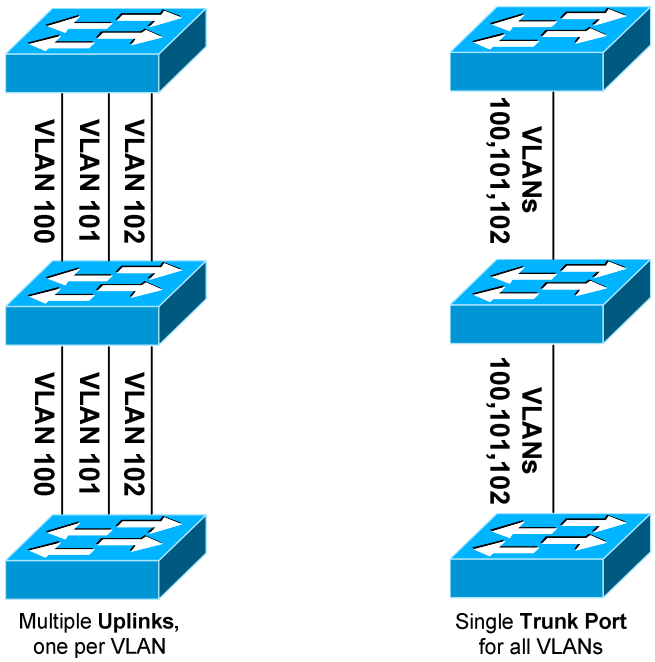
Any host connected to an access port immediately becomes a member of the VLAN configured on that port. This is *transparent* to the host - it is unaware that it belongs to a VLAN.

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:

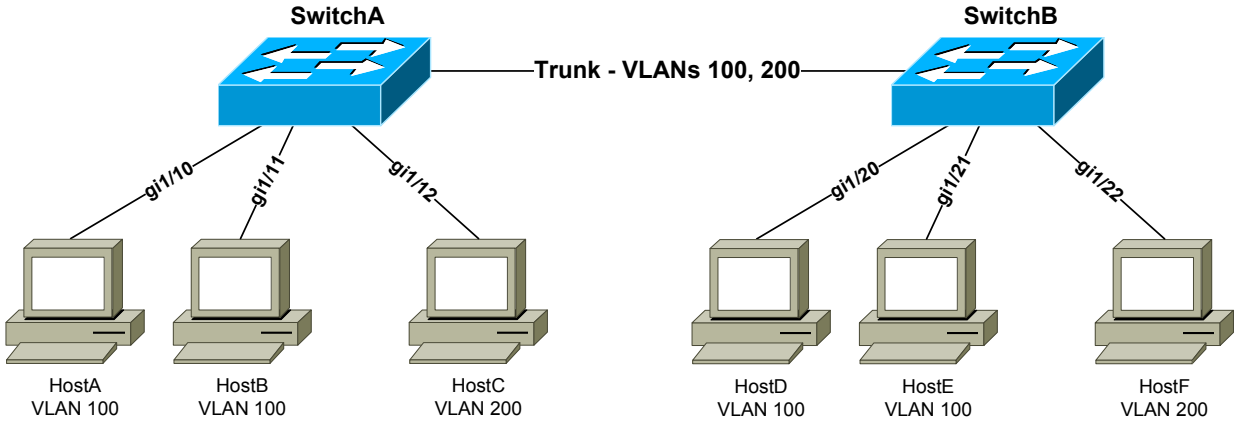


* * *

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:

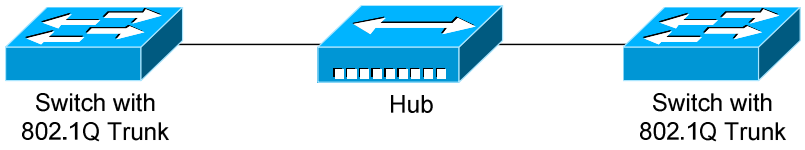
```
Switch(config)#  interface gi2/24
Switch(config-if)#  switchport mode trunk
Switch(config-if)#  switchport trunk native vlan 42
```

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:

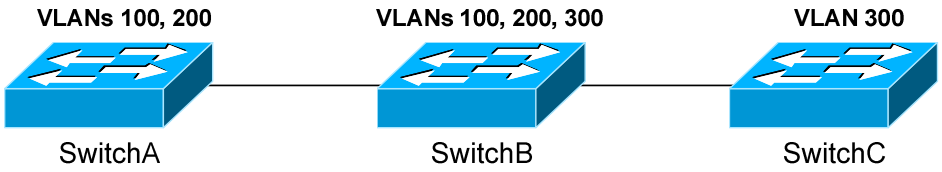
Switch# *show vtp counters*

* * *

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 22
- Access Control Lists -

Access Control Lists (ACLs)

Access control lists (ACLs) can be used for two purposes on Cisco devices: to **filter** traffic, and to **identify** traffic.

Access lists are a set of rules, organized in a rule table. Each rule or line in an access-list provides a condition, either **permit** or **deny**:

- When using an access-list to filter traffic, a *permit* statement is used to "allow" traffic, while a *deny* statement is used to "block" traffic.
- Similarly, when using an access list to identify traffic, a *permit* statement is used to "include" traffic, while a *deny* statement states that the traffic should "not" be included. It is thus interpreted as a **true/false** statement.

Filtering traffic is the primary use of access lists. However, there are several instances when it is necessary to identify traffic using ACLs, including:

- Identifying interesting traffic to bring up an ISDN link or VPN tunnel
- Identifying routes to filter or allow in routing updates
- Identifying traffic for QoS purposes

When filtering traffic, access lists are applied on interfaces. As a packet passes through a router, the top line of the rule list is checked first, and the router continues to go down the list until a match is made. Once a match is made, the packet is either permitted or denied.

There is an implicit 'deny all' at the end of all access lists. You don't create it, and you can't delete it. Thus, access lists that contain **only deny statements** will **prevent all traffic**.

Access lists are applied either inbound (packets received on an interface, before routing), or outbound (packets leaving an interface, *after* routing). Only one access list **per interface**, **per protocol**, **per direction** is allowed.

More specific and frequently used rules should be at the top of your access list, to optimize CPU usage. New entries to an access list are added to the bottom. You **cannot remove individual lines** from a numbered access list. You must delete and recreate the access to truly make changes. Best practice is to use a text editor to manage your access-lists.

* * *

Types of Access Lists

There are two categories of access lists: **numbered** and **named**.

Numbered access lists are broken down into several ranges, each dedicated to a specific protocol:

1–99	IP standard access list
100-199	IP extended access list
200-299	Protocol type-code access list
300-399	DECnet access list
400-499	XNS standard access list
500-599	XNS extended access list
600-699	Appletalk access list
700-799	48-bit MAC address access list
800-899	IPX standard access list
900-999	IPX extended access list
1000-1099	IPX SAP access list
1100-1199	Extended 48-bit MAC address access list
1200-1299	IPX summary address access list
1300-1999	IP standard access list (expanded range)
2000-2699	IP extended access list (expanded range

Remember, individual lines *cannot* be removed from a numbered access list. The entire access list must be deleted and recreated. All new entries to a numbered access list are added to the bottom.

Named access lists provide a bit more flexibility. Descriptive names can be used to identify your access-lists. Additionally, individual lines *can* be removed from a named access-list. However, like numbered lists, all new entries are still added to the bottom of the access list.

There are two common types of named access lists:

- IP standard named access lists
- IP extended named access lists

Configuration of both numbered and named access-lists is covered later in this section.

Wild Card Masks

IP access-lists use **wildcard masks** to determine two things:

1. Which part of an address must match exactly
2. Which part of an address can match any number

This is as opposed to a **subnet mask**, which tells us what part of an address is the network (subnet), and what part of an address is the host. Wildcard masks look like inversed subnet masks.

Consider the following address and wildcard mask:

Address: 172.16.0.0
Wild Card Mask: 0.0.255.255

The above would match any address that begins "172.16." The last two octets could be anything. How do I know this?

Two Golden Rules of Access Lists:

1. If a bit is set to **0** in a wild-card mask, the corresponding bit in the address must be **matched exactly.**
2. If a bit is set to **1** in a wild-card mask, the corresponding bit in the address can **match any number.** In other words, we "don't care" what number it matches.

To see this more clearly, we'll convert both the address and the wildcard mask into binary:

Address: 10101100.00010000.00000000.00000000
Wild Card Mask: 00000000.00000000.11111111.11111111

Any **0** bits in the wildcard mask, indicates that the corresponding bits in the address must be matched exactly. Thus, looking at the above example, we must exactly match the following in the first two octets:

10101100.00010000 = 172.16

Any **1** bits in the wildcard mask indicates that the corresponding bits can be anything. Thus, the last two octets can be any number, and it will still match this access-list entry.

* * *

Wild Card Masks (continued)

If wanted to match a **specific address** with a wildcard mask (we'll use an example of 172.16.1.1), how would we do it?

Address: 172.16.1.1
Wild Card Mask: 0.0.0.0

Written out in binary, that looks like:

Address: 10101100.00010000.00000001.00000001
Wild Card Mask: 00000000.00000000.00000000.00000000

Remember what a wildcard mask is doing. A **0** indicates it must match exactly, a **1** indicates it can match anything. The above wildcard mask has all bits set to 0, which means we must match all four octets exactly.

There are actually two ways we can match a host:

- Using a wildcard mask with all bits set to 0 – **172.16.1.1 0.0.0.0**
- Using the keyword "host" – **host 172.16.1.1**

How would we match **all addresses** with a wildcard mask?

Address: 0.0.0.0
Wild Card Mask: 255.255.255.255

Written out in binary, that looks like:

Address: 00000000.00000000.00000000.00000000
Wild Card Mask: 11111111.11111111.11111111.11111111

Notice that the above wildcard mask has all bits set to 1. Thus, each bit can match anything – resulting in the above address and wildcard mask matching all possible addresses.

There are actually two ways we can match all addresses:

- Using a wildcard mask with all bits set to 1 – **0.0.0.0 255.255.255.255**
- Using the keyword "any" – **any**

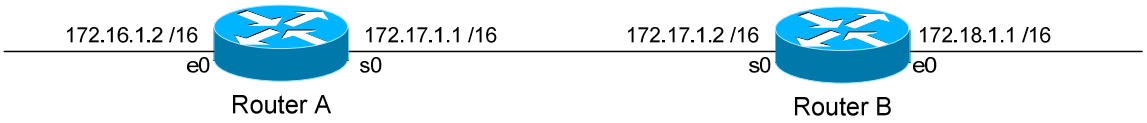
* * *

Standard IP Access List

access-list *[1-99] [permit | deny] [source address] [wildcard mask] [log]*

Standard IP access-lists are based upon the source host or network IP address, and should be placed closest to the destination network.

Consider the following example:



In order to block network 172.18.0.0 from accessing the 172.16.0.0 network, we would create the following access-list on Router A:

Router(config)# *access-list 10 deny 172.18.0.0 0.0.255.255*
Router(config)# *access-list 10 permit any*

Notice the wildcard mask of 0.0.255.255 on the first line. This will match (*deny*) all hosts on the 172.18.x.x network.

The second line uses a keyword of *any*, which will match (*permit)* any other address. Remember that you must have at least one permit statement in your access list.

To apply this access list, we would configure the following on Router A:

Router(config)# *int s0*
Router(config-if)# *ip access-group 10 in*

To view all IP access lists configured on the router:

Router# *show ip access-list*

To view what interface an access-list is configured on:

Router# *show ip interface*
Router# *show running-config*

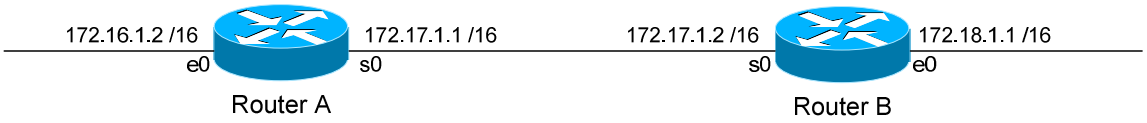
* * *

Extended IP Access List

access-list *[100-199] [permit | deny] [protocol] [source address] [wildcard mask] [destination address] [wildcard mask] [operator [port]] [log]*

Extended IP access-lists block based upon the source IP address, destination IP address, and TCP or UDP port number. Extended access-lists should be placed closest to the source network.

Consider the following example:



Assume there is a webserver on the 172.16.x.x network with an IP address of 172.16.10.10. In order to block network 172.18.0.0 from accessing anything on the 172.16.0.0 network, EXCEPT for the HTTP port on the web server, we would create the following access-list on Router B:

Router(config)# *access-list 101 permit tcp 172.18.0.0 0.0.255.255 host 172.16.10.10 eq 80*
Router(config)# *access-list 101 deny ip 172.18.0.0 0.0.255.255 172.16.0.0 0.0.255.255*
Router(config)# *access-list 101 permit ip any any*

The first line allows the 172.18.x.x network access only to port 80 on the web server. The second line blocks 172.18.x.x from accessing anything else on the 172.16.x.x network. The third line allows 172.18.x.x access to anything else.

We could have identified the web server in one of two ways:

Router(config)# *access-list 101 permit tcp 172.18.0.0 0.0.255.255 host 172.16.10.10 eq 80*
Router(config)# *access-list 101 permit tcp 172.18.0.0 0.0.255.255 172.16.10.10 0.0.0.0 eq 80*

To apply this access list, we would configure the following on Router B:

Router(config)# *int e0*
Router(config-if)# *ip access-group 101 in*

* * *

Extended IP Access List Port Operators

In the preceding example, we identified TCP port 80 on a specific host use the following syntax:

Router(config)# *access-list 101 permit tcp 172.18.0.0 0.0.255.255 host 172.16.10.10 eq 80*

We accomplished this using an operator of *eq*, which is short for **equals**. Thus, we are identifying host *172.16.10.10* with a port that *eq*uals 80.

We can use several other operators for port numbers:

eq	Matches a specific port
gt	Matches all ports greater than the port specified
lt	Matches all ports less than the port specified
neq	Matches all ports except for the port specified
range	Match a specific inclusive range of ports

The following will match all ports *greater* than *100*:

Router(config)# *access-list 101 permit tcp any host 172.16.10.10 gt 100*

The following will match all ports *less* than *1024*:

Router(config)# *access-list 101 permit tcp any host 172.16.10.10 lt 1024*

The following will match all ports that do *not equal 443*:

Router(config)# *access-list 101 permit tcp any host 172.16.10.10 neq 443*

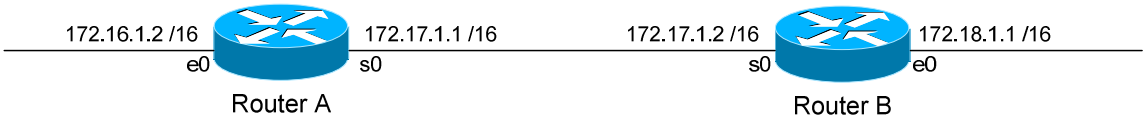
The following will match all ports between *80* and *88*:

Router(config)# *access-list 101 permit tcp any host 172.16.10.10 range 80 88*

* * *

Access List Logging

Consider again the following example:



Assume there is a webserver on the 172.16.x.x network with an IP address of 172.16.10.10.

We wish to keep track of the number of packets permitted or denied by each line of an access-list. Access-lists have a built-in logging mechanism for such a purpose:

Router(config)# *access-list 101 permit tcp 172.18.0.0 0.0.255.255 host 172.16.10.10 eq 80 log*
Router(config)# *access-list 101 deny ip 172.18.0.0 0.0.255.255 172.16.0.0 0.0.255.255 log*
Router(config)# *access-list 101 permit ip any any log*

Notice we added an additional keyword *log* to each line of the access-list. When viewing an access-list using the following command:

Router# *show access-list 101*

We will now have a counter on each line of the access-list, indicating the number of packets that were permitted or denied by that line. This information can be sent to a syslog server:

Router(config)# *logging on*
Router(config)# *logging 172.18.1.50*

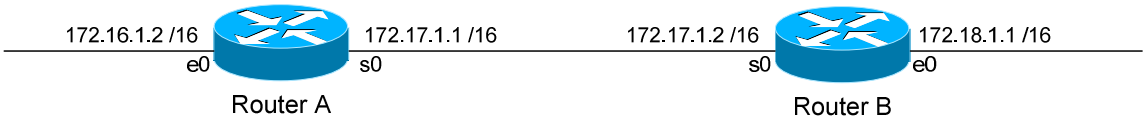
The *logging on* command enables logging. The second *logging* command points to a syslog host at *172.18.1.50.*

We can include more detailed logging information, including the source MAC address of the packet, and what interface that packet was received on. To accomplish this, use the *log-input* argument:

Router(config)# *access-list 101 permit ip any any log-input*

* * *

ICMP Access List



Consider this scenario. You've been asked to block anyone from the 172.18.x.x network from "pinging" anyone on the 172.16.x.x network. You want to allow everything else, including all other ICMP packets.

The specific ICMP port that a "ping" uses is **echo**. To block specific ICMP parameters, use an extended IP access list. On Router B, we would configure:

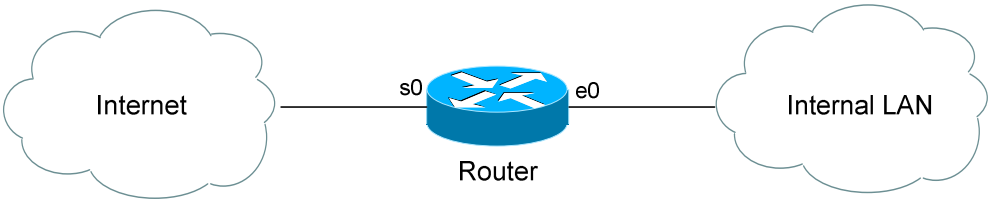
Router(config)# access-list 102 deny icmp 172.18.0.0 0.0.255.255 172.16.0.0 0.0.255.255 echo
Router(config)# access-list 102 permit icmp 172.18.0.0 0.0.255.255 172.16.0.0 0.0.255.255
Router(config)# access-list 102 permit ip any any

The first line blocks only ICMP echo requests (pings). The second line allows all other ICMP traffic. The third line allows all other IP traffic.

Don't forget to apply it to an interface on Router B:

Router(config)# *int e0*
Router(config-if)# *ip access-group 102 in*

Untrusted networks (such as the Internet) should usually be blocked from pinging an outside router or any internal hosts:



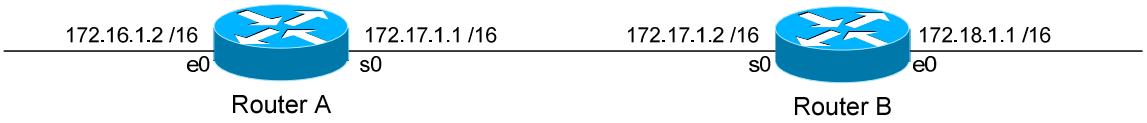
Router(config)# *access-list 102 deny icmp any any*
Router(config)# *access-list 102 permit ip any any*

Router(config)# *interface s0*
Router(config-if)# *ip access-group 102 in*

The above access-list completed disables ICMP on the serial interface. However, this would effectively disable ICMP traffic *in both directions* on the router. Any replies to pings initiated by the Internal LAN would be blocked on the way back in.

* * *

Telnet Access List



We can create access lists to restrict telnet access to our router. For this example, we'll create an access list that prevents anyone from the evil 172.18.x.x network from telneting into Router A, but allow all other networks telnet access.

First, we create the access-list on Router A:

Router(config)# *access-list 50 deny 172.18.0.0 0.0.255.255*
Router(config)# *access-list 50 permit any*

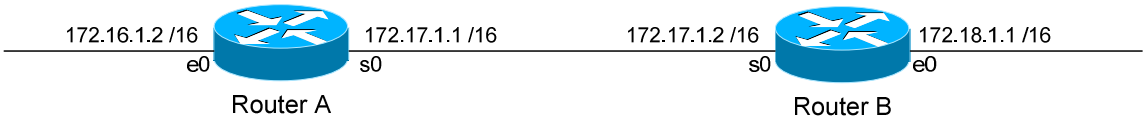
The first line blocks the 172.18.x.x network. The second line allows all other networks.

To apply it to Router A's telnet ports:

Router(config)# *line vty 0 4*
Router(config-line)# *access-class 50 in*

* * *

Named Access Lists



Named access lists provide us with two advantages over numbered access lists. First, we can apply an identifiable name to an access list, for documentation purposes. Second, we can remove individual lines in a named access-list, which is not possible with numbered access lists.

Please note, though we can *remove* individual lines in a named access list, we cannot *insert* individual lines into that named access list. New entries are always placed at the bottom of a named access list.

To create a standard named access list, the syntax would be as follows:

Router(config)# *ip access-list standard NAME*
Router(config-std-nacl)# *deny 172.18.0.0 0.0.255.255*
Router(config-std-nacl)# *permit any*

To create an extended named access list, the syntax would be as follows:

Router(config)# *ip access-list extended NAME*
Router(config-ext-nacl)# *permit tcp 172.18.0.0 0.0.255.255 host 172.16.10.10 eq 80*
Router(config-ext-nacl)# *deny ip 172.18.0.0 0.0.255.255 172.16.0.0 0.0.255.255*
Router(config-ext-nacl)# *permit ip any any*

Notice that the actual configuration of the named access-list is performed in a separate router "mode":

Router(config-std-nacl)#

Router(config-ext-nacl)#

* * *

Time-Based Access-Lists

Beginning with IOS version 12.0, access-lists can be based on the time and the day of the week.

The first step to creating a time-based access-list, is to create a *time-range*:

Router(config)# *time-range BLOCKHTTP*

The above command creates a *time-range* named *BLOCKHTTP*. Next, we must either specify an *absolute* time, or a *periodic* time:

Router(config)# *time-range BLOCKHTTP*
Router(config-time-range)# *absolute start 08:00 23 May 2006 end 20:00 26 May 2006*

Router(config)# *time-range BLOCKHTTP*
Router(config-time-range)# *periodic weekdays 18:00 to 23:00*

Notice the use of military time. The first *time-range* sets an *absolute* time that will *start* from May 23, 2006 at 8:00 a.m., and will *end* on May 26, 2006 at 8:00 p.m.

The second time-range sets a *periodic* time that is always in effect on *weekdays* from 6:00 p.m. to 11:00 p.m.

Only one *absolute* time statement is allowed per time-range, but multiple *periodic* time statements are allowed.

After we establish our time-range, we must reference it in an access-list:

Router(config)# *access-list 102 deny any any eq 80 time-range BLOCKHTTP*
Router(config)# *access-list 102 permit ip any any*

Notice the *time-range* argument at the end of the access-list line. This will result in HTTP traffic being blocked, but only during the time specified in the time-range.

Source:
(*http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120t/120t1/timerang.htm*)

* * *

Advanced Wildcard Masks

Earlier in this section, we discussed the basics of wildcard masks. The examples given previously matched one of three things:

- A specific host
- A specific octet(s)
- All possible hosts

It is also possible to match groups or ranges of hosts with wildcard masks. For example, assume we wanted a standard access-list that denied the following hosts:

172.16.1.4
172.16.1.5
172.16.1.6
172.16.1.7

We could create an access-list with four separate lines:

Router(config)# *access-list 10 deny 172.16.1.4 0.0.0.0*
Router(config)# *access-list 10 deny 172.16.1.5 0.0.0.0*
Router(config)# *access-list 10 deny 172.16.1.6 0.0.0.0*
Router(config)# *access-list 10 deny 172.16.1.7 0.0.0.0*

However, it is also possible to match all four addresses in **one** line:

Router(config)# *access-list 10 deny 172.16.1.4 0.0.0.3*

How do I know this is correct? Let's write out the above four addresses, and my wildcard mask in binary:

172.16.1.4:	10101100.00010000.00000001.00000100
172.16.1.5:	10101100.00010000.00000001.00000101
172.16.1.6:	10101100.00010000.00000001.00000110
172.16.1.7:	10101100.00010000.00000001.00000111
Wild Card Mask:	00000000.00000000.00000000.00000011

Notice that the first 30 bits of each of the four addresses are identical. Each begin "*10101100.00010000.00000001.000001*". Since those bits must match exactly, the first 30 bits of our wildcard mask are set to **0**.

* * *

Advanced Wildcard Masks (continued)

Notice now that the *only* bits that are different between the four addresses are the last two bits. Not only that, but we use every computation of those last two bits: 00, 01, 10, 11.

Thus, since those last two bits can be anything, the last two bits of our wildcard mask are set to **1**.

The resulting access-list line:

Router(config)# *access-list 10 deny 172.16.1.4 0.0.0.3*

We also could have determined the appropriate address and wildcard mask by using AND/XOR logic.

To determine the address, we perform a logical **AND** operation:

1. If all bits in a column are set to **0**, the corresponding address bit is **0**
2. If all bits in a column are set to **1,** the corresponding address bit is **1**
3. If the bits in a column are a mix of **0**'s and **1's**, the corresponding address bit is a **0.**

Observe:

172.16.1.4:	10101100.00010000.00000001.00000100
172.16.1.5:	10101100.00010000.00000001.00000101
172.16.1.6:	10101100.00010000.00000001.00000110
172.16.1.7:	10101100.00010000.00000001.00000111
Result:	10101100.00010000.00000001.00000100

Our resulting address is **172.16.1.4**. This gets us half of what we need.

Advanced Wildcard Masks (continued)

To determine the wildcard mask, we perform a logical **XOR** (exclusive OR) operation:

1. If all bits in a column are set to **0**, the corresponding wildcard bit is **0**
2. If all bits in a column are set to **1,** the corresponding wildcard bit is **0**
3. If the bits in a column are a mix of **0**'s and **1's**, the corresponding wildcard bit is a 1**.**

Observe:

172.16.1.4:	10101100.00010000.00000001.00000100
172.16.1.5:	10101100.00010000.00000001.00000101
172.16.1.6:	10101100.00010000.00000001.00000110
172.16.1.7:	10101100.00010000.00000001.00000111
Result:	00000000.00000000.00000000.00000011

Our resulting wildcard mask is **0.0.0.3**. Put together, we have:

Router(config)# *access-list 10 deny 172.16.1.4 0.0.0.3*

Please Note: We can determine the number of addresses a wildcard mask will match by using a simple formula:

$$2^n$$

Where "n" is the number of bits set to **1** in the wildcard mask. In the above example, we have two bits set to 1, which matches exactly **four addresses** ($2^2 = 4$).

There *will* be occasions when we cannot match a range of addresses in one line. For example, if we wanted to deny 172.16.1.4-6, instead of 172.16.1.4-7, we would need two lines:

Router(config)# *access-list 10 permit 172.16.1.7 0.0.0.0*
Router(config)# *access-list 10 deny 172.16.1.4 0.0.0.3*

If we didn't include the first line, the second line would have denied the 172.16.1.7 address. Always remember to use the above formula (2^n) to ensure your wildcard mask doesn't match more addresses than you intended (often called overlap).

* * *

Advanced Wildcard Masks (continued)

Two more examples. How would we deny all **odd** addresses on the 10.1.1.x/24 subnet in one access-list line?

Router(config)# *access-list 10 deny 10.1.1.1 0.0.0.254*

Written in binary:

10.1.1.1:	00001010.00000001.00000001.00000001
Wild Card Mask:	00000000.00000000.00000000.11111110

What would the result of the above wildcard mask be?

1. The first three octets must match exactly.
2. The last bit in the fourth octet must match exactly. Because we set this bit to **1** in our address, every number this matches will be **odd**.
3. All other bits in the fourth octet can match any number.

Simple, right? How would we deny all **even** addresses on the 10.1.1.x/24 subnet in one access-list line?

Router(config)# *access-list 10 deny 10.1.1.0 0.0.0.254*

Written in binary:

10.1.1.0:	00001010.00000001.00000001.00000000
Wild Card Mask:	00000000.00000000.00000000.11111110

What would the result of the above wildcard mask be?

4. The first three octets must match exactly.
5. The last bit in the fourth octet must match exactly. Because we set this bit to **0** in our address, every number this matches will be **even**.
6. All other bits in the fourth octet can match any number.

* * *

Section 23
- DNS and DHCP -

Name Resolution

Name resolution systems provide the translation between alphanumeric *names* and numerical *addresses*, alleviating the need for users and administrators to memorize long strings of numbers.

There are two common methods for implementing name resolution:

- A **static file** on each host on the network, containing all the name-to-address translations (examples include the HOSTS/LMHOSTS files).
- A **centralized server** that all hosts on the network connect to for name resolution.

The two most common name resolution systems are **Domain Name System (DNS)** and **Windows Internet Name Service (WINS)**. WINS was used in Microsoft networks to translate IP addresses to NetBIOS names, and is mostly deprecated.

DNS is heavily utilized on the Internet and on systems such as Active Directory.

Domain Name System (DNS)

Domain Name System (DNS) translates between *domain names* and *IP addresses,* and is supported by nearly every operating system. All Internet-based name resolution utilizes DNS.

DNS is organized as a hierarchy. Consider the following translation:

www.google.com = 209.85.225.104

The above domain name represents a **Fully Qualified Domain Name (FQDN):**

- **.com** represents a top level domain.
- **.google** represents a secondary level domain
- **www** represents a host computer in the .google.com domain.

Other top level domains include **.org, .net,** and **.gov.** Top level domains can also include country codes, such as **.ca**, **.nl**, and **.de**

* * *

Methods of configuring DNS

Recall that DNS name resolution can be implemented in the form of local HOSTS files, or a centralized name server(s). When employing **HOSTS** files, each translation must be statically configured on each device. In Windows 2000/XP operating systems, this file is located:

c:\windows\system32\drivers\etc\hosts

In UNIX/Linux operating systems, this file is generally located: /etc/hosts

There are many disadvantages to using HOSTS files. The HOSTS file must be configured on every device. If a change occurs, every device's HOSTS file must be updated.

Using one or more **DNS servers** provides several advantages over HOSTS files. All devices point to this centralized DNS server for name resolution, ensuring that changes only need to occur in one place.

If a particular DNS server does not contain the required DNS information, the request will can be *forwarded* to servers up the DNS hierarchy.

BIND (Berkeley Internet Name Domain) is the standard implementation of DNS. Microsoft, UNIX/Linux, and Novell all employ some version of BIND.

DNS servers assume one of three roles:

- **Primary** (or **master) DNS Server -** maintains the **SOA (Start of Authority)**, and contains the master **zone file** containing the DNS records for the domain. This server is often referred to as the **Authoritative Name Server** for a specific domain.
- **Secondary** (or **slave) DNS Server** - maintains a current *copy* of the master zone file, obtained from the primary server. The secondary server cannot make changes to the zone file, but instead forwards changes to the primary server.
- **Caching DNS Server** - does not maintain a zone file, and is not authoritative for any domain. This server will merely cache the results of DNS queries.

Both hosts and DNS servers will cache the result of DNS queries for a period of time.

* * *

DNS Zone File Example

There are two types of **zones** in DNS:

- **Forward Lookup Zones -** translates a host name to an IP address.
- **Reverse Lookup Zones -** translates an IP address to a hostname (otherwise known as the IN-ADDR.ARPA zone).

The following is an example zone file for the fictional *example.com* domain:

```
$ORIGIN example.com
$TTL 86400
@     IN     SOA    dns1.example.com.
hostmaster.example.com. (
                     2001062501 ; serial
                     21600      ; refresh after 6 hours
                     3600       ; retry after 1 hour
                     604800     ; expire after 1 week
                     86400 )    ; minimum TTL of 1 day

      IN     NS     dns1.example.com.
      IN     NS     dns2.example.com.
      IN     MX     10     mail.example.com.
      IN     MX     20     mail2.example.com.

             IN     A       10.0.1.5
server1      IN     A       10.0.1.5
server2      IN     A       10.0.1.7
dns1         IN     A       10.0.1.2
mail         IN     CNAME   server1
mail2        IN     CNAME   server2
www          IN     CNAME   server2
```

Entries within a zone file are referred to as DNS **records.** There are a variety of DNS record types, including:

- **NS (Name Server)** – identifies a DNS server for the domain.
- **SOA (Start of Authority)** – identifies the primary (authoritative) DNS server for the domain.
- **A (Address)** – identifies an individual host in the domain.
- **CNAME (Canonical Name)** – assigns an alias for another host name.
- **MX (Mail Exchanger)** - identifies a mail server in the domain.
- **PTR (Pointer)** - used for *reverse* DNS lookups.

The number defined in the MX record is a *priority*. A *lower* priority is more preferred.

* * *

DNS Process

DNS follows a strict process when performing a query. The process is as follows:

1. The *local DNS cache* on the host is queried first.
2. If there is no entry in the local cache, the *local HOSTS file* is queried next.
3. If there is no entry in the local HOSTS, the query is forwarded to any *configured DNS servers* on the host. If no DNS servers are configured, the query will fail.
4. If the configured DNS server is not authoritative for that domain, and does not have that DNS entry locally cached, the query will be *forwarded* up the DNS hierarchy. DNS servers can be configured with one or more **forwarders.** Organizations often point to their ISP's DNS servers for DNS forwarding purposes.
5. If no forwarders are available, the query is forwarded to the *Root DNS server(s)*, which will likely have the entry cached.
6. In the rare circumstance that the Root servers do not have a cached entry, the query will be forwarded back down the hierarchy to the *authoritative DNS server* for that domain.

Dynamic DNS allows DNS to be integrated with Dynamic Host Configuration Protocol (DHCP). When DHCP hands out an IP address lease, it will automatically update the DNS entry for that host on the DNS server.

* * *

Resolving Hostnames on Cisco IOS Devices

There are two methods of name resolution on Cisco IOS devices:

- A static **host table** on each device (similar to a HOSTS file).
- A **centralized DNS server**(s) configured on each device.

To manually build a local host table on an IOS device:

Router(config)# *ip host Router1 172.16.1.1*
Router(config)# *ip host Router2 172.17.1.2*

To view the local host table:

Router# *show hosts*

To point an IOS device to a centralized DNS server:

Router(config)# *ip name-server 10.0.1.2*

To disable DNS lookups on an IOS device:

Router(config)# *no ip domain-lookup*

To configure the local domain on an IOS device:

Router(config)# *ip domain-name CISCO.COM*

* * *

DHCP (Dynamic Host Control Protocol)

In networks with a large number of hosts, statically assigning IP addresses and other IP information quickly becomes impractical.

Dynamic Host Control Protocol (DHCP) provides administrators with a mechanism to *dynamically* allocate IP addresses, rather than manually setting the address on each device.

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.

By default, DHCP leases an address for **8 days**. Once 50% of the lease expires, the client will try to renew the lease with the *same* DHCP server. If successful, the client receives a *new* 8 day lease.

If the renewal is *not* successful, the client will continue "attempting" to renew, until 87.5% of the lease has expired. Once this threshold has been reached, the client will attempt to find another DHCP server to bind to.

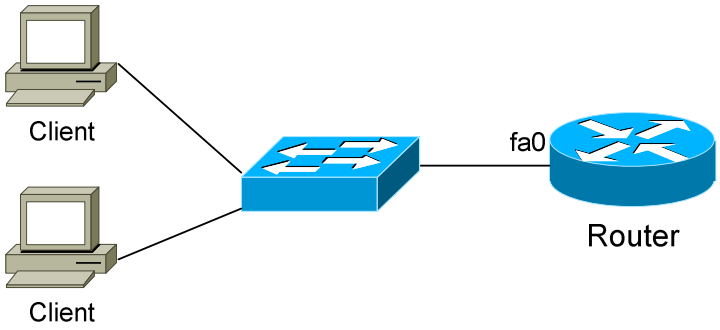
In addition to IP address and subnet mask information, DHCP can provide the following protocol parameters:

- Default Gateway
- Domain Name and DNS servers
- Time Servers
- WINS servers

These are just a few examples of the many DHCP "options" that exist.

* * *

Configuring a Cisco Router as a DHCP Server



Cisco routers can be configured to function as DHCP servers. The first step is to create a DHCP **pool**:

Router(config)# *ip dhcp pool MYPOOL*
Router(dhcp-config)# *network 192.168.1.0 255.255.255.0*

The first command creates a *dhcp pool* named *MYPOOL*. The second command creates our DHCP **scope**, indicating the range of addresses to be leased. The above command indicates any address between 192.168.1.1 – 192.168.1.255 can be leased.

Specific addresses can be excluded from being leased:

Router(config)# *ip dhcp excluded-address 192.168.1.1*
Router(config)# *ip dhcp excluded-address 192.168.1.5 192.168.1.10*

The first command excludes only address *192.168.1.1*. The second command excludes address *192.168.1.5* through *192.168.1.10*.

To specify DHCP options to be leased with the address:

Router(config)# *ip dhcp pool MYPOOL*
Router(dhcp-config)# *default-router 192.168.1.1*
Router(dhcp-config)# *dns-server 192.168.1.5*
Router(dhcp-config)# *domain-name MYDOMAIN*

To specify the duration of the DHCP lease:

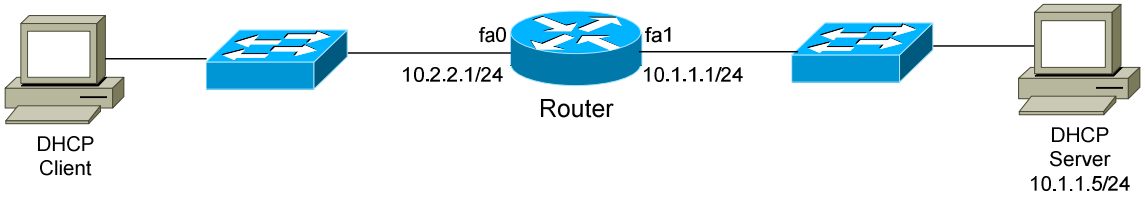
Router(config)# *ip dhcp pool MYPOOL*
Router(dhcp-config)# *lease 1 12*

The above changes the default *lease* from 8 days to *1* day, *12* hours. To view current DHCP leases:

Router# *show ip dhcp binding*

* * *

IP Helper Address



Recall that DHCP clients *broadcast* their **DHCPDiscover** packets, when searching for a DHCP server.

What would happen if the DHCP server is on a different network, separated from the clients by a router? Routers, by default, will never forward a broadcast.

Thus, in the above example, the client would never be able to reach the DHCP server to acquire its IP address. That is, unless the *ip helper-address* command is used:

Router(config)# *interface fa0*
Router(config-if)# *ip helper-address 10.1.1.5*

Notice that the *ip helper-address* command is configured on the interface connecting to the DHCP client, pointing to the IP address of the DHCP server. When the client broadcasts its DHCPDiscover packet, the router will direct that broadcast to the DHCP server. And there was much rejoicing.

By default, the *ip helper-address* command will forward the following UDP traffic:

- TFTP (port 69)
- DNS (port 53)
- Time (port 37)
- NetBIOS (ports 137-138)
- ND (Network Disks – used by Sun workstations)
- TACACS (port 49)
- BOOTP/DHCP (ports 67-68)

Customized UDP traffic can be specified using the following command:

Router(config)# *ip forward-protocol udp 107*
Router(config)# *no ip forward-protocol udp 69*

(Reference: http://www.cisco.com/en/US/products/sw/iosswrel/ps5207/products_command_reference_chapter09186a0080238b72.html#wp1182972)

* * *

Part V

WANs

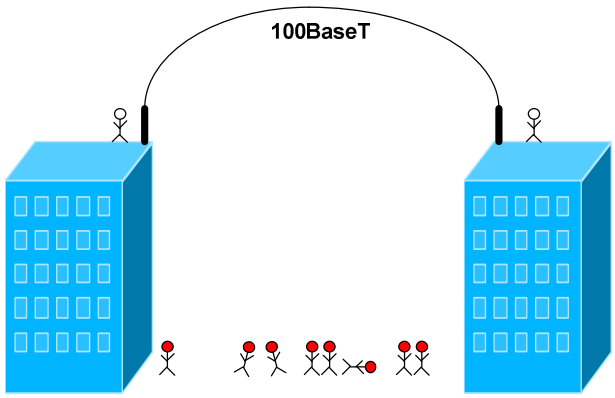
Section 24
- Basic WAN Concepts -

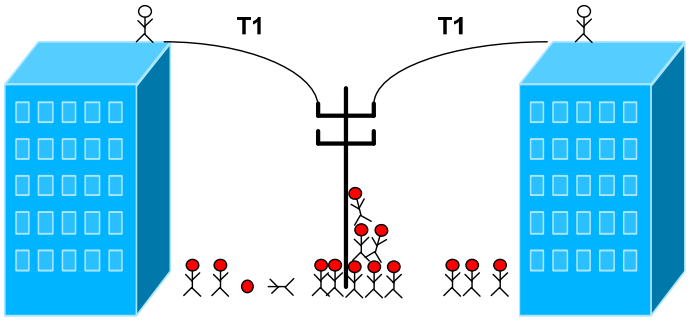
What is a WAN?

There are two prevailing definitions of a **Wide Area Network (WAN)**. The **book definition** of a WAN is a network that spans large geographical locations, usually to interconnect multiple Local Area Networks (LANs).

The **practical definition** of a WAN is a network that traverses a public network or commercial carrier, using one of several WAN **technologies.**

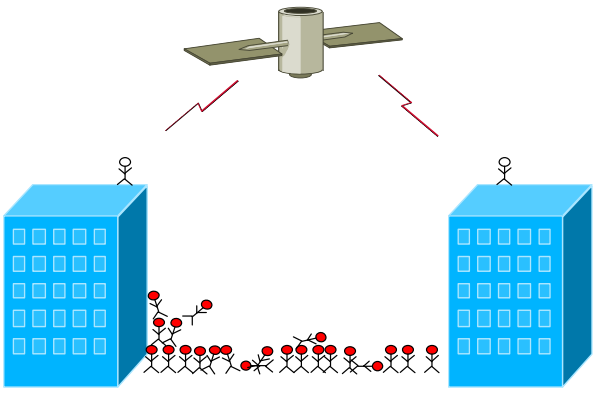
Consider the following example. A connection between two buildings using Ethernet as a medium would generally be considered a **LAN**. However, this is because of the *technology* used, and not the zombie-infested *distance* between the two buildings.





A connection between the *same* two buildings, using a dedicated T1 line as a medium, would generally be considered a **WAN**.

Remember, the difference is the *technology* used. A variety of WAN technologies exist, each operating at both the **Physical** and **Data-link** layers of the OSI models. Higher-layer protocols such as IP are **encapsulated** when sent across the WAN link.



* * *

WAN Connection Types

WANs are generally grouped into three separate **connection types**:

- Point-to-Point technologies
- Circuit-switched technologies
- Packet-switched technologies

Point-to-Point technologies (often called **dedicated** or **leased lines)** are usually the most expensive form of WAN technology. Point-to-Point technologies are *leased* from a service provider, and provide *guaranteed* bandwidth from location to another (hence point-to-point). Cost is determined by the *distance* of the connection, and the amount of *bandwidth* allocated.

Generally, point-to-point links require **no call-setup**, and the connection is usually **always on**. Examples of point-to-point technologies include:

- T1 lines
- T3 lines

Circuit-Switched technologies **require call-setup** to occur before information can be transferred. The session is usually torn down once data transfer is complete (this is identified as an **On-Demand Circuit**). Circuit-switched lines are generally low-speed compared to point-to-point lines.

Examples of circuit-switched technologies include:

- Dial-up
- ISDN

Packet-Switched technologies share a common infrastructure between all the provider's subscribers. Thus, bandwidth *is* not guaranteed, but is instead allocated on a **best effort** basis. Packet-switched technologies are ill-suited for applications that require consistent bandwidth, but are considerably less expensive than dedicated point-to-point lines.

Examples of packet-switched technologies include:

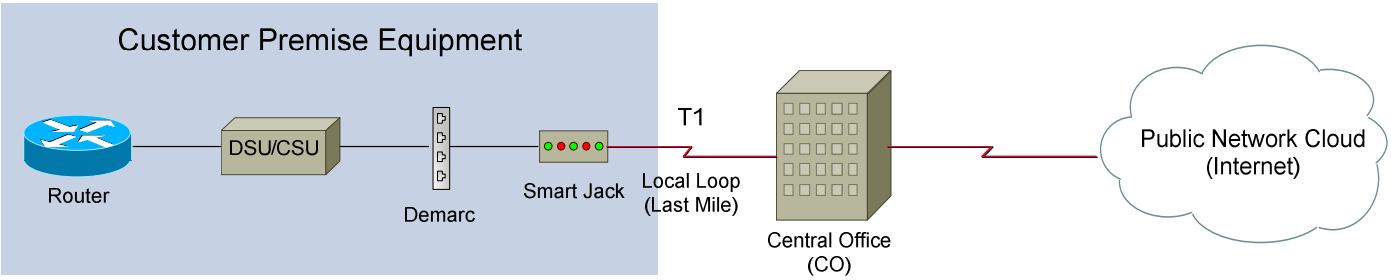
- Frame-Relay
- X25

(Reference: *http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/introwan.htm*
http://www.ciscopress.com/content/images/chap01_1587051486/elementLinks/1587051486content.pdf)

* * *

Common WAN Terms

A wide variety of hardware is used with WANs. Equipment that is housed at the subscriber is referred to as **Customer Premise Equipment (CPE).**



The above example demonstrates the basic equipment required for a T1 line.

A **CSU/DSU (Channelized Service Unit/Data Service Unit)** provides the clocking and channelization for T1 or T3 technology. The CSU/DSU converts the signal for use on an Ethernet (or other LAN technology) network. If a WAN technology other than a T1 line is used, a different device will be required. Examples include (but are no limited to):

- ISDN – a **terminal adapter**
- Dialup – **a modem**

The **Demarc** (short for demarcation) refers to the point of last responsibility for the service provider. All equipment on the Customer Premises side of the Demarc is the customer's responsibility to maintain. The Demarc is not *always* physically labeled or identifiable. Occasionally, a two-port or four-port patch-panel will be used as a physical Demarc.

The **Smart Jack** physically terminates the T1 line. If there is a connectivity issue, the provider will perform a ping test to the smart jack. If communication to the smart jack is successful, the provider will assume the issue resides on the customer's side of responsibility. The smart jack is often locked in a glass enclosure, and labeled with the T1's circuit number.

The **Local Loop** (or Last Mile) refers to the physical line connecting from the Customer Premises to the provider's nearest **Central Office (CO).**

WAN Encapsulation

Recall that WAN technologies operate at both **Physical** and **Data-link** layers of the OSI models, and that higher-layer protocols such as IP are **encapsulated** when sent across the WAN link.

A WAN is usually terminated on a Cisco device's serial interface. Serial interfaces support a wide variety of **WAN encapsulation types**, which must be manually specified.

By default, a serial interface will utilize **HDLC** for encapsulation. Other supported encapsulation types include:

- SDLC
- PPP
- LAPB
- Frame-Relay
- X.25
- ATM

Regardless of the WAN encapsulation used, it must **identical** on both sides of a point-to-point link.

Each encapsulation type is described in detail in separate guides.

* * *

Section 25
- PPP -

WAN Encapsulation

Recall that WAN technologies operate at both **Physical** and **Data-link** layers of the OSI models, and that higher-layer protocols such as IP are **encapsulated** when sent across the WAN link.

A WAN is usually terminated on a Cisco device's serial interface. Serial interfaces support a wide variety of **WAN encapsulation types**, which must be manually specified.

By default, a serial interface will utilize **HDLC** for encapsulation. Other supported encapsulation types include:

- SDLC
- PPP
- LAPB
- Frame-Relay
- X.25
- ATM

Regardless of the WAN encapsulation used, it must **identical** on both sides of a point-to-point link.

HDLC Encapsulation

High-Level Data-link Control (HDLC) is a WAN encapsulation protocol used on dedicated point-to-point serial lines.

Though HDLC is technically an ISO standard protocol, Cisco's implementation of HDLC is proprietary, and will not work with other routers.

HDLC is also Cisco's **default encapsulation** type for serial point-to-point links. HDLC provides *no* authentication mechanism.

* * *

PPP Encapsulation

Point-to-Point Protocol (PPP) is a standardized WAN encapsulation protocol that can be used on a wide variety of WAN technologies, including:

- Serial dedicated point-to-point lines
- Asynchronous dial-up (essentially dialup)
- ISDN

PPP has four components:

- **EIA/TIA-232-C** – standard for physical serial communication
- **HDLC** – for encapsulating packets into frames over serial lines
- **LCP** – for establishing, setting-up, and terminating point-to-point links
- **NCP** – allows multiple Layer-3 protocols (such as IP and IPX) to be encapsulated into frames

PPP supports several features that HDLC does not:

- Authentication
- Compression
- Multi-link
- Error Control

* * *

Configuring PPP

To configure a serial interface for PPP encapsulation:

Router(config)# *int s0/0*
Router(config-if)# *encapsulation ppp*

PPP supports two methods of authentication, **PAP** and **CHAP**. PAP (**Password Authentication Protocol**) sends passwords in clear text, and thus does not provide much security. CHAP (**Challenge Handshake Authentication Protocol**) uses MD5 to apply an irreversible hash.

To configure PPP authentication:

Router(config)# *hostname Router1*
Router(config)# *username Router2 password PASSWORD*

Router(config)# *int s0/0*
Router(config-if)# *ppp authentication chap*

The first line sets the hostname of the router. The second line sets the username and password used for PPP authentication. The username must be the hostname of the *remote* router, and the password must be the same on both routers.

The above configuration sets the authentication to *chap*. To instead configure *pap* authentication:

Router(config)# *int s0/0*
Router(config-if)# *ppp authentication pap*

To view the encapsulation configured on the interface:

Router# *show interface s0/0*

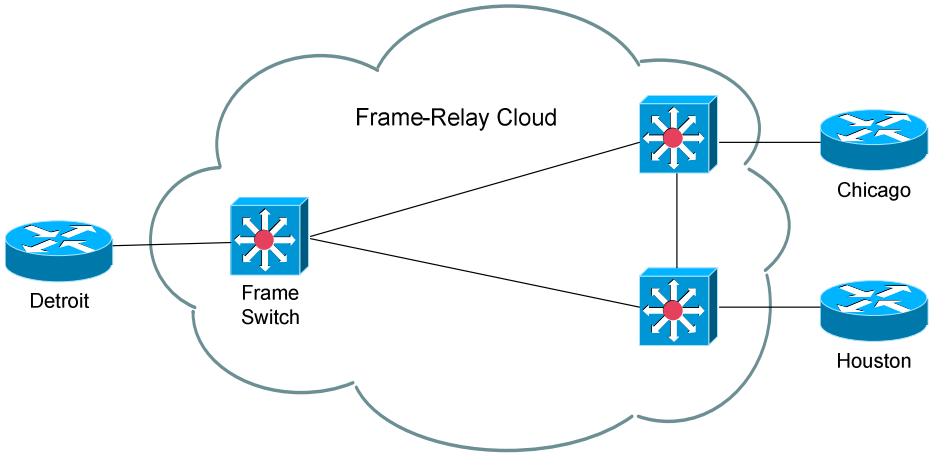
To troubleshoot PPP authentication between two routers:

Router# *debug ppp authentication*

* * *

Section 26
- Frame-Relay -

Frame-Relay



Frame-Relay is a packet-switched technology, which shares bandwidth between users on the switched network. Frame-relay service providers assume that all clients will *not* need the full capacity of their bandwidth at all times. Thus, in general, frame-relay is less expensive than dedicated WAN lines, but customers are not guaranteed bandwidth.

All locations plug into the frame relay "cloud," which is a conglomeration of dozens or hundreds of Frame-Relay switches and routers. The cloud is the Frame provider's network, and the customer has no control (or even knowledge) of what occurs inside that infrastructure.

For communication to occur between locations, v**irtual circuits (VC)** must be created. A VC is a one-way path through the Frame-Relay cloud.

In the above example, in order to establish full communication between Detroit and Houston, we would need to create two virtual circuits:

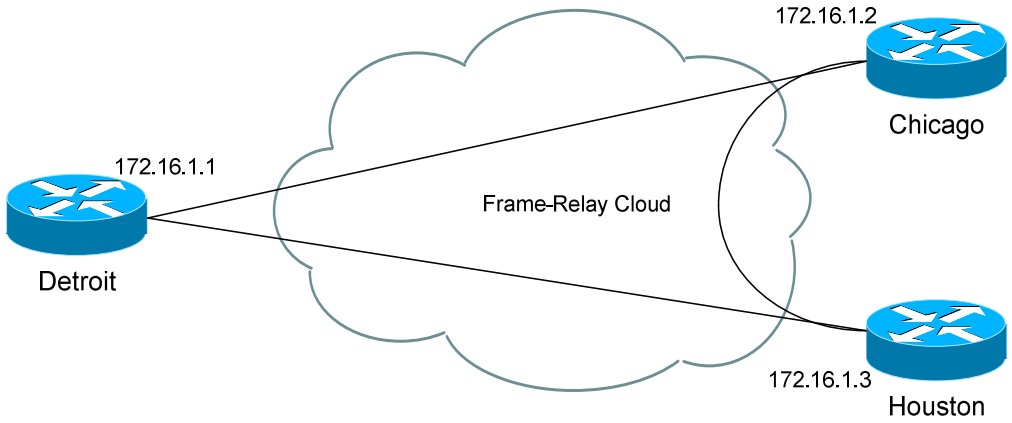
- A virtual circuit between Detroit and Houston
- A separate virtual circuit between Houston and Detroit

Frame-relay circuits can either be **permanent (PVC)**, or **switched (SVC).** A permanent virtual circuit is always kept active, and is the most common virtual circuit. A switched virtual circuit is created only when traffic needs to be sent, and is torn down when communication is complete.

Virtual circuits are identified with **Data Link Connection Identifiers (DLCIs).** Frame-Relay switches make decisions based on DLCIs, whereas Ethernet switches make decisions based on MAC addresses.

* * *

Frame-Relay Global vs Local DLCI



The difference between a globally or locally significant DLCI is all based on perspective. Remember that a DLCI identifies a one-way virtual circuit. For example, the connection between Detroit and Chicago would be considered one virtual circuit, and Chicago to Detroit would be a separate virtual circuit.

To get this to work, we need to map a DLCI to an IP address. For example, on router Detroit, we're going to create a virtual circuit to router Chicago. We'll assign it a DLCI of "102," and point it to Chicago's IP address.

We call this *locally* significant, because it only affects the interface on the Detroit router. We could, on the Chicago router, set a DLCI of "102" and point it to the IP address of the Detroit router. Because the DLCI is set on a different router (and interface), there will be no conflict.

When we set a *globally* significant DLCI, it is really only an administrative feature. It means that an administrator has consciously decided that all virtual circuits going to Chicago will be set to DLCI 102 (or whatever DLCI number you choose), whether it is from Detroit or Houston.

In essence, you are symbolically assigning the DLCI of 102 to the Chicago *location*. Keep in mind that you are still *technically* assigning the DLCI to the virtual circuits connecting to Chicago.

Virtual circuits pointing to other locations will be configured with *different* DLCIs (Detroit could be 101; Houston could be 103, etc.). The advantage to this is that it is now easy to determine the destination of a packet, based on its DLCI.

* * *

Frame-Relay CIR

Bandwidth is provided on a best effort basis in Frame-Relay.

The Frame provider and customer agree on a **Committed Information Rate (CIR),** which is not always a guarantee of bandwidth. The provider will give a best effort to meet the CIR, which is measured in bits per second:

- 256000 bps
- 512000 bps
- 1544000 bps

The above are examples of possible CIR settings, though technically the CIR can be set to anything. At times, bandwidth speeds can **burst (Be)** above the CIR. However, speeds above the CIR are certainly not guaranteed, and if the Frame Network becomes congested, any data exceeding the CIR becomes **Discard Eligible,** and is at risk of being dropped.

Frame-Relay Encapsulation Types

On Cisco routers, two possible Frame encapsulations can be configured on the router's serial ports.

- **Cisco –** the default, and proprietary, Frame-Relay encapsulation
- **IETF –** the standardized Frame-Relay encapsulation.

Frame-Relay Local Management Interface (LMI)

LMI is the type of signaling used between your router and your provider's Frame-Relay switch. LMI provides status updates of Virtual Circuits between the Frame switch and the router.

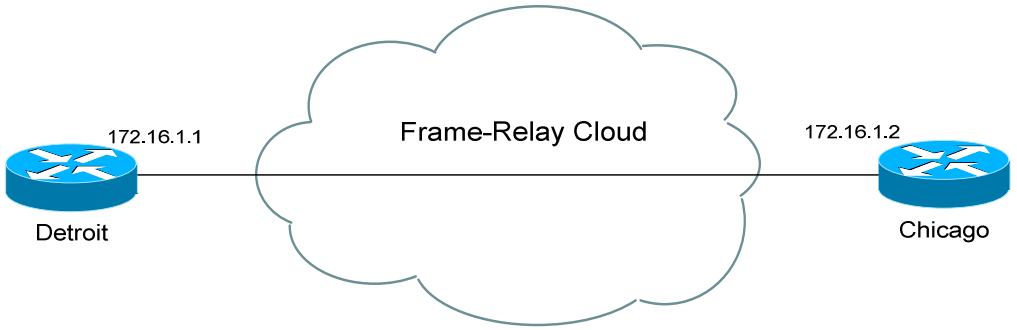
There are three LMI-types:

- **Cisco –** default and proprietary (naturally)
- **ANSI**
- **Q.933a**

LMI type is auto-sensed on Cisco routers, but can be manually set if desired.

* * *

Frame-Relay Point-to-Point Configuration Example



Point-to-Point is the simplest form of Frame-Relay configuration. Remember that PVCs are only one-way circuits, and thus we need to create *two* PVCs in order for full communication to occur.

Configuration on the Detroit and Chicago routers would be as follows:

Detroit Router:

Router(config)# *int s0/0*
Router(config-if)# *ip address 172.16.1.1 255.255.0.0*
Router(config-if)# *encapsulation frame-relay*
Router(config-if)# *frame-relay lmi-type q933a*
Router(config-if)# *frame-relay interface-dlci 102*
Router(config-if)# *no shut*

Chicago Router:

Router(config)# *int s0/0*
Router(config-if)# *ip address 172.16.1.2 255.255.0.0*
Router(config-if)# *encapsulation frame-relay*
Router(config-if)# *frame-relay lmi-type q933a*
Router(config-if)# *frame-relay interface-dlci 201*
Router(config-if)# *no shut*

Notice that both routers are in the same IP subnet.

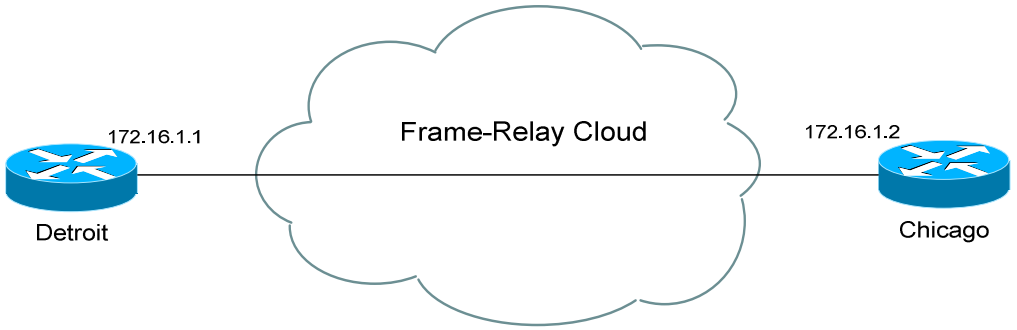
The *encapsulation frame-relay* command sets the frame encapsulation type to the default of *cisco*. The encapsulation must be the **same** on both routers. To change the default encapsulation type, simply append the *ietf* keyword to the *encapsulation frame-relay* command:

Router(config)# *int s0/0*
Router(config-if)# *ip address 172.16.1.1 255.255.0.0*
Router(config-if)# *encapsulation frame-relay ietf*

The *frame-relay lmi-type* command sets the signaling type. The Frame-Relay provider dictates which LMI-type to use. Remember that *cisco* is the default LMI-type, and that LMI is usually auto-sensed.

* * *

Frame-Relay Point-to-Point Configuration Example (continued)



Detroit Router:

Router(config)# *int s0/0*
Router(config-if)# *ip address 172.16.1.1 255.255.0.0*
Router(config-if)# *encapsulation frame-relay*
Router(config-if)# *frame-relay lmi-type q933a*
Router(config-if)# *frame-relay interface-dlci 102*
Router(config-if)# *no shut*

Chicago Router:

Router(config)# *int s0/0*
Router(config-if)# *ip address 172.16.1.2 255.255.0.0*
Router(config-if)# *encapsulation frame-relay*
Router(config-if)# *frame-relay lmi-type q933a*
Router(config-if)# *frame-relay interface-dlci 201*
Router(config-if)# *no shut*

The *frame-relay interface-dlci* command identifies the one-way PVC. The connection between Detroit and Chicago has been assigned DLCI 102. The connection between Chicago and Detroit has been assigned DLCI 201.

The Frame-Relay provider usually dictates which DLCI numbers to use, as the provider's Frame switch is configured with the appropriate DLCI information.

The router can actually receive all PVC and DLCI information directly from the Frame-Relay switch via LMI, using **Inverse-ARP.** Inverse-ARP is **enabled by default** on Cisco routers.

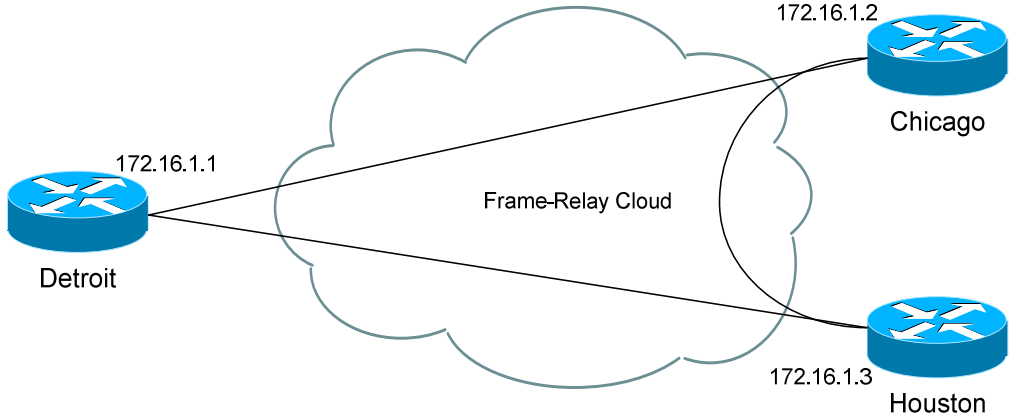
Thus, if the Frame-Relay switch is configured correctly, the *frame-relay interface-dlci* command could theoretically be removed, and the frame-relay connection will still work.

There are circumstances when DLCIs should be manually assigned. Inverse-ARP can be disabled on an interface with the following command:

Router(config)# *int s0/0*
Router(config-if)# *no frame-relay inverse-arp*

* * *

Frame-Relay Full Mesh Configuration Example



Consider the above example, a full mesh between three locations. All routers can still belong to the same IP subnet; however, DLCI's must now be *mapped* to IP addresses, as multiple PVCs are necessary on each interface.

This can be dynamically configured via **Inverse-Arp**, which is enabled by default (as stated earlier). Otherwise, the DLCI-to-IP mapping can be performed manually. Looking at the Detroit and Chicago router's configuration:

Detroit Router:

Router(config)# *int s0/0*
Router(config-if)# *ip address 172.16.1.1 255.255.0.0*
Router(config-if)# *encapsulation frame-relay ietf*
Router(config-if)# *no frame-relay inverse-arp*
Router(config-if)# *frame-relay lmi-type ansi*
Router(config-if)# *frame-relay map ip 172.16.1.2 102 broadcast*
Router(config-if)# *frame-relay map ip 172.16.1.3 103 broadcast*
Router(config-if)# *no shut*

Chicago Router:

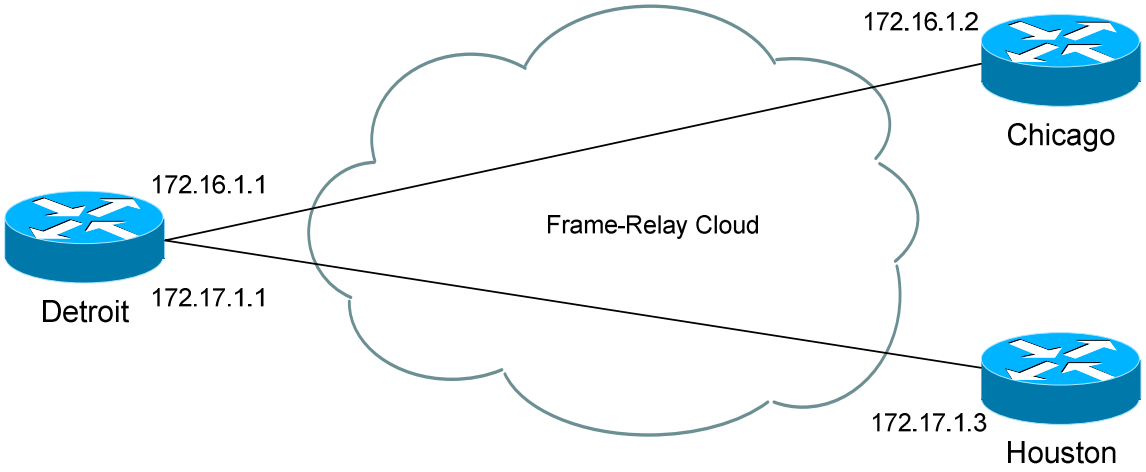
Router(config)# *int s0/0*
Router(config-if)# *ip address 172.16.1.2 255.255.0.0*
Router(config-if)# *encapsulation frame-relay ietf*
Router(config-if)# *no frame-relay inverse-arp*
Router(config-if)# *frame-relay lmi-type ansi*
Router(config-if)# *frame-relay map ip 172.16.1.1 201 broadcast*
Router(config-if)# *frame-relay map ip 172.16.1.3 203 broadcast*
Router(config-if)# *no shut*

Inverse-ARP was disabled using the *no frame-relay inverse-arp* command.

The *frame-relay map* command maps the remote router's IP address to a DLCI. On the Detroit router, a *map* was created to Chicago's IP (*172.16.1.2*), and that PVC was assigned a DLCI of *102*. The *broadcast* option allows broadcasts and multicasts to be forwarded to that address, so that routing protocols such as OSPF can form neighbor relationships.

* * *

Frame-Relay Partial Mesh Configuration Example



Full-mesh Frame-Relay environments can get quite expensive. Partial-mesh environments are often more cost-effective. A partial-mesh is essentially a hub-and-spoke design, with one **central** or **hub** location that all other locations must connect through.

In the above example, the Detroit router serves as the hub router. In a partial-mesh environment, each **spoke** must be on a different IP subnet, which presents a special problem.

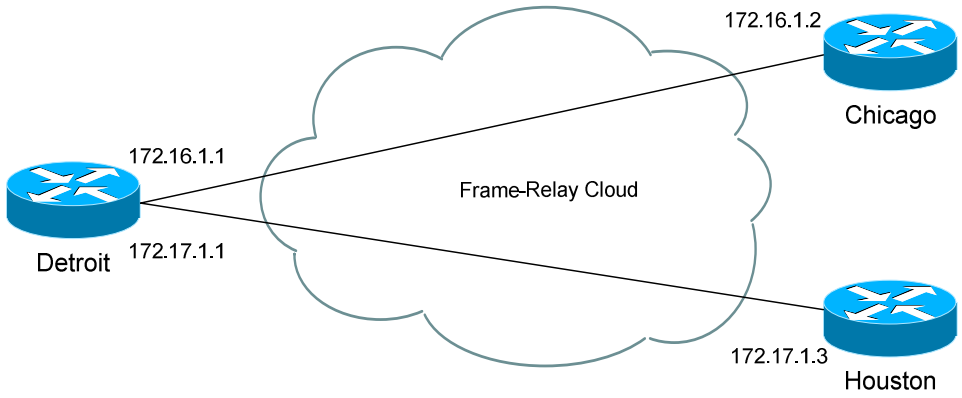
If both spokes terminate on the Detroit router's physical serial interface, **split-horizon** will prevent Chicago's routing updates from ever reaching Houston, and vice versa. Recall that split-horizon dictates that updates *received* on an interface cannot be *sent back out* the *same* interface.

Thus, on router Detroit, **sub-interfaces** must be created off of the serial interface. Sub-interfaces are **virtual** interfaces that the router treats as separate physical interfaces, providing a workaround for the split-horizon problem.

The **network type** must be specified when creating a sub-interface. A **point-to-point** sub-interface has only a single Virtual Circuit to another router. A **multipoint** sub-interface can have multiple Virtual Circuits to multiple locations.

* * *

Frame-Relay Partial Mesh Configuration Example (continued)



Configuration of the Detroit and Chicago routers would be as follows:

Detroit Router:

Router(config)# *int s0/0*
Router(config-if)# *encapsulation frame-relay*
Router(config-if)# *frame-relay lmi-type ansi*

Router(config)# *int s0/0.102 point-to-point*
Router(config-subif)# *no frame-relay inverse-arp*
Router(config-subif)# *ip address 172.16.1.1 255.255.0.0*
Router(config-subif)# *frame-relay interface-dlci 102*
Router(config-subif)# *no shut*

Router(config)# *int s0/0.103 point-to-point*
Router(config-subif)# *no frame-relay inverse-arp*
Router(config-subif)# *ip address 172.17.1.1 255.255.0.0*
Router(config-subif)# *frame-relay interface-dlci 103*
Router(config-subif)# *no shut*

Chicago Router:

Router(config)# *int s0/0*
Router(config-if)# *encapsulation frame-relay*
Router(config-if)# *frame-relay lmi-type ansi*

Router(config)# *int s0/0.201 point-to-point*
Router(config-subif)# *no frame-relay inverse-arp*
Router(config-subif)# *ip address 172.16.1.2 255.255.0.0*
Router(config-subif)# *frame-relay interface-dlci 201*
Router(config-subif)# *no shut*

Notice first that the Detroit router, serving as the hub, has two sub-interfaces configured pointing to Chicago and Houston. The Chicago router only has one sub-interface pointing to Detroit.

On the Detroit router, the *int s0/0.102* command creates a sub-interface numbered *102* on the Serial0/0 interface. Using the DLCI number for the sub-interface number is an arbitrary choice, useful for documentation purposes. On the Detroit router, each sub-interface contains only one virtual circuit, thus the interface's network type was set to *point-to-point*.

Notice also that encapsulation and LMI-type information is set on the physical *interface,* but IP address and DLCI information is set on the *sub-interface.*

* * *

Frame-Relay Traffic Shaping (FRTS)

Frame-Relay's method of QoS is called **traffic-shaping**, which controls the amount of traffic sent out an interface, and dictates congestion control mechanisms.

Frame-Relay Traffic-Shaping (FRTS) is used for two purposes:

- Adhering to the Frame provider's traffic rates.
- Preventing an oversubscription of the line between hub and spoke routers.

Several terms must be understood before configuring traffic-shaping:

- **Committed Information Rate (CIR) –** the "average" traffic rate provided on a best-effort basis. By default, the CIR on a serial interface configured for traffic shaping is **56000 bits per second**.
- **Available Rate (AR) –** the maximum traffic rate, dictated either by the speed of the physical interface (using the *clock rate* command), or the restrictions of the Frame Provider.
- **Minimum CIR (MinCIR)** – the minimum traffic rate the router will "throttle" down to if congestion occurs on the Frame-Relay network (i.e., a BECN is received). This is usually the provider's guaranteed traffic rate. By default, the MinCIR is *half* that of the CIR.
- **Discard Eligible (DE) –** a bit that is set for all traffic sent above the MinCIR. Essentially, traffic that is sent above the Frame Provider's guaranteed rate can or will be dropped when congestion occurs.
- **Committed Burst (Bc)** – the amount of bits sent during a specific interval, measured as **Time Committed (Tc).** Tc is measured in milliseconds (default is 125ms, or 8 intervals a second), and determines the number of intervals per second. The CIR is derived from the Bc and Tc using the following formula:
  CIR = Bc X 1000/Tc
- **Excess Burst (Be)** – the amount of bits that can be sent exceeding the Bc (or CIR). Any bits sent at this rate will be marked as DE.

* * *

Configuring Frame-Relay (FRTS)

To configure FRTS, a **map-class** must be created:

```
Router(config)# map-class frame-relay MYCLASS
Router(config-map-class)# frame-relay cir 64000
Router(config-map-class)# frame-relay bc 8000
Router(config-map-class)# frame-relay be 0
Router(config-map-class)# frame-relay mincir 32000
Router(config-map-class)# frame-relay adaptive-shaping becn
```

A *map-class* was created for *frame-relay* called *MYCLASS*. The first three commands configure the *CIR, Bc,* and *Be* respectively.

The final commands must be used in conjunction with each other. The *adaptive-shaping* feature has been specified, indicating that the router will throttle back to the *mincir* if a *becn* is received. The router does not throttle down to the *mincir* immediately, but rather will lower the rate by 25% until either the congestion stops, or the *mincir* is reached.

A map-class applied to an interface affects *all* PVCs on that interface. Additionally, map classes can be applied to a specific PVC, providing more granular control of FRTS.

To apply a map class to an interface:

```
Router(config)# interface s0/0
Router(config-if)# encapsulation frame-relay
Router(config-if)# frame-relay traffic-shaping
Router(config-if)# frame-relay class MYCLASS
```

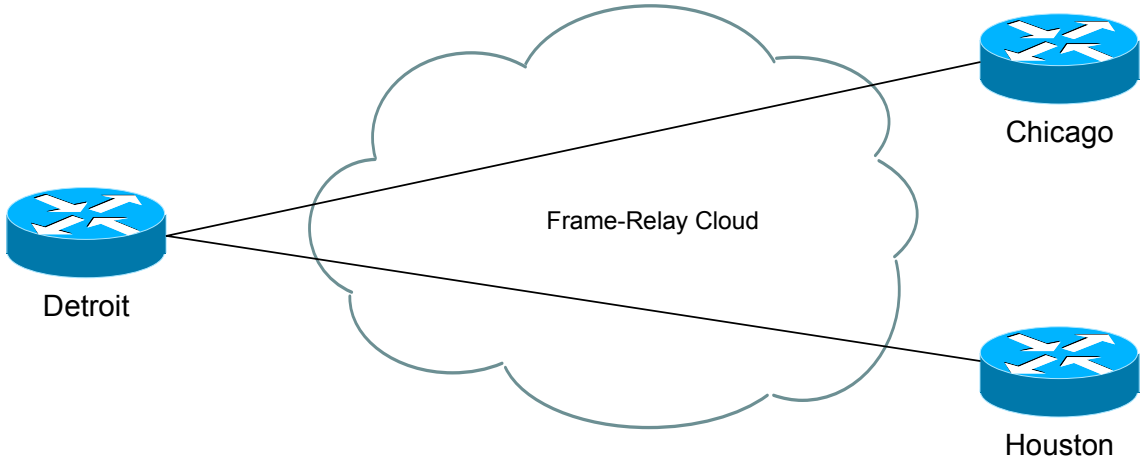
To apply a map class to a specific PVC:

```
Router(config)# interface s0/0
Router(config-if)# encapsulation frame-relay
Router(config-if)# frame-relay traffic-shaping
Router(config-if)# frame-relay interface-dlci 101 class MYCLASS
```

Do not forget the *frame-relay traffic-shaping* command. Once this command is configured, all PVCs are configured with the default CIR of 56,000 bps.

* * *

EIGRP and Frame-Relay



Observe the above Frame-Relay network. Two possible configuration options exist for the Detroit router:

- Configure frame-relay map statements on the physical interface
- Create separate sub-interfaces for each link, treating them as separate point-to-points.

If choosing the latter, EIGRP will treat each sub-interface as a separate link, and routing will occur with no issue.

If choosing the former, EIGRP will be faced with a split-horizon issue. Updates from Houston will not be forwarded to Chicago, and visa versa, as split horizon prevents an update from being sent out the link it was received on.

It is possible to *disable* split horizon for EIGRP:

Detroit(config)# *interface s0/0*
Detroit(config-router)# *no ip split-horizon eigrp 10*

Using sub-interfaces is Cisco's preferred method of circumventing the split-horizon issue, however.

* * *

Troubleshooting Frame-Relay

To view information concerning each PVC:

Router# *show frame-relay pvc*

The above command includes the following information:

- DLCI numbers
- Status of PVCs (active, inactive, deleted)
- Congestion information
- Traffic counters

To list Frame-Relay DLCI-mappings, whether manually created using the *frame-relay map* command, or created dynamically using Inverse ARP:

Router# *show frame-relay map*

To display the LMI-type configured on each interface, and LMI traffic statistics:

Router# *show frame-relay lmi*

To troubleshoot communication problems between the router and Frame-Relay switch:

Router# *debug frame-relay lmi*

To display information on packets *received* on a Frame-Relay interface:

Router# *debug frame-relay*

To display information on packets *sent* on a Frame-Relay interface:

Router# *debug frame-relay packet*

* * *

Section 27
- Network Address Translation -

NAT (Network Address Translation)

The rapid growth of the Internet resulted in a shortage of available IPv4 addresses. In response, a specific subset of the IPv4 address space was designated as *private,* to temporarily alleviate this problem.

A **public address** can be routed on the Internet. Thus, devices that must be Internet-accessible must be configured with (or *reachable* by) public addresses. Allocation of public addresses is governed by the Internet Assigned Numbers Authority (IANA).

A **private address** is intended for internal use within a home or organization, and can be freely used by anyone. However, private addresses can *never be routed* on the Internet. In fact, Internet routers are configured to immediately drop traffic with private addresses.

Three private address ranges were defined in RFC 1918, one for each IPv4 class:

- Class A - **10.x.x.x /8**
- Class B - **172.16.x.x /12**
- Class C - **192.168.x.x /24**

It is possible to *translate* between private and public addresses, using **Network Address Translation (NAT).** NAT allows a host configured with a private address to be *stamped* with a public address, thus allowing that host to communicate across the Internet. It is also possible to translate multiple privately-addressed hosts to a single public address, which conserves the public address space.

NAT provides an additional benefit – hiding the specific addresses and addressing structure of the internal (or *private*) network.

Note: NAT is *not* restricted to private-to-public address translation, though that is the most common application. NAT can also perform public-to-public address translation, as well as private-to-private address translation.

NAT is only a temporarily solution to the address shortage problem. IPv4 will eventually be replaced with IPv6, which supports a vast address space.

Both Cisco IOS devices and PIX/ASA firewalls support NAT.

* * *

Types of NAT

NAT can be implemented using one of three methods:

Static NAT – performs a static one-to-one translation between two addresses, or between a *port* on one address to a port on another address. Static NAT is most often used to assign a public address to a device behind a NAT-enabled firewall/router.

Dynamic NAT – utilizes a **pool** of global addresses to dynamically translate the outbound traffic of clients behind a NAT-enabled device.

NAT Overload or **Port Address Translation (PAT) –** translates the outbound traffic of clients to unique port numbers off of a *single* global address. PAT is necessary when the number of internal clients exceeds the available global addresses.

NAT Terminology

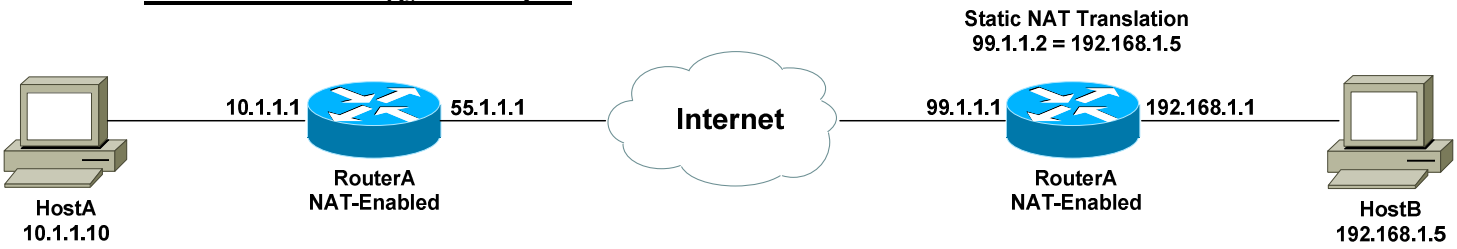
Specific terms are used to identify the various NAT addresses:

- **Inside Local –** the specific IP address assigned to an *inside* host behind a NAT-enabled device (usually a *private* address).
- **Inside Global –** the address that identifies an *inside* host to the *outside* world (usually a *public* address). Essentially, this is the dynamically or statically-assigned public address assigned to a private host.
- **Outside Global** – the address assigned to an *outside* host (usually a *public* address).
- **Outside Local** – the address that identifies an *outside* host to the *inside* network. Often, this is the **same** address as the Outside Global. However, it is occasionally necessary to translate an outside (usually *public*) address to an inside (usually *private*) address.

For simplicity sake, it is generally acceptable to associate **global** addresses with **public** addresses, and **local** addresses with **private** addresses. However, remember that public-to-public and private-to-private translation is still possible. **Inside** hosts are within the local network, while **outside** hosts are external to the local network.

* * *

NAT Terminology Example



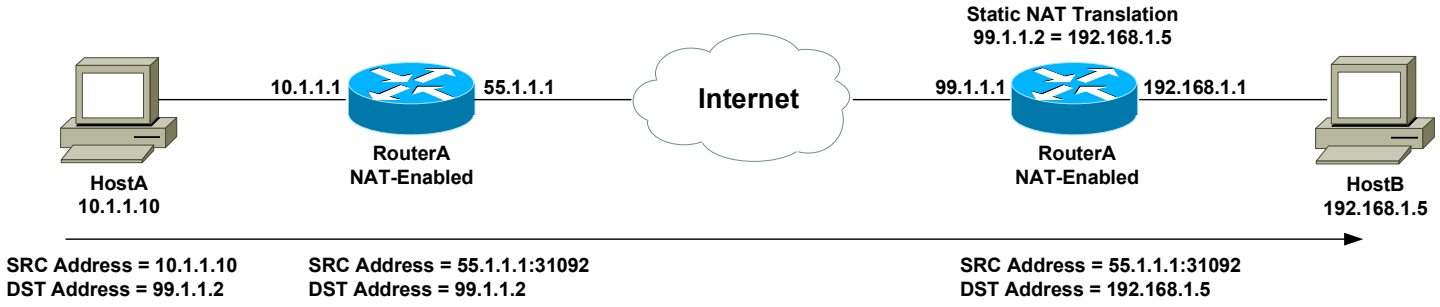
Consider the above example. For a connection *from* HostA *to* HostB, the NAT addresses are identified as follows:

- **Inside Local Address** - 10.1.1.10
- **Inside Global Address** - 55.1.1.1
- **Outside Global Address** – 99.1.1.2
- **Outside Local Address –** 99.1.1.2

HostA's configured address is *10.1.1.10*, and is identified as its *Inside Local* address. When HostA communicates with the Internet, it is stamped with RouterA's public address, using PAT. Thus, HostA's *Inside Global* address will become *55.1.1.1*.

When HostA communicates with HostB, it will access HostB's *Outside Global* address of *99.1.1.2*. In this instance, the *Outside Local* address is also *99.1.1.2*. HostA is never aware of HostB's configured address.

It is possible to map an address from the local network (such as 10.1.1.5) to the global address of the remote device (in this case, 99.1.1.2). This may be required if a legacy device exists that will only communicate with the local subnet. In this instance, the *Outside Local* address would be *10.1.1.5*.



The above example demonstrates how the source (SRC) and destination (DST) IP addresses within the Network-Layer header are translated by NAT.

(Reference: *http://www.cisco.com/warp/public/556/8.html*)

* * *

Configuring Static NAT

The first step to configure **Static NAT** is to identify the *inside* (usually private) and *outside* (usually public) interfaces:

Router(config)# *int e0/0*
Router(config-if)# *ip nat inside*

Router(config)# *int s0/0*
Router(config-if)# *ip nat outside*

To statically map a public address to a private address, the syntax is as follows:

Router(config)# *ip nat inside source static 172.16.1.1 158.80.1.40*

This command performs a *static* translation of the *source* address *172.16.1.1* (located on the *inside* of the network), to the outside address of *158.80.1.40.*

Configuring Dynamic NAT

When configuring **Dynamic NAT**, the *inside* and *outside* interfaces must first be identified:

Router(config)# *int e0/0*
Router(config-if)# *ip nat inside*

Router(config)# *int s0/0*
Router(config-if)# *ip nat outside*

Next, a *pool* of global addresses must be specified. Inside hosts will dynamically choose the next available address in this pool, when communicating outside the local network:

Router(config)# *ip nat pool POOLNAME 158.80.1.1 158.80.1.50 netmask 255.255.255.0*

The above command specifies that the *pool* named *POOLNAME* contains a range of public addresses from *158.80.1.1* through *158.80.1.50*.

Finally, a list of private addresses that are **allowed** to be dynamically translated must be specified:

Router(config)# *ip nat inside source list 10 pool POOLNAME*
Router(config)# *access-list 10 permit 172.16.1.0 0.0.0.255*

The first command states that any *inside* host with a *source* that matches access-*list 10* can be translated to any address in the *pool* named *POOLNAME*.

The *access-list* specifies any host on the *172.16.1.0* network.

* * *

Configuring NAT Overload (or PAT)

Recall that **NAT Overload** (or **PAT**) is necessary when the number of internal clients exceeds the available global addresses. Each internal host is translated to a unique port number off of a *single* global address.

Configuring NAT overload is relatively simple:

Router(config)# *int e0/0*
Router(config-if)# *ip nat inside*

Router(config)# *int s0/0*
Router(config-if)# *ip nat outside*

Router(config)# *ip nat inside source list 10 interface Serial0/0 overload*
Router(config)# *access-list 10 permit 172.16.1.0 0.0.0.255*

Any *inside* host with a *source* that matches access-*list 10* will be translated with *overload* to the IP address configured on the *Serial0/0* interface.

Troubleshooting NAT

To view all current static and dynamic translations:

Router# *show ip nat translations*

To view whether an interface is configure as an *inside* or *outside* NAT interface, and to display statistical information regarding active NAT translations:

Router# *show ip nat statistics*

To view NAT translations in real-time:

Router# *debug ip nat*

To clear all dynamic NAT entries from the translation table:

Router# *clear ip nat translation*

* * *