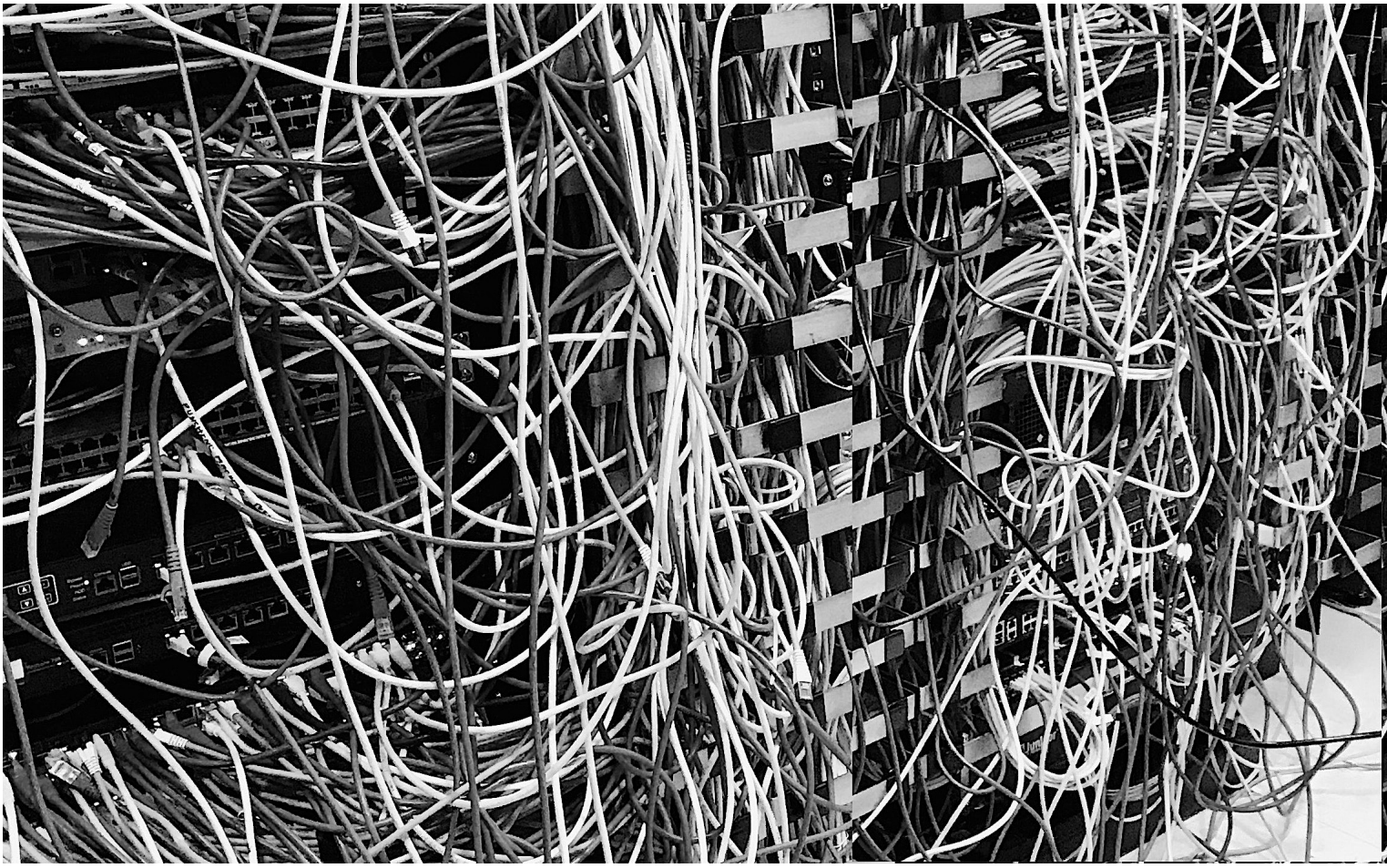
# Network Topics Compendium

## Omnibus Compilation of Primers, Reference, Refreshers, and Student Resources



***Compiled writings by Tristan Mendoza 2023***

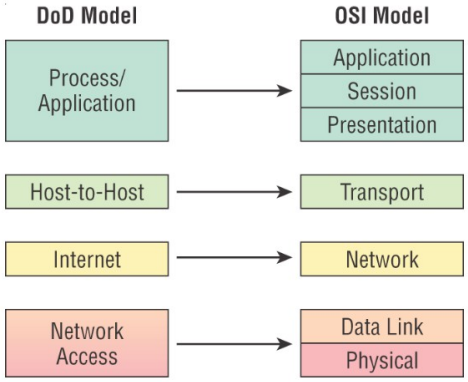


*This version missing topics that still need editing (in progress), such as WiFi technologies, SSL and IPSec VPNs, Cisco ASA NAT configuration, Zone-based firewall, plus command one-liners for Netcat, Nmap, Hping, and more.*

# *The OSI and DARPA TCP/IP Networking Models*

DEC/IBM (SNA) before 1980s, into 90s when TCP/IP prevailed and overtook the OSI model
The TCP/IP model is not a top-down comprehensive design reference - the purpose is illustrating the logical groups and scopes of functions needed. In general, direct or strict comparisons of the OSI and TCP/IP models should be avoided, because the layering in TCP/IP is not a principal design criterion. It just makes it easier to understand the interaction of the overlying technologies.

OSI Networking Model (DARPA's TCP/IP stack remains, but OSI won the terminology war)

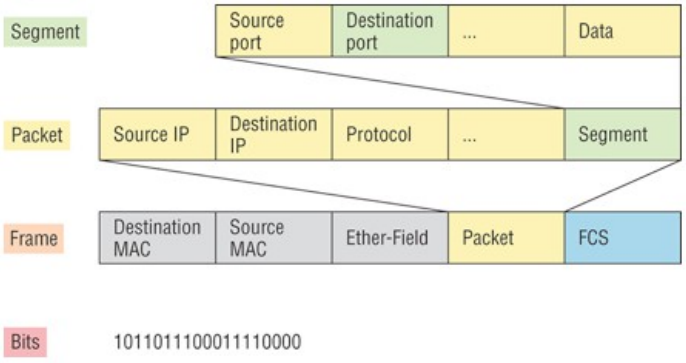



The DoD's Application layer is split into 3 layers in the OSI.
Transport is often still called Host-to-Host in the DoD model.
and it calls the network layer the Internet layer
Finally, the Data Link and Physical layers are unified in the DoD model, and called the Network Access layer, aka Network Interface, or Link layer.

The OSI has remained- as the network stack is always referred to by it's 7 layers. Additionally, layer 2 refers to data link, and layer 4 for transport. The DoD model retains layers 5, 6, and 7 as one application layer.

The TCP/IP and OSI stack are *reference models* of a *layered architecture.*

***Same-Layer Interactions*** *are between two computers at the same layer (like HTTP to HTTP)*
***Adjacent-Layer*** *- A layer passes info to neighboring layer (up or down) on the same computer*
A layer's data is referred as a Protocol Data Unit. A layer 2 frame is a L2PDU, a network IP packet is a L3PDU, and a UDP or TCP segment is a L4PDU.
An example of outgoing data, HTTP sends to TCP to put into a segment, it then sends to IP to pack up in a packet, and data-link frames it to go out the wire as bits. Likewise, when frame is pulled out of the incoming bits, it's frame is unpacked, the IP packet passed up the stack to be opened it up, revealing a segment to give to the adjacent transport layer.




Layer 7 - Application Layer - Telnet, HTTP, FTP, SMTP, POP3, VoIP, SNMP. Provides an interface between the communications software and any applications that need to communicate outside the computer on which the application resides. It also defines processes for user authentication

Layer 6 - Presentation Layer - ASCII, EBCDIC; de/compression, de/encryption. "Defines and negotiates data formats"; "how standard data should be formatted." Includes JPEG, TIFF, GIF, PICT, MPEG, MIDI, etc.

Layer 5 - Session Layer - PPTP, L2TP, NetBIOS, NFS, PAP, RPC, SOCKS. From a client to a server; three different modes: simplex, half-duplex, and full-duplex. How to start, control, and end conversations (sessions). This includes the control and management of multiple bidirectional messages so that the application can be notified if only some of a series of messages are completed. Sockets associate traffic with the transport layer's ports

Layer 4 - The Transport Layer - TCP, UDP, and SCTP - Determines how to handle data sent/ delivered over the network layer. Depending on the protocol, may provide for retransmission, i.e., error recovery, and may use flow control to prevent unnecessary congestion by attempting to send data at a rate that the network can accommodate. Multiplexing of incoming data for different flows to applications on the same host is also performed (ports and sessions). Reordering of the incoming data stream when packets arrive out of order is included.

The Network Layer (Layer 3) defines end-to-end delivery of packets and defines logical addressing to accomplish this. It also defines how routing works and how routes are learned; and how to fragment a packet into smaller packets to accommodate media with smaller maximum transmission unit sizes. Examples include. IP, IPX, and ICMP. Both IP and IPX define logical addressing, routing, the learning of routing information, and end-to-end delivery rules.

The Data-Link Layer (Layer 2) is concerned with getting data across one particular link or medium. The data link protocols define delivery across an individual link. These protocols are necessarily concerned with the type of media in use. Examples include. IEEE 802.3/802.2, HDLC, Frame Relay, PPP, FDDI, ATM. The lower level works with standards governing the physical transmission medium (Layer 1), including use of connector pins, electrical properties like voltage, etc.

The Physical Layer (Layer 1)
This is the medium: copper wire, fiber, or over the airwaves. This will be in an appendix of sorts to talk protocols sooner here.

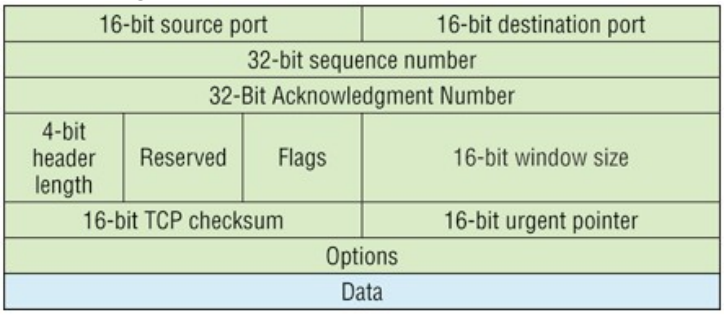
**Layer 4 - Transport Layer (aka Host-to-Host)** - end-to-end data transport services between hosts. TCP is a protocol with sequencing, acknowledgements- communications between the participating hosts. UDP is connectionless and there is no verification either side gets the other host's data.

TCP implements *reliable networking* - requiring acknowledgments, sequencing, and flow control will all be used. The 3-way handshake virtual circuit setup; removes a lot of programming work, but for real-time video and VoIP, UDP is often better because using it results in less overhead, but is less reliable)
Connection establishment and termination: SYN-->ACK-->SYN-ACK-->SYN and later, FIN-->FINACK?-->FIN!

*The types of flow control are buffering, windowing, and congestion avoidance.*
Sliding Window - TCP allows the receiving device to dictate the amount of data the sender can send before receiving an acknowledgment, the mechanism to grant future windows is typically just a number that grows (slides) upward slowly after each acknowledgment.

"Positive acknowledgment with retransmission" - The sender documents each segment measured in bytes, then sends and waits for acknowledgment before sending the next segment. The transmitting machine starts a timer and will retransmit if it expires before it gets an acknowledgment back from the receiving end.

**The TCP Segment's Format and Header Fields**




Source port: port number of the application on the host sending the data
Destination port: This is the port number of the application requested on the destination host.
Sequence number: Used to put the data back in the correct order (or retransmit missing or damaged data) during sequencing process.
Acknowledgment number: The value is the TCP octet that is expected next.
Header length aka Offset: Specifies the size of the TCP header in 32-bit words (a "word" is 4 bytes). The minimum is 5 words and the maximum is 15 words (so minimum is 20 bytes and maximum of 60 bytes, allowing for up to 40 bytes of options in the header)
Reserved: Always set to zero.
Code bits/flags: Control bits for functions to manage a session. (SYN, ACK, FIN, URG, PSH, etc)
Window: The window size the sender is willing to accept, in octets.
Checksum: This checksum results in combining data such as addresses involved, segment size and types with a formula that can be checked on the receiving end (it is recalculated in a misleadingly- labeled 'pseudoheader' to do it) - it isn't a CRC which checks all data, but it is sufficient.
Urgent: A valid field only if the Urgent pointer in the code bits is set. If so, this value indicates the offset from the current sequence number, in octets, where the segment of non-urgent data begins.
Options: May be 0, meaning that no options have to be present, or a multiple of 32 bits. However, if any options are used that do not cause the option field to total a multiple of 32 bits, padding of 0s must be used to make sure the data begins on a 32-bit boundary (aka "words").
Data: Handed down to the TCP PDU by the upper-layer headers

| | | | |
|---|---|---|---|
| TCP - Transport Control Protocol | | Code: | %011000 |
| Source Port: | 5973 | | Ack is valid, Push Request |
| Destination Port: | 23 | Window: | 61320 |
| Sequence Number: | 1456389907 | Checksum: | 0x61a6 |
| Ack Number: | 1242056456 | Urgent Pointer: | 0 |
| Offset: | 5 | No TCP Options | |
| Reserved: | %000000 | TCP Data Area: | vL.5.+.5.+.5.+.5 76 4c |

**The TCP handshake and termination - quick overview:**
The Initial Sequence and Response Numbers (ISN and IRN) are numbers exchanged in TCP segments during computer network communication between a client and a server. These are central in a SYN flood defense known as SYN cookies.

Here is a sample session:
1. The client sends a SYN with an ISN of 1664882716.
2. Server replies with a SYNACK with an IRN of 829007135 and an ACK value of 1664822717. The ACK reports the next the server expects from the client in this sequence (1664822717)
3. Client sends an ACK back 829007136 to increment the server's IRN in the SYNACK, which also reports it expects from the server in this sequence (829007136) It sends this with a sequence number of 1664882717, just like the server expects.

| | | | |
|---|---|---|---|
| 1. Client: | SYN | seq 1664882716 | |
| 2. Server: | SYNACK | seq 829007135 | ack 1664882717 |
| 3. Client: | ACK | seq 1664882717 | ack 829007136 |
| 4. Server: | ACK | seq 829007136 | ack 1664882718 |
| 5. Client: | ACK | seq 1664882718 | ack 829007137 |
| *Then later, server terminates the connection:* | | | |
| 1. Client: | ACK | seq 1664882733 | ack 829008199 |
| 2. Server: | FIN-ACK | seq 829008199 | ack 1664882734 |
| 3. Client: | ACK! | seq 1664882734 | ack 829008200 |
| 4. Client: | FIN-ACK? | seq 1664882734 | ack 829008200 (yes, two different responses) |
| 5. Server: | ACK | seq 1664882735 | |

**User Datagram Protocol** (**UDP**)
There are times that it's wise for developers to opt for UDP rather than TCP, one of them being when reliability is already taken care of at the upper layers. If the segments arrive out of order, which is commonplace in IP networks, they'll simply be passed up to the next layer in whatever order they were received, without sequencing or other features TCP provides. A UDP header is only 8 bytes (compared to TCP's 20 bytes) - has 4 fields, each of which are 2 bytes. Source and checksum are optional in IPv4 (only source is optional in IPv6)

UDP - User Datagram Protocol

| | |
|---|---|
| Source Port: | 1085 |
| Destination Port: | 5136 |
| Length: | 41 |
| Checksum: | 0x7a3c |
| UDP Data Area: | ..Z.....00 01 5a 96 00 01 00 00 00 00 00 11 0000 00 |

**Port numbers for communicating with upper layers**
Ordered data transfer and data segmentation. Ports with numbers 0-1023 are called well-known ports; ports with numbers 1024-49151 are called egistered ports, and ports with numbers 49152-65535 are called dynamic, private or ephemeral ports.*The source port number is arbitrary- usually ephemeral. Destination port is specific to process/application (workstation sends SSH connection out it's port using an ephemeral port #, to port 22 at the listening SSH process on the other end)* . The virtual circuit is defined by the source and destination port number plus the source and destination IP address and called a socket.

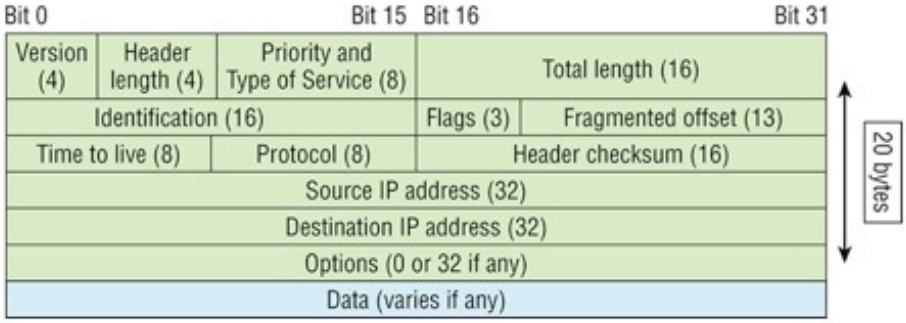
| | | | |
|---|---|---|---|
| 20-21 FTP | 110 POP3 | 465 SMTP over SSL | 587 SMTP |
| 22 SSH/SCP | 123 NTP | 500 ISAKMP | 636 LDAP over SSL |
| 23 Telnet | 135 Microsoft RPC | 512 rexec | 646 LDP (MPLS) |
| 25 SMTP | 137-139 NetBIOS | 513 rlogin | 860 iSCSI |
| 49 TACACS | 143 IMAP4 | 514 syslog | 902 VMware Server |
| 53 DNS (uses both) | 161-162 SNMP | 515 LPD/LPR | 989-990 FTP over SSL |
| 67-68 DHCP/BOOTP | 389 LDAP | 520 RIP | 993 IMAP4 over SSL |
| 69 TFTP | 443 HTTP over SSL | 521 RIPng (IPv6) | 995 POP3 over SSL |
| 80 HTTP | 445 Microsoft DS | 546-547 DHCPv6 | 1025 Microsoft RPC |
| 88 Kerberos | 464 Kerberos | | |

IANA assigns a port number for both TCP and UDP even if the service uses only one. Lists like the one above are ok, but often not very accurate (especially with VoIP). In Wireshark, it has a builtin tool for that.
Authoritative list here:
https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

**Layer 3 - Network/ Internet Layer** - IPv4, IPv6, IPX, IPSec, ICMP.

Has three main features: logical addressing, routing (forwarding), and path determination. Routing defines how devices (typically routers) forward packets to their final destination. Logical addressing defines how each device can have an address that can be used by the routing process. Path determination refers to the work done by routing protocols to learn all possible routes and determine the best way to move data. The network layer handles two types: data packets and route update packets. *Ultimately, Layer 3 doesn't care about where a particular host is located- only about where networks are located and the best way to reach them.*

IPv4 (below) has more on it in the following pages, in the form of a subnetting chart and worksheet on subnetting.




Header length: HLEN in 32-bit words.
Priority/TOS/Precedence: how to handle the datagram. The first 3 bits are the priority bits, called the differentiated services bits.
Total length: including header and data.
Identification: Unique IP-packet value used to differentiate fragmented packets from different datagrams.
Fragmentation Flags: Specifies whether fragmentation should occur.
Fragment offset: Provides fragmentation and reassembly if the packet is too large to put in a frame. It accommodates different MTUs on the Internet.
Time To Live: TTL set when packet is generated. If it doesn't get to where it's supposed to go before the TTL expires, it's dropped.
Protocol: Port of ecapsulated payload protocol; for example, TCP is port 6 or UDP is port 17
Header checksum: CRC on header only.
Source IP address: 32-bit IP address of sending station.
Destination IP address: 32-bit IP address of the station this packet is destined for.
Options: Used for network testing, debugging, security, and more.
Data: The upper-layer data.

IP Header - Internet Protocol Datagram

| | |
|---|---|
| Version: | 4 |
| Header Length: | 5 |
| Precedence: | 0 |
| Type of Service: | %000 |
| Unused: | %00 |
| Total Length: | 187 |
| Identifier: | 22486 |

| | |
|---|---|
| Fragmentation Flags: | %010 Do Not Fragment |
| Fragment Offset: | 0 |
| Time To Live: | 60 |
| IP Type: | 0x06 TCP |
| Header Checksum: | 0xd031 |
| Source IP Address: | 10.7.1.30 |
| Dest. IP Address: | 10.7.1.10 |
| No Internet Datagram Options | |

Protocols found in the protocol field of an IP header
ICMP 1, TCP 6, UDP 17, IPv6 41, GRE 47, EIGRP 88, OSPF 89, L2TP 115
For more: http://www.iana.org/assignments/protocol-numbers

- IP packets in WiFi have a noticeably larger size, which is mainly to hold not only the source and destination IP addresses of the workstation, but also the IP addresses of the access points on either end. We may get into that more in the section on WiFi.

- IPSec is it's own big topic which includes it's mechanism and tunneling uses. This also will have it's own dedicated area when we talk about VPN.

- IPv6 is saved toward the end of the routing and switching material, since it has a lot to talk about those things and the differences in IPv6 v.s. IPv4 (not much but it keeps things simpler to explain everything in IPv4 terms first instead of complicating it for people that might be new to all this).

**ICMP**
As it's name implies, this is a simple general messaging protocol, mostly about traffic stuff.

IANA maintains lists of ICMPv4 and ICMPv6 message types:
*http://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml*
*http://www.iana.org/assignments/icmpv6-parameters/icmpv6-parameters.xhtml*

From a network scanning perspective, the following types of ICMP messages are useful:
Type 8 (echo request) aka ping packets. Perform ping sweeping and identify accessible hosts. Type 0 is the reply.
Type 13 (timestamp request) get system time from host; in decimal, number of milliseconds since midnight GMT.

**Common ICMP type 3 message (destination unreachable) codes**
Firewalls and routers generate type 3 ICMP responses, providing insight into network configuration.
0 - Network unreachable
1 - Host unreachable
2 - Protocol unreachable
3 - Port unreachable
6 - Destination network unknown
7 - Destination host unknown
9 - Communication administratively prohibited (network)
10 - Communication administratively prohibited (host)
13 - Communication administratively prohibited (general)
*Cisco exams used to like mentioning "buffer full/source quench (deprecated Type 3 msg 4)- router's buffer for incoming datagrams is full.*

There is more on ICMP in the section about the tools PING and traceroute, since those also discuss TCP.

ICMP error messages include a copy of the IPv4 header, plus at least the first 8 bytes of data from the error-triggering frame The length of ICMP error messages should not exceed 576 bytes. It is also intended to hold the type field indicating the upper layers contained within.

The ICMP data field can be exploited, as in the "Ping of death", large or fragmented ICMP packets are used for denial-of-service attacks. ICMP data can also be used to create covert tunnels for communication. There are more specific uses- encapsulation of IP packets into ICMP packets, leads to TCP session inclusion which opens the door for creative applications - there is also a telnet-like shell that can be opened.

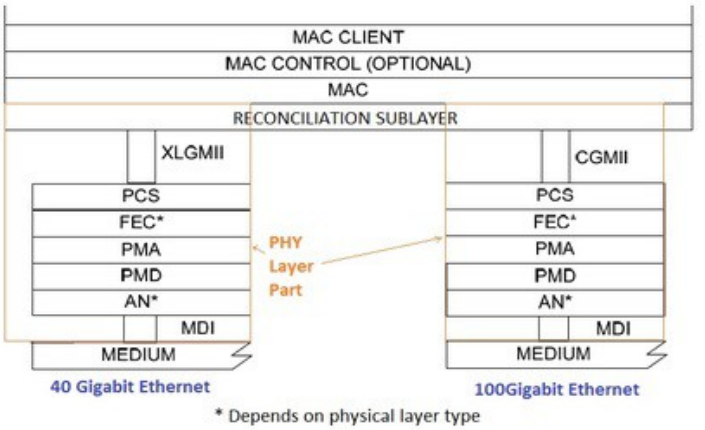
**Layer 2 - Data Link Layer**
ARP, MAC (IEEE 802.3 Ethernet, DSL, ISDN, FDDI), PPP, L2TP, HDLC
Defines the rules that determine when a device can send data over a particular medium. Data link protocols also define the format of a header and trailer that allows devices attached to the medium to successfully send and receive data.

This layer transfers data between adjacent network nodes in a wide area network (WAN) or between nodes on the same local area network (LAN) segment. Frames do not cross the boundaries of a local network (these frames are encapsulated/replaced by new frames native to the WAN environment). Internetwork routing and global addressing are higher-layer functions, allowing data-link protocols to focus on local delivery, addressing, and media arbitration (between parties contending for access to a medium, without concern for their ultimate destination). When devices attempt to use a medium simultaneously, frame collisions can occur. Data-link protocols specify how devices detect and recover from such collisions, and often provide mechanisms to reduce or prevent them.

Logical link control sublayer
The uppermost sublayer, LLC, multiplexes protocols running atop the data link layer, and optionally provides flow control, acknowledgment, and error notification. The LLC provides addressing and control of the logical links between local devices: which mechanisms are to be used for addressing stations over the transmission medium and for controlling the data exchanged between the originator and recipient machines. It basically provides services to the network layer and hides the rest of the data link layer. Is outlined by IEEE 802.2 - https://en.wikipedia.org/wiki/IEEE_802.2

Media access control sublayer
The sublayer defines methods to control access to the physical layer. Since many networks use a shared medium (such as a single network cable, or a series of cables that are electrically connected into a single virtual medium) it is necessary to have rules for managing the medium to avoid conflicts. Examples include ethernet's CSMA/CD method of media access control, while Token Ring uses token passing. It also determines where one frame of data ends and the next one starts- frame synchronization. There are four means of frame synchronization: time based, character counting, byte stuffing and bit stuffing.

MAC Services: physical addressing, channel-access control (CSMA/CD and CSMA/CA), LAN switching (packet switching), including MAC filtering, Spanning Tree Protocol and Shortest Path Bridging (SPB), data packet queuing or scheduling, Store-and-forward switching or cut-through switching, Quality of Service (QoS) control, encoding schemes. It is outlined by IEEE 802.3 - See https://en.wikipedia.org/wiki/IEEE_802.3



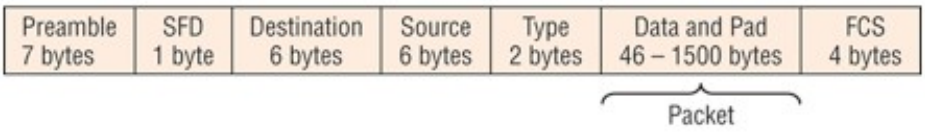
IEEE 802.3ba architecture

**Network engineers have it easy**
The MAC sublayer of Layer 2 almost has it's own huge protocol stack inside it, dealing with the gritty details of electrical signal properties of pins on connectors, light modulation for fiber optics, things we never have to think about with the physical medium.

Thankfully, most network engineers don't have to deal with this. This diagram is tame compared to some I have seen. This is the MAC layer substack for gigabit ethernet. We only have to think about the very top one layer, "MAC client".

Most networking classes never mention this- it is just "layer two 2nd half, then layer 1, da wire". I guess they think it will make their student's head explode or something.

**Layer 2 Protocols - Ethernet Frames at the Data Link Layer**

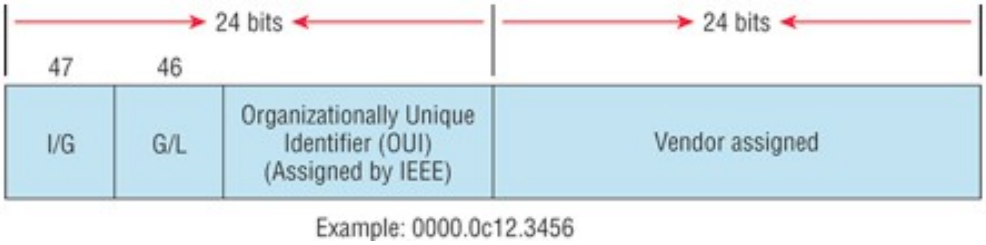



Preamble - 7 bytes - pattern of alternating 1 and 0 bits, allowing network devices to easily synchronize their receiver clocks. Since least significant bits are transmitted first, the one breaking the alternating pattern tends to be last.
Start Frame Delimiter (SFD): 1 byte - The SFD octet is designed to break the bit pattern of the preamble and signal the start of the actual frame. Arguably, the SFD could more easily be called part of the preamble, and often you will just see 8 bytes: preamble. Wireshark won't see this (L2 strips it off)
Destination Address (DA) - 6 bytes - The 48-bit value of the intended recipient. Can also be BCast or MCast.
Source Address (SA) - 6 bytes - Sender. BC and MC address formats are illegal within the SA field.
802.1Q tag (optional) - 4 bytes - Ethernet II only
Length or Type - 2 bytes - Old 802.3 uses a Length field, but Ethernet_II frame uses a Type field to ID the layer 3 PDU. 0x86dd for IPv6 data; 0x0800 for IPv4; 0x8100 for VLAN-tagged (802.1Q); 0x0806 for ARP; 0x8147/48 MPLS; 0x88CC LLDP; 0x8163/64 PPPoE; 0x8906 FCoE; 0x8100 Jumbo;
Data - 46-1500 bytes - This is a packet sent down to the Data Link layer from the Network layer. The size can vary from 46 to 1,500 bytes. (44 if 802.1Q tag is not present, and greater than 1500 if jumbo). Padding to meet the minimum 46 bytes is added if necessary.
Frame Check Sequence (FCS/CRC) - Algorithm is run when each frame is built based on the data in the frame. If CRCs don't match, the frame is discarded, assuming errors have occurred.

The original IEEE 802.3 defined the min/max Ethernet frame size as 64/1518 bytes (max later increased to 1522 bytes for VLAN tagging. MTU in an Ethernet LAN 1500 bytes by default.
The minimum size of an Ethernet frame that carries an ICMP packet is 74 bytes. (ping packet with no options will generate a 74 byte packet with a 60 byte IP Header, 8 byte ICMP header
Minimum 64 bytes (header + data + FCS) -- 64-(14+4)=46 bytes for data, padded if needed

Size of Ethernet frame - 24 Bytes - [this doesn't include preamble 8 bytes]
Size of IPv4 Header (without any options) - 20 bytes
Size of TCP Header (without any options) - 20 Bytes
So total size of empty TCP datagram - 24 + 20 + 20 = 64 bytes
Size of UDP header - 8 bytes
So total size of empty UDP datagram - 24 + 20 + 8 = 52 bytes
Ethernet HW usually filters preamble and FCS on incoming packets (so applications won't see them- Wireshark)

**Data Link and Ethernet - Hex to Dec and MAC Addresses**




Example: 0000.0c12.3456

48-bit (6-byte) MAC address written in a hexadecimal format (instead of binary for readability).
The organizationally unique identifier (OUI) is assigned by the IEEE to an organization. It's composed of 24 bits, or 3 bytes, and it in turn assigns a globally administered address

Individual/Group (I/G) bit. When it has a value of 0, we can assume that the address is the MAC address of a device and that it may well appear in the source portion of the MAC header. When it's a 1, we can assume that the address represents either a broadcast or multicast address in Ethernet."

Global/local bit, sometimes called the G/L bit or U/L bit, where U means universal. When set to 0, this bit represents a globally administered address, as assigned by the IEEE, but when it's a 1, it represents a locally governed and administered address. This becomes clearer when discussing IPv6 and MAC addresses.

Manufacturer-assigned code commonly starts with 24 0s for the first card made and continues to 16,777,216 (24 ones in binary). It is often incorporated into the serial number as well.
**Hex to Binary conversion: A "nibble" is 4 bits and a byte is 8 bits (or an octet)**
If we have a 1 placed in each spot of our nibble, we would then add up 8 + 4 + 2 + 1 to give us a maximum value of 15. Another example for our nibble values would be 1001, meaning that the 8 bit and the 1 bit are turned on, which equals a decimal value of 9. If we have a nibble binary value of 0110, then our decimal value would be 6, because the 4 and 2 bits are turned on. You take these two nibbles as binary and run them together as a byte: you get the real value 10010110, which is 128+16+2+4 = 150

Example: 0x6A- each hex character is one nibble and that two hex characters joined together make a byte. To figure out the binary value, put the hex characters into two nibbles and then join them together into a byte. 6 = 0110; A, which is 10 in hex = 1010; so the complete binary byte would be 01101010, and 64+32+8+2= 106 is the decimal value.

A binary number: 11001100. What's it in hex? Split it: 1100 = 12 and 1100 = 12, so therefore, it's representation is CC in hex, but the decimal conversion would be 128 + 64 + 8 + 4 = 204, the real value.
What about 10110101? The hex answer would be 0xB5, since 1011 converts to B and 0101 converts to 5 in hex value. The decimal equivalent is 128 + 32 + 16 + 4 + 1 = 181

| **Hex** | **Bin** | **Dec** | 5 | 0101 | 5 | B | 1011 | 11 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0000 | 0 | 6 | 0110 | 6 | C | 1100 | 12 |
| 1 | 0001 | 1 | 7 | 0111 | 7 | D | 1101 | 13 |
| 2 | 0010 | 2 | 8 | 1000 | 8 | E | 1110 | 14 |
| 3 | 0011 | 3 | 9 | 1001 | 9 | F | 1111 | 15 |
| 4 | 0100 | 4 | A | 1010 | 10 | | | |

**Layer 2 Protocols - Address Resolution Protocol (ARP)**
ARP was defined by RFC 826 in 1982, and is used for mapping a IPv4 address to a physical MAC address. In IPv6 the functionality of ARP is provided by the Neighbor Discovery Protocol (NDP). When IP has a datagram to send, it must inform a Network Access protocol, such as Ethernet, of the destination's hardware address on the local network. If IP doesn't find the destination host's hardware address in the ARP cache, it uses ARP to find this information.

A continuing discussion about ARP is where Layer 2 and 3 begin and end, since it seems to almost straddle both. It is a good reminder that the OSI model is just that- a conceptual model for design and development of protocols. This is a good example (there aren't many) of where the real world is a bit less tangible and technological requirements move things beyond the roadmap given to us. In the end it is best to call it L2 and be done with it- it is most likely that the LLC invokes service calls to the MAC sublayer in order to report to the CAM table and layer 3 services

ARP - Address Resolution Protocol

| | |
|---|---|
| Hardware | 1 (Ethernet 10Mb) |
| Protocol: | 0x0800 (IP) |
| Hardware Address Length: | 6 |
| Protocol Address Length: | 4 |
| Operation: | 1 ARP Request |
| Sender Hardware Address: | 00:A0:24:48:60:A5 |
| Sender Internet Address: | 172.16.10.3 |
| Target Hardware Address: | 00:00:00:00:00:00 (ignored) |
| Target Internet Address: | 172.16.10.10 |

ARP related terms, (or, terms with ARP in the name)

*Inverse ARP* is primarily used in Frame Relay (DLCI) and ATM networks, which need the corresponding Layer 3 addresses before those virtual circuits can be used

*An ARP probe* is an ARP request constructed with an all-zero sender IP address. Before beginning to use an IPv4 address a host must test to see if the address is already in use, by broadcasting ARP probe packets

A *gratuitous ARP* announcement updates any cached entries in the ARP tables of other hosts that receive the packet. Many operating systems perform gratuitous ARP during startup. It helps in case a network card was recently changed and other hosts need to update ARP caches. Gratuitous ARP is also needed in teaming network cards, and can be used to defend link-local IP addresses in zeroconf.

*ARP spoofing* - ARP has no authentication, so replies can come from systems other than the one with the required address. ARP spoofing is where a system impersonates another system's address with the aim of intercepting data bound for that system, resulting in man-in-the-middle and DoS attacks. An ARP proxy is a legitimate system which answers the ARP request on behalf of another system for which it will forward traffic, such as for a dialup internet service.

*ARP mediation* is WAN resolution of Layer 2 addresses through a Virtual Private Wire Service (VPWS) when different resolution protocols are used circuits- e.g., Ethernet on one end and Frame Relay on the other. In IPv4, each Provider Edge (PE) device discovers the IP address of the locally attached Customer Edge (CE) device and distributes that IP address to the corresponding remote PE device. Then each PE device responds to local ARP requests using the IP address of the remote CE device and the hardware address of the local PE device.

In IPv6, ARP is replaced with NDP (Network Discovery Protocol) - each device discovers the IP address of both local and remote CE devices and then intercepts local Neighbor Discovery (ND) and Inverse Neighbor Discovery (IND) packets and forwards them to the remote PE device.

**Layer 2 - Switching basic decision-making, other issues**
IEEE 802.1D/w Spanning Tree puts each port in forward or blocking state. Ports in a blocking/ discarding state won't process any frames except STP messages - the switch physically receives the frame on blocked port, but ignores it.

*That being said, for all ports in a forwarding state:*
If destination address is:
*- same as source? Ignore (filter).*
*- known? Forward to correct port*
*- unknown? LEARN: map the source MAC address to it's port number. Flood out all ports except entry*

**Address learning:**
- When flooded out, source MAC is added to table, but the destination will NOT be learned.
*For each received frame, examine the source MAC, note the interface.*
- If not in table, add- *set inactivity timer to 0.*
- If it is in table, *reset the inactivity timer for the entry to 0.*
On new frame, if MAC-to-port is different than previously recorded, it will be updated.
*- MAC table instability (info repeatedly updated erroneously from loops) is prevented by STP*

**Modes of Layer 2 forwarding:**
Store-and-forward - receive all bits in the frame and check the FCS before forwarding.
Fragment-free - receiving the first 64 bytes first to weed out collision-damaged frames.
Cut-through - checks dest MAC, forwards frame ASAP -reduces latency but no FCS check (later versions check for QoS and/or ACLs)
If asked, the three switch Layer 2 functions: address learning, forward/filter, loop avoidance (STP)
Forward vs Filter simply refers to: if MAC table says for Port 1, not port 2, forward to port 1, filter from port 2.
Is looked at by some people as two decisions:
1. Forwarding decision: to send it to the right port (associated with that MAC address)
2. Filtering decision: to NOT send it out the other ports.

**IP Routing basic decisionmaking:**
Four-step process of how routers route (forward) packets
Layer 2 first- check FCS. If errors occurred, discard the frame.
Trash the old layer 2 header and trailer, leaving the IP packet, which is sent to Layer 3.
Compare to the routing table, and find the outgoing interface of the router (next-hop IP).
Encapsulate into new layer 2 header/trailer for the outgoing interface and forward the frame. (how does it know the destination MAC? The ARP table)

Goals of IP routing protocols
- dynamically learn and fill the routing table with a route to each subnet on the network.
- choose and place the best route in the routing table.
- notice when table's routes are no longer valid, remove them. Get new one from neighbor.
- work quickly - for fast convergence time and routing updates
- prevent routing loops.

**Layer 2: Broadcast and Collision Domains - FF:FF:FF:FF:FF:FF**
*1. Each router port is a separate broadcast domain (also a separate collision domain)*
*2. Each port on a switch (or bridge) is a separate collision domain.*
*3. Everything connected to a hub is in one collision domain (switches break up collision domains).*

Collision domains: sending and receiving data frames at same time is not supported (half-duplex).
If devices send at the same time the frames collide, a jamming signal (backoff) is sent, devices set timers to try again, which slows everything down. Rely on CSMA/CD to try to mediate collisions.

Routers move traffic between IP networks/subnets. Routers also separate broadcast domains.
If a router interconnects two networks, each network is its own broadcast domain
If a router has two ports in the same subnet, those are two different broadcast domains.
*Because, routers do not forward layer 2 broadcast traffic*

*- At a home office: adding collision domains means increased bandwidth.* With a hub (1 collision domain), only one PC can send at a time, for a theoretical maximum capacity of 100 Mbps for the entire LAN. Replace with a switch, and you get 100 Mbps per link, for a total of 1000 Mbps (1Gbps) and the ability to use full-duplex on each link, effectively doubling the capacity to 2000 Mbps (2Gbps)

**Ping (Packet Internet Groper)**
The Ping program uses the alphabet in the data portion of the packet as a payload, typically around 100 bytes by default, unless, of course, you are pinging from a Windows device, which thinks the alphabet stops at the letter W

**Controlling the Source IP Address with Extended ping**
R1# ping
Protocol [ip]:
Target IP address: 172.16.2.101
Repeat count [5]:
Datagram size [100]:
Timeout in seconds [2]:
*Extended commands [n]: y*
Source address or interface: 172.16.1.1
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.2.101, timeout is 2 seconds: Packet sent with a source address of 172.16.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms

Typically you'll see three values at the end separated with the slash "/," the minimum, average, and maximum RTT
Cisco ping Return Codes:
! - Each exclamation point indicates receipt of an ICMP echo reply. The ping completed successfully.
. - Each period indicates one timed out - blocked by access list or firewall, connectivity problem, or "unreachable" error
U - An ICMP unreachable message was received. - A router along the path did not have a route to the destination address.
C - An ICMP source quench message was received. - A device along the path- may be receiving to much traffic.
& - An ICMP time exceeded message was received. - A routing loop may have occurred.

**Demystifying the lost first ping**
When pinging a directly-attached host (end-station) from a router, it's quite common to lose the first reply, as shown in the following example (the same symptom might occur when pinging a remote host that has been inactive).
Actually, it's not the reply that was lost, the request was never sent out. Whenever a router has to send a packet to the next-hop (or directly attached destination) that has no entry in the ARP table, the ARP request is sent out, but the original packet is unconditionally dropped.

Debugging ARP requests while using the ping command:

```
Router2#show arp
Protocol  Address      Age (min)   Hardware Addr   Type   Interface
Internet  10.0.0.6     -           0016.c876.8b38  ARPA   FastEthernet0/0
Internet  10.0.0.5     0           0016.c7fe.f150  ARPA   FastEthernet0/0
Router2#debug arp
ARP packet debugging is on
Router2#ping 10.0.0.10
Sending 5, 100-byte ICMP Echos to 10.0.0.10, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 1/1/4 ms
08:26:21: IP ARP: creating incomplete entry for IP address: 10.0.0.10 interface FastEthernet0/0
08:26:21: IP ARP: sent req src 10.0.0.6 0016.c876.8b38, dst 10.0.0.10 0000.0000.0000 FastEthernet0/0
08:26:21: IP ARP: rcvd rep src 10.0.0.10 000c.29a7.8ade, dst 10.0.0.6 FastEthernet0/0
```

ICMPv6 includes a different set of message types that are particularly useful when performing local network testing. Type 128 (echo request) messages, which you can use to remotely sweep IPv6 networks.

Firewalk first calculates the distance to the gateway, and then sends packets destined for the target with a TTL that is one hop beyond. Based on the responses, the tool is able to map the filtering policy.

- If an ICMP type 11 code 0 (TTL exceeded in transit) message is received, the packet passed through the filter and a response was later generated.
- If the packet is dropped without comment, it was probably done at the firewall.
- If an ICMP type 3 code 13 (communication administratively prohibited) message is received, a simple filter such as a router ACL is being used.

If the packet is dropped without comment, this doesn't necessarily mean that traffic to the target host and port is filtered. Some firewalls know that the packet is due to expire and will send the expired message whether the policy allows the packet or not.

**Traceroute**
Cisco traceroute sends 3 UDP datagrams for each hop- first, three with a TTL field value set to 1 to cause the datagram to "timeout" as soon as it hits the first router, which responds with ICMP "time exceeded". Next, three more with the TTL value set to 2 for the second router, and so forth, incrementing the TTL for each hop. Since these datagrams try to access an invalid port (default 33434) at the destination host, the host will respond with ICMP "port unreachable" which tells the program to finish (Windows destinations send an ICMP echo reply back)
Standard traceroute defaults to using UDP on Linux, ICMP echo on Windows, but the principle is the same.

On Cisco IOS, running ping or traceroute without an IP address will allow the option to use extended versions of the programs. This will query for source and destination IPs in order to run those programs between specified gateways to check routing, troubleshoot routing loops, or packet loss; to determine the type of connectivity problem, to narrow down where the problem occurs.

**Cisco IOS Extended Traceroute example:**
Router A#traceroute
Protocol [ip]:
Target IP address: 192.168.40.2
!--- The address to which the path is traced.
Source address: 172.16.23.2
Numeric display [n]:
Timeout in seconds [3]:
Probe count [3]:
Minimum Time to Live [1]:
Maximum Time to Live [30]:
Port Number [33434]:
Loose, Strict, Record, Timestamp, Verbose[none]:
Type escape sequence to abort.
Tracing the route to 192.168.40.2
1 172.31.20.2 16 msec 16 msec 16 msec
2 172.20.10.2 28 msec 28 msec 32 msec
3 192.168.40.2 32 msec 28 msec *

Cisco IOS traceroute returns these codes, here's their possible cause.
nn msec - This gives, for each node, the round-trip time (in milliseconds) for the specified number of probes
* - The probe timed out. - A device along the path either did not receive the probe or did not reply with an ICMP "packet life exceeded" message.
A - Administratively prohibited. - a firewall or router, may be blocking the probe and possibly other or all traffic; check access lists.
Q - Source quench. - A device along the path may be receiving to much traffic; check input queues.
H - An ICMP unreachable message has been received. A routing loop may have occurred.

**TCP Traceroute**
Both traceroute and tcptraceroute work on the same basic principle. Employing a tool for performing TCP traceroute is becoming a more popular option since it is less likely to be filtered. All will generate ICMP error messages when the TTL expires. Some firewalls block the "time exceeded" ICMP packets. SYN packets are used in TCP traceroutes since firewalls will often block other TCP packets that aren't part of an established connection.

**intrace** waits for an existing TCP connection. When it sees a connection it will send short-lived packets which appear as being part of the observed connection. intrace can do that because it has seen the packets, and so knows the IP addresses, ports and sequence numbers. The short-lived packets are adjusted so that they will not disrupt the TCP connection (i.e. they are simple "ACK" packets with no data by themselves, so the destination OS will simply ignore them).

Classic traceroute rely on ICMP type 11 (Time exceeded) responses from each hop along the route. If ICMP type 11 responses are being blocked by your firewall, traceroute will not work. These packets are inbound, not outbound.

TCP SYN packets will cause either a RST packet or a SYN ACK packet in response when they reach their destination. There is more about those sort of TCP interactions and what they mean in the section on scanning

It is possible to get ICMP type 3 code 4 responses back instead of ICMP type 11 responses if you send a large packet with the "Do not fragment" flag set, however this is likely only to allow you to find the hop with the smallest MTU. You will normally only get this sort of response from one hop along the route. Not all of them.

Windows has the pathping utility, and there is also MTR (https://github.com/White-Tiger/WinMTR). Older tools include vtrace and Cheops which you may need to use archive.org's WayBackMachine to find.

**Simple routing and switching round trip *HostA (172.16.10.2) pings HostB (172.16.20.5) through one router***
*Note: this example ignores the issue mentioned in the ping section "what happens to the first ping?"*

1- ICMP's ping process creates it's echo request payload, which is simply the alphabet in the data field. It's handed to IP to form a packet. At a minimum, this packet contains an IP source address, an IP destination address, and a Protocol field with 0x01

3- IP checks to see if the destination IP address is a device on the local LAN or on a remote network. Since the destination device is on a remote network, the packet needs to be sent to the default gateway, so it finds it in the routing table. *Remember that hardware addresses are always local, so...*

4- It finds the default GW is 172.16.10.1, so the ARP cache is checked for the MAC if it has already been resolved. If the ARP cache doesn't have it, an ARP broadcast is sent out on local network for 172.16.10.1, router responds, and the cache updated with the info and an activity timer for it.

It turns out that the ARP table had it, and says it's the router's interface with the MAC address on Ethernet0, so it's activity timer is reset to 0, and the packet is *then* ready to be handed to a different part of the Data Link layer for framing.

5- With destination gateway and source MAC addresses, ether-type (0x0800 for IP) and the packet payload, the LAN driver is used to provide media access via the type of LAN (Ethernet), a frame is then generated, adding the FCS field (the result of the CRC).

6- Frame is given to the wire bit-by-bit. *THE FRAME HAS FINALLY ENTERED THE NETWORK*

7- All devices on the collision domain get the bits, rebuild the frame, and run a CRC to compare the FCS field for integrity. If it fails the integrity check it's discarded. *(note: this takes account if we are in a collision domain)*

8- If it passes, the destination MAC is checked to see if it matches it's own address. Eth0 on the router gets it, checks the ether-type field and gives the enclosed packet to IP (the rest is discarded).

9- IP on the router checks the packet header for errors (it doesn't run a complete CRC like the Data Link layer does on the frame). Since the packet's destination IP doesn't match any of the addresses configured on the router's interfaces, the router will look in its routing table. *If there is no entry for the network 172.16.20.0 the packet will be discarded immediately and an ICMP message will be sent back to the originating device with a destination network unreachable message.*
In this case it was found- the packet is routed to the exit interface (Eth1) for that subnet. Since the routing table shows "directly connected," no routing protocols are needed.

10- As the IP layer hands the packet to Layer 2, the MAC of HostB's IP address is either found in the ARP cache, or an ARP request is sent out for it (just as before in steps 3-6). We can safely say that the IP packet has been switched to a buffer before it's framing. It's framed, and sent out, just like in steps 5 and 6. *THE FRAME HAS LEFT THE ROUTER*

11- HostB receives the frame and immediately runs a CRC. If the result matches the information in the FCS field, the hardware destination address will be then checked next. If the host finds a match, the Ether-Type field is finally checked to determine the Network layer protocol that the packet belongs with (IPv4), and it is passed up the stack.

12- IP gets the packet, runs a CRC on the header, checks the destination address and sees it's own address, so it checks Protocol field to find out which Layer 4 protocol gets the payload segment. DESTINATION REACHED!

13- The payload is handed to ICMP, which understands that this is an echo request. ICMP responds to this by immediately discarding the packet and generating a new payload as an echo reply.

14- A packet is then created including the source and destination addresses, Protocol field, and payload. The destination device is now HostA.

15- Steps 3-6 are repeated on HostB: determining it goes to the gateway's IP address, getting the gateway's MAC, framing it, and sending it to wire for the router.

16- The router's Eth1 interface receives the bits and builds a frame. The CRC is run, and the FCS field is checked to make sure the answers match. It's handed to IP and steps 7-10 are repeated with destination swapped to HostA

17- To route the packet, the device knows to get to network 172.16.10.0 it should exit Eth0, and the MAC for 172.16.10.2 is already cached from the originating trip to HostB, so it builds a frame and sends it out on the wire. (LEAVES THE ROUTER)

18- The destination host receives the frame, runs a CRC, checks the destination MAC, then looks at the Ether-Type field, saying to hand it to off at the Network layer, it checks the Protocol field which says ICMP, and ICMP determines the packet to be an ICMP echo reply. ICMP acknowledges that it has received the reply by sending an exclamation point (!) to the user interface. ICMP then attempts to send four more echo requests to the destination host.

***Fast Subnetting Examples - Subnet and Binary Practice***

*Remember to always place largest subnets first. How many hosts and how many subnets?*

**Class C Subnetting - Subnetting for Hosts**

**You are given 192.168.1.0 to start with, and are being asked for:**

***- one subnet that contains at least 50 usable IP addresses.***

***- two subnets that contains at least 14 usable IP addresses each.***

*Start all this with the biggest subnet. Consider our bit increments:*

We need 50, so draw the line where 50 is:

128 64 | 32 16 8 4 2 1

|

32+16+8+4+2+1=63 This fits so we use it. *That 64 marks where our first network bit is.*

Our block ends at 192.168.1.63 (our broadcast address) and our first network is 192.168.1.0

What's the subnet mask? Add the unused bits: 128+64=192. The subnet mask is 255.255.255.192

CIDR/ Prefix length? 8 8 8 2 Count all the network bits- 26, 8+8+8+(128 and 64= 2 bits)

So, this subnet is 192.168.1.0/26 Part 1 is done, move on to part 2:

---

*Second subnet starts at 192.168.1.64*

We need 14, so draw the line:

128 64 32 16 | 8 4 2 1

|

8+4+2+1=15 - This fits so we use it. *That 16 marks where our first network bit is.*

64 is this new network number, and with a block of 15, it's broadcast is 192.168.1.79

What's the subnet mask? Add unused bits: 128+64+32+16=240 The subnet mask is 255.255.255.240

CIDR/ Prefix length? 8 8 8 4 Count all the network bits- 28 (128, 64, 32, 16= 4 bits)

So, our second subnet is 192.168.1.64/28 Part 2 is done, and we are almost finished

---

*Third subnet starts at 192.168.1.80*

We need 14 again:

128 64 32 16 | 8 4 2 1

|

8+4+2+1=15 - Just like the last step

Our block ends at 192.168.1.95 (our broadcast address)

What's the subnet mask? Add unused bits: 128+64+32+16=240 The subnet mask is 255.255.255.240

CIDR/ Prefix length? 8 8 8 4 Count all the network bits- 28 (128, 64, 32, 16= 4 bits)

So, our third subnet is 192.168.1.80/28, and we are done!

*Next block available to continue begins at 192.168.1.96 so you have 96-255 to use in the future (block of 159 numbers for the 192.168.1.0 network we own)*

*You may be wondering... The line for subnet 1 was drawn between 64 and 32. 128 and 64 were included in counting up network bits when we determine subnet mask and the CIDR, but 64 is the block size, which is the 7th bit, not the 6th. Why? The reason for that is that is the 7th bit is now designated to chop up into networks, so it's a network bit. It has been appropriated (i.e. "borrowed) from the host bits. The block size is designated by the number to the left of where we draw the line.*

**Class B Subnetting - Subnetting for Hosts**

**You are given 170.70.0.0 to start with, and are being asked for:**

*- two subnets that contain 1000 usable IP addresses.*
*- one subnet that contains 500 usable IP addresses.*
*- one subnet that contains 100 usable IP addresses.*

128 64 32 16 | 8 4 2 1

So what we were handed and are starting off with 170.70.0.0, 255.255.0.0
To make the first subnet of 1000 hosts. We need to add the 3rd octet in the diagram like this:

32768 16384 8192 4096 2048 1024 | 512 256 • 128 64 32 16 8 4 2 1

1023 - This fits so let's use it. *That 1024 marks where our first network bit is.*
This uses 10 host bits (8 bits for the 4th octet + 2 used in the 3rd= 10)
32 bits-10 bits= 22 bits for the network 8 8 6 0 - 170.70.0.0/22
The line was drawn in the 3rd octet going up to the 1023 mark.
/22 has a block size of 4, a subnet of 252 (256-4)
The subnet mask is = 255.255.252.0
Broadcast is 170.70.3.255 - the next network is 170.70.4.0
---
The second subnet starts at 170.70.4.0.
We need 1000 hosts again- what we just did, so this is easy. Just update the network number with what we already figured out. We know this is a /22 subnet with increments of 4 as the block size.
Subnet mask is 255.255.252.0
Broadcast is 170.70.7.255 - the next network is 170.70.8.0
---
Third subnet starts at 170.70.8.0
To make the subnet of 500 hosts:

32768 16384 8192 4096 2048 1024 512 | 256 • 128 64 32 16 8 4 2 1

511 - This fits so we use it. This equals 9 bits, leaving 23 remaining (32 bits-9 bits)
CIDR notation? 8 8 7 0 Network bits used- 23 (512 leaves 7 bits in 3rd octet)
170.70.8.0/23 with a block size of 2
Subnet mask is 255.255.254.0
Broadcast is 170.70.9.255 - the next network is 170.70.10.0
---
Fourth subnet: 170.70.10.0
To make the subnet of 100 hosts:

32768 16384 8192 4096 2048 1024 512 256 • 128 | 64 32 16 8 4 2 1

127 - This fits so we use it. This equals 7 bits, leaving 25 remaining (32 bits-7 bits)
CIDR notation? 8 8 8 1 Count the bits used- 25 (128 leaves 1 bits in 4rd octet)
170.70.10.0/25 - We didn't even have to touch the third octet for this one.
128 is the 25nd bit, thus subnet mask is 255.255.255.128
Broadcast is 170.70.10.127

Next network will start at 170.70.10.128. *170.70.10.128 through 170.70.244.128 are still unused!*

At the beginning, we determined how many bits, which is the first step. Here it was easy- we found where we could fit 50 bits on the bit counting line, but notice that in a way we counted from the right side. When provisioning networks instead of hosts, we have to figure out how many bits will do the job, and then count from the LEFT, adding to the original network numbers, borrowing from host bits.

**Class B Subnetting - Subnetting for Networks**

We get a class B: 140.78.0.0, 255.255.0.0; asked for 29 networks. Add 2 for NetID and BC is 31

*First question! How many bits?* 1+2+4+8+16 = 31 = uses 5 bits

Count 5 bits (from the LEFT for networks) and borrow from the host bits to extend network bits:

| 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | • | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Our new starting subnet mask is going to be 255.255.248.0 (the 5 bits)

---

This is our first network - 140.78.8.0

| **140.78.** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is our first network's broadcast- 140.78.15.255

| **140.78.** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | • | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

---

Increment 1 binary in the network portion (the left). This is our second network - 140.78.16.0

| **140.78.** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is our second network's broadcast- 140.78.23.255

| **140.78.** | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | • | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

---

Increment 1 binary in the network portion. This is our third network- 140.78.24.0

| **140.78.** | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is our third network's broadcast- 140.78.31.255

| **140.78.** | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | • | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

---

Increment 1 binary in the network portion. This is our fourth network - 140.78.32.0

| **140.78.** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is our fourth network's broadcast- 140.78.39.255

| **140.78.** | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | • | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

*...And so forth. We begin to see the pattern- multiples of 8, our dividing line gave us the block size*

| **Net ID** | **1st address** | **Last address** | **Broadcast** |
|---|---|---|---|
| 140.78.8.0 | 140.78.8.1 | 140.78.15.254 | 140.78.15.255 |
| 140.78.16.0 | 140.78.16.1 | 140.78.16.254 | 140.78.23.255 |
| 140.78.24.0 | 140.78.24.1 | 140.78.31.254 | 140.78.31.255 |
| 140.78.32.0 | 140.78.32.1 | 140.78.39.254 | 140.78.39.255 |

What's the last network you can give out? 140.78.240.0:

| **140.78.** | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

It's broadcast is 140.78.247.255

| **140.78.** | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | • | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Class B Subnetting - Subnetting for Networks - Example 2**

We are given a Class B: 150.9.0.0, BC: 255.255.0.0
We are asked for 10 networks- always add 2 = We really need 12; for networks count from the LEFT

How many bits? 12, so 1100=12 = 4 bits. Count 4 from the left - New magic number is 16
Our new starting subnet mask is going to be 255.255.240.0 (the 5 bits)

| 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 • | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is our first network - 150.9.16 .0

| **150.9.** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is our first network's broadcast- 150.9.31.255

| **150.9.** | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | • | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

The last example walked through discovering our networks and we saw the pattern for block of 8 bits repeating.
Below it is the same thing using the new info, but the block is 16.

| **Net ID** | **1st address** | **Last address** | **Broadcast** |
|---|---|---|---|
| 150.9.0.0 | 150.9.0.1 | 150.9.15.254 | 150.9.15.255 |
| 150.9.16.0 | 150.9.16.1 | 150.9.31.254 | 150.9.31.255 |
| 150.9.32.0 | 150.9.32.1 | 150.9.47.254 | 150.9.47.255 |

150.9.48.0, 150.9.64.0, 150.9.80.0, 150.9.96.0, 150.9.112.0, 150.9.128.0, 150.9.144.0, 150.9.160.0...

**What if...? Subnet Masks and borrowing from the fourth octet**

What if we instead were working with 10 bits instead of 4? How do we deal with the subnet mask?
Remember we are given a Class B: 150.9.0.0

| 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 • | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is our first network - 150.9.0.0- 150.9.0.63

| **150.9.** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| **Net ID** | **1st address** | **Last address** | **Broadcast** |
|---|---|---|---|
| 150.9.0.0 | 150.9.0.1 | 150.9.0.62 | 150.9.0.63 |
| 150.9.64.0 | 150.9.0.64 | 150.9.0.126 | 150.9.1.127 |

Use 64 (the 7th bit), 255.255.255.192 - since we are borrowing from the 4th octet it might look like a class C, but we can ultimately see by the first octet of the address (150) that this is Class B, but it's the net/host boundary of the address dictates the subnet mask.

**Class B Subnetting - Subnetting for Hosts - Review**

We are given a Class B: 160.12.0.0, BC: 255.255.0.0 We are asked for 4080 hosts so add 2= 4082 Bits
So we need 12 bits- for hosts count from the RIGHT

| 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 • | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Subnet mask is 240 (128+64+32+16) "Magic number" is 16
160.12.0.0... 160.12.15.255
160.12.16.0… 160.12.31.255
160.12.32.0, 160.12.48.0, 160.12.64.0, etc

**Class C Subnetting - Subnetting for Networks - Review**

We are given 201.9.6.0 told to make 25 subnets- add 2 and it's 27.
27 = 5 bits - count from the left
201.9.6. 0 0 0 0 1

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|

CIDR is /29 and 255.255.255.248 mask
Network IDs are 201.9.6.8, 201.9.6.16, 201.9.6.24, 201.9.6.32, 201.9.6.40 ... etc.

**Variable Length Subnet Mask - VLSM #1**
Rule 1: Always make networks from highest to lowest in size
We have a starting address of 172.16.0.0
We need 4 subnets, sized with 8000, 1000, 400, and 100 hosts respectively.

First net of 8000 hosts- how many bits? 13 Bits start with the originating network ID (172.16.0.0) *Count from the right:*

32768 16384 8192 | 4096 2048 1024 512 256 • 128 64 32 16 8 4 2 1

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **172.16.** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

We get 172.16.0.0/19 and know the next network ID is 172.16.32.0.
*So, the first VLSM block is 172.16.0.0/19, mask is 255.255.224.0, and broadcast of 172.16.31.255*

---
Second network: 1000 hosts. Line is now between 1024 and 512 (10 bits).
*Depending how you do yours, you can leave the 32 marker from the first network turned on, but move our dividing line to the new spot.* So we start with 172.16.32.0

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **172.16.** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

32768 16384 8192 4096 2048 1024 | 512 256 • 128 64 32 16 8 4 2 1

*This VLSM block is 172.16.32.0/22, subnet mask is 255.255.252.0, with a broadcast of 172.16.35.255*

---
Third network: 400 hosts - starts at 172.16.36.0

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **172.16.** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Do the same thing we just did with previous markers.
Line is between 256 and 512

32768 16384 8192 4096 2048 1024 512 | 256 • 128 64 32 16 8 4 2 1

This is a /23 subnet (8+8+7). We know from 32 + 4 + 2=38 that the next network ID is 172.16.38.0
*This VLSM block is 172.16.36.0/23, subnet mask is 255.255.254.0, with a broadcast of 172.16.37.255*

---
Fourth network: 100 hosts - starts at 172.16.38.0

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **172.16.** | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The line is between 128 and 64 we are now borrowing host bits from the 4th octet

128 | 64 32 16 8 4 2 1

This is a /25 subnet (8+8+9). 128 is the next network ID
*This VLSM block is 172.16.38.0/25, subnet mask is 255.255.128.0, with a broadcast of 172.16.127.255*

| Hosts | Net ID | Last address | Broadcast |
|---|---|---|---|
| 8000 | 172.16.0.0 | 172.16.31.254 | 172.16.31.255 |
| 1000 | 172.16.32.0 | 172.16.35.254 | 172.16.35.255 |
| 400 | 172.16.36.0 | 172.16.37.254 | 172.16.37.255 |
| 100 | 172.16.38.0 | 172.16.38.126 | 172.16.38.127 |

***Next network would begin at 172.16.38.128 for future space (room for about 216 /24 networks or in ip addressed formatted binary 00000000.00000000.11010010.10000000)***

**Variable Length Subnet Mask - VLSM #2**
140.58.0.0 is our starting IP. We need networks that are big enough to cover these host blocks:
Blocks for 7000, 2500, 1800; and we need 2 blocks of 900, a 500 block, and 2 tiny two-host networks.

First network: 7000 hosts- how many bits? 13 Bits:

32768 16384 8192 | 4096 2048 1024 512 256 • 128 64 32 16 8 4 2 1

So we have this- start with the originating network ID:

| **140.58.** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is a /19 subnet (8+8+3).
*This VLSM block is 140.58.0.0/19, subnet mask is 255.255.254.0, with a broadcast of 140.58.31.255*

---
So we have 140.58.32.0 as a starting network ID:

| **140.58.** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Second network: 2500 hosts. Line is now between 2048 and 4096 (12 bits)

32768 16384 8192 4096 | 2048 1024 512 256 • 128 64 32 16 8 4 2 1

This is a /20 subnet (8+8+4). 32 + 16 (48) is the next network ID
*This VLSM block is 140.58.32.0/20, subnet mask is 255.255.240.0, with a broadcast of 140.58.47.255*

---
So we have 140.58.48.0 as a starting network ID:

| **140.58.** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Third network: 1800 hosts. Line is now between 1024 and 2048(11 bits)

32768 16384 8192 4096 2048 | 1024 512 256 • 128 64 32 16 8 4 2 1

This is a /21 subnet (8+8+5). 32 + 16 + 8 (56) is the next network ID
*This VLSM block is 140.58.48.0/21, subnet mask is 255.255.248.0, with a broadcast of 140.58.55.255*

---
So we have 140.58.56.0 as a starting network ID:

| **140.58.** | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Fourth network: 900 hosts. Line is now between 512 and 1024 (10 bits)

32768 16384 8192 4096 2048 1024 | 512 256 • 128 64 32 16 8 4 2 1

This is a /22 subnet (8+8+6). 32 + 16 + 8 + 4 (60) is the next network ID
*This VLSM block is 140.58.56.0/22, subnet mask is 255.255.252.0, with a broadcast of 140.58.59.255*

---
So we have 140.58.60.0 as a starting network ID:

| **140.58.** | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Fifth network: also 900 hosts. Line is between 512 and 1024 (10 bits) again (see 4th network just above for diagram)

*Ok- we need to draw our line, but it's already at the spot we need to put it. The interval hasn't changed so add a 4 to the network. The number line is just a workspace to help us keep track of things, so it's not a big deal.*
This is a /22 subnet (8+8+6). 32 + 16 + 8 + 4 **+ 4** (64) is the next network ID
*This block is 140.58.60.0/22, mask is 255.255.252.0, broadcast of 140.58.63.255; next network number is 140.58.64.0*

If you are doing this on paper/ by hand it can get messy and you may have 172.16.00111100.00000000 down from drawing the number line over and over. Clean up the 64 in the 3rd octet like this since it is 64: 172.16.01000000.00000000

Continuing, we have 140.58.60.0 as a starting network ID:

| **140.58.** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Sixth network: 500 hosts. Line is now between 256 and 512 (9 bits)

32768 16384 8192 4096 2048 1024 512 | 256 • 128 64 32 16 8 4 2 1

This is a /23 subnet (8+8+7). 64 + 2 (66) is the next network ID
*This VLSM block is 140.58.64.0/23, subnet mask is 255.255.254.0, with a broadcast of 140.58.65.255*

---

So we have 140.58.66.0 as a starting network ID:

| **140.58.** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | • | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Seventh and eighth networks: 2 networks of 2 hosts each. We are zooming in to the bottom of the 4th octet. Hypothetically, you could ask if it's a point-to-point not needing reserved network ID and BC addresses, or just throw those in anyway just to save the trouble of having the conversation. I am opting for the latter (2 IP addresses, 4 total, big deal)

128 64 32 16 8 4 2 | 1

These are /30 subnets (8+8+8+6). Now that we are in the 4th octet, our network IDs are 140.58.66.0 (with .1 and .2 hosts, and .3 BC), and 140.58.66.4 (with .5 and .6 hosts, and .7 broadcast)

| **Hosts** | **Net ID** | **Last address** | **Broadcast** |
|---|---|---|---|
| 7000 | 140.58.0.0 | 140.58.31.254 | 140.58.31.255 |
| 2500 | 140.58.32.0 | 140.58.47.254 | 140.58.47.255 |
| 1800 | 140.58.48.0 | 140.58.55.254 | 140.58.55.255 |
| 900 | 140.58.56.0 | 140.58.59.254 | 140.58.59.255 |
| 900 | 140.58.60.0 | 140.58.63.254 | 140.58.63.255 |
| 500 | 140.58.64.0 | 140.58.64.254 | 140.58.64.255 |
| 2 | 140.56.66.0 | 140.58.66.2 | 140.58.66.3 |
| 2 | 140.58.66.4 | 140.58.66.6 | 140.58.66.7 |

Next network would begin at 140.58.66.8

The /30 is considered the smallest network you can make to provision a tiny network for isolated connection coupling. In practice, you will hear about /31s working fine for serial connections which need only 2 addresses provisioned for them, but this is NOT recognized by Cisco as working. A /31 may work fine in a production network, but is guaranteed to be the wrong answer to Cisc exams.

140.58.0.0, our starting class B. For network 1, we needed 7000 provisioned and we got 8190 so there is plenty of room for growth. Network 2 the same thing: we needed 2500 and got a 4094 ceiling for future growth.

All networks have significant overhead, and the only snag might be is if we needed to add routers to the networks with 2 hosts, in which case it is easy to just make some more tiny networks, since all of these networks to handle the enterprise of thousands of hosts barely made a dent in our original Class B address-space.

*The previous pages (subnetting for hosts, subnetting for networks, VLSM 1 & 2) all were made out of the information presented in the classroom videos available here:* ***https://www.youtube.com/user/TheLITCNTT*** *I transcribed them since they were so fast and simple, and didn't require more than making a number line and stuff from memory (for the most part). I am not affiliated with that school, just found the videos of the whiteboard presentation.*

For templating/ reuse:

| 128 | | 64 | | 32 | | 16 | | 8 | | 4 | | 2 | | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | |

| 32768 | | 16384 | | 8192 | | 4096 | | 2048 | | 1024 | | 512 | | 256 | • | 128 | | 64 | | 32 | | 16 | | 8 | | 4 | | 2 | | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Subnet Shortcuts**
*Using binary AND: 1 and 1 = 1, all others = 0 (see below)*

*What is the network number of the address 192.168.100.115 with a subnet mask of 255.255.255.240?*
192.168.100.115 = 11000000.10101000.01100100.01110011
255.255.255.240 = 11111111.11111111.11111111.11110000
ANDed result = 11000000.10101000.01100100.01110000 = 192.168.100.112
192.168.100.115 belongs to the 192.168.100.112 network when a mask of 255.255.255.240 is used.

*It get's simpler: octets of the addresses with a common 255 or 0 can be ignored*
What's the broadcast for the IP address 192.168.100.164 if it has a subnet mask of 255.255.255.248?
192.168.100.164 = 192.168.100.10100100
255.255.255.248 = .....11111000
ANDed result = 192.168.100.10100000 = 192.168.100.160 (here's our subnet number)

Separate the network bits from the host bits:
255.255.255.248 = /29 = the first 29 bits are network bits - so the last three are host bits
----.10100 **000**. Change all host bits to 1, so it's -----.10100111 = 192.168.100.167
The broadcast of 192.168.100.164 is 192.168.100.167 when the subnet mask is 255.255.255.248.

*To what network does 131.186.227.43 belong, if its subnet mask is 255.255.240.0?*
Based on the two shortcut rules, the answer should be 131.186.???.0
So now you only need to convert one octet to binary for the ANDing process:
227 = 11100011
240 = 11110000
11100000 = 224 Therefore, the answer is 131.186.224.0

**What subnet does this belong to?**
*What subnet does 192.168.12.78/29 belong to?*
Our mask is a /29. The next boundary is 32. So 32-29=3.
Now 2^3 = 8 which gives us our block size i.e. 2 to the power of 3 equals 8.

We have borrowed from the last octet as the 29th bit is in the last octet. We start from zero and count up in our block size.
Therefore it follows that the subnets are:
192.168.12.0
192.168.12.8
192.168.12.16
...
192.168.12.72
192.168.12.80
Our address is 192.168.12.78 so it must sit on the 192.168.12.72 subnet.

*What subnet does 172.16.116.4/19 sit on?*
Our mask is /19 and our next boundary is 24. Therefore 24-19=5. The block size is 2^5 = 32.
We have borrowed into the third octet as bit 19 is in the third octet so we count up our block size in that octet. The subnets are:
172.16.0.0
172.16.32.0
172.16.64.0
172.16.96.0
172.16.128.0
Our address is 172.16.116.4 so it must sit on the 172.16.96.0 subnet.

What subnet does 10.34.67.234/12 sit on?
Our mask is 12. Our next boundary is 16. Therefore 16-12=4. 2^4=16 which gives us our block size.
We have borrowed from the second octet as bit 12 sits in the second octet so we count up the block size in that octet. The subnets are:
10.0.0.0
10.16.0.0
10.32.0.0
10.48.0.0
.....etc
Our address is 10.34.67.234 which must sit on the 10.32.0.0 subnet.

**What is the *valid* host range of the Nth subnet of abc.xxx.yyy.zzz/xx?**

*What is the valid host range of the 4th subnet of 192.168.10.0/28?*
The block size is 16 since 32-28=4 and 2^4 = 16.
We need to count up in the block size in the last octet as bit 28 is in the last octet.

192.168.10.0
192.168.10.16
192.168.10.32
192.168.10.48
192.168.10.64
The 4th subnet is 192.168.10.48 and the host range must be 192.168.10.49 to 192.168.10.62, remembering that the subnet and broadcast address cannot be used.
---

*What is the valid host range of the 1st subnet of 172.16.0.0/17?*
The block size is 128 since 24-17=7 and 2^7 = 128.
We are borrowing in the 3rd octet as bit 17 is in the 3rd octet. Our subnets are:

172.16.0.0
172.16.128.0

The first subnet is 172.16.0.0 and the valid host range is 172.16.0.1 to 172.16.127.254. You must remember not to include the subnet address (172.16.0.0) and the broadcast address (172.16.127.255).

*What is the valid host range of the 7th subnet of address 10.0.0.0/14?*
The block size is 4, from 16-14=2 then 2^2 = 4.
We are borrowing in the 2nd octet, so count in the block size from zero 7 times to get the 7th subnet.

The seventh subnet is 10.24.0.0. Our valid host range must be 10.24.0.1 to 10.27.255.254 again remembering not to include our subnet (10.24.0.0) and the broadcast address (10.27.255.255).

10.0.0.0
10.4.0.0.0
10.8.0.0.0
...
10.20.0.0.0
10.24.0.0.0

**Dotted decimal mask? What's the block size?**
A mask of 255.255.192.0 - you would simply count up in 256-192 = 64 in the third octet.
Or 255.224.0.0 - block size is 256-224=32 in the second octet.

---
Question: You are designing a subnet mask for the 10.0.0.0 network. You want 3800 subnets with up to 3800 hosts on each subnet. What subnet mask should you use?
Answer: 255.255.240.0

Question: How many subnets and hosts per subnet can you get from the network 172.29.0.0 255.255.254.0?
Answer: 128 subnets and 510 hosts

Just memorize how many items you get for each number of bits (subtract two for hosts)

000000000001 = 2 (1)
000000000011 = 4 (2)
000000000111 = 8 (3)
000000001111 = 16 (4)
000000011111 = 32 (5)
000000111111 = 64 (6)
000001111111 = 128 (7)
000011111111 = 256 (8 )
000111111111 = 512 (9)
001111111111 = 1024 (10)
011111111111 = 2048 (11)
111111111111 = 4096 (12)

If you can't remember what one is, just remember it's twice the number of the one before it (or halve the number after it if you're counting down). There's no math, tricks, etc. involved.

So for the first question, you'll need 12 bits for both because the value is between 2048 and 4096 (you obviously have to go with the higher value).

You can look at the other question the same way. /23 on a class B will give you 7 subnet bits and 9 host bits. 7 bits is 128 and 9 bits is 512 (minus 2 gives you 510).

Different answer:
Bits do you need to borrow to accommodate 3800 subnets?
12 bits as 2 ^ 12 = 4096.
Your network address is a /8 by default as it is a Class A address so 8 + 12 = /20 mask.

The second question is straightforward.
Your address is Class B so it has a default mask of /16.
255.255.254.0 is the same as /23, so 23-16=7.
2^7 = 128 subnets
(2 ^ (32 - 23))-2 = 2^9-2 = 510 hosts.

**Interesting:**
Question: *What is the last valid host on the subnetwork 172.25.248.0/21?*
Answer: 172.25.255.254

Subnet mask: inside an octet, mask is your current block size added to the next-highest neighboring block size (except 128 has a  mask of 128)
**128+64=192, 192+32=224, 224+16=240, etc:**

| 128 | 192 | 224 | 240 | 248 | 252 | 254 | 255 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

# Subnetting Chart Tools for Quick Reference

*Constructed by Tristan Mendoza*

CIDR is the number of network bits used. 255.255.255.192 = 11111111.11111111.11111111.11000000 = /26 network bits, 32-26 = 6 host bits
Many teachers and some exams require you know the long way to do things- longhand calculation of usable hosts per subnet= $2^{H}$-2 (H= host bits)
Usable subnets=$2^{N}$ with N for network. If you know how to draw the number line of binary with bits up high enough, it's clear every bit after 4096 is just as before going to double the previous bit's number. It's easier to draw what you need like done in this sheet.
Block Sizes: Subtract subnet mask from 256 to get the bit position or "magic number." Mask of 192 (256-192=64) gives the network numbers of 0, 64, 128, and 192. So 255.255.255.192 has a /26 CIDR. Since 32-26= 6 host bits, we get $2^{6}$-2=62 usable hosts per subnet.
*Remembering mask numbers: Add bit value to it's bigger neighbor's mask like this: 128+64=192, 192+32=224, 224+16=240, etc.*

| **Normal Bit Value/ Position** | **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
|---|---|---|---|---|---|---|---|---|
| **2-exponent representation** | $2^{7}$ | $2^{6}$ | $2^{5}$ | $2^{4}$ | $2^{3}$ | $2^{2}$ | $2^{1}$ | $2^{0}$ |
| **Subnet Mask for Bit Position** | 128 | 192 | 224 | 240 | 248 | 252 | 254 | 255 |
| | | | | | | | | |
| **Class C CIDR** | /25 | /26 | /27 | /28 | /29 | /30 | /31 | /32 |
| Subnets | 2 | 4 | 8 | 16 | 32 | 64 | 128 | n/a |
| Hosts per Subnet | 126 | 62 | 30 | 14 | 6 | 2 | 2-no NetID/BC | n/a |
| | | | | | | | | |
| | | | *A /24 is all host bits in the 4th octet, thus 1 class C (2 /25's, 4 /26's, or 8 /27's, etc.)* | | | | | |
| **Class B CIDR (3rd Octet)** | /17 | /18 | /19 | /20 | /21 | /22 | /23 | /24 |
| Subnets | 128 /24's | 64 /24's | 32 /24's | 16 /24's | 8 /24's | 4 /24's | 2 /24's | 1 Class C |
| Hosts (unsubnetted) | 32766 | 16382 | 8190 | 4094 | 2046 | 1022 | 510 | 254 |
| | | | | | | | | |
| **Class B CIDR (4th Octet)** | /25 | /26 | /27 | /28 | /29 | /30 | | |
| Subnets | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | | |
| Hosts (unsubnetted) | 126 | 62 | 30 | 14 | 6 | 2 | | |
| | | | | | | | | |
| | | | *Like /24, a /16 is a whole class B, holding all host bits in the 3rd and 4th octets* | | | | | |
| **Class A CIDR (2nd Octet)** | /9 | /10 | /11 | /12 | /13 | /14 | /15 | /16 |
| Subnets | 128 /16's | 64 /16's | 32 /16's | 16 /16's | 8 /16's | 4 /16's | 2 /16's | 1 Class B |
| Hosts per Subnet (subtract 2) | 8,388,606 | 4,194,302 | 2,097,150 | 1,048,574 | 524,286 | 262,142 | 131,070 | 65,534 |
| | | | | | | | | |
| **Class A CIDR (3rd Octet)** | /17 | /18 | /19 | /20 | /21 | /22 | /23 | /24 |
| Subnets | 512 | 1024 | 2048 | 4096 | 8192 | 16,384 | 32,768 | 65,536 |
| Hosts per Subnet (subtract 2) | 32766 | 16382 | 8190 | 4094 | 2046 | 1022 | 510 | 254 |

***Default Classful System***

| | Leading Bits | | Starts At | Net Bits | Bits Left | # of Nets | Hosts per Net | Default Mask | Native CIDRs |
|---|---|---|---|---|---|---|---|---|---|
| **Class A** | 0xxx, 1-126 | - | 0.0.0.1 | 8 | 24 | 128 | 16,777,216 | 255.0.0.0 | /9 to /16 |
| **Class B** | 10xx, 128-191 | - | 128.0.0.0 | 16 | 16 | 16,384 | 65,534 | 255.255.0.0 | /17 to /24 |
| **Class C** | 110x, 192-223 | - | 192.0.0.0 | 24 | 8 | 2,097,152 | 254 | 255.255.255.0 | /25 to /32 |
| **Class D** | 1110, 224-239 | Multicast | 224.0.0.0 | - | - | - | - | - | |
| **Class E** | 1111, 240-254 | Reserved | 240.0.0.0 | - | - | - | - | - | |

***Reserved and Private Addresses***

| | | |
|---|---|---|
| *Class A* | *10.0.0.0 - 10.255.255.255* | *10.0.0.0/8* |
| *Class B* | *172.16.0.0 - 172.31.255.255* | *172.16.0.0/12* |
| *Class C* | *192.168.0.0 - 255.255* | *192.168.0.0/16* |
| *Loopback* | *127.x.x.x* | *127.0.0.0/8* |
| *APIPA* | *169.254.x.x* | *169.254.0.0/16* |
| *Carrier NAT* | *100.64.0.0 – 100.127.255.255* | *100.64.0.0/14* |
| *Stress-testing* | *198.18.0.0 – 199.19.255.255* | *198.18.0.0/15* |

The main reason the chart above lists CIDR info for unusual-looking address space is to assist with imagining the "subnetting" of large chunks of private address space as is mostly the case in 10.0.0.0/8, but also 172.16.0.0/12 blocks. In those cases, for ease of reference mimicking the traditional routed addressing while adding a more human-recognizable patterns might help, (i.e., 10.1.x.x for eastern US, 10.2.x.x for central U.S., 10.3.x.x for western U.S., and so forth, as appropriate). An entire 10.0.0.0/8 has plenty of space for creativity.

## *Using the worksheet on the right hand side*

**Class B Subnetting - Subnetting for Hosts**
We are given a Class B: 160.12.0.0, BC: 255.255.0.0. We are asked for 4080 hosts so add 2= 4082 Bits: where is the line drawn? 4096 ^ 2048 1024 512 256 128 64 32 16 8 4 2 1
So we need 12 bits- for hosts count from the RIGHT
160.12. 0 0 0 1
128 64 32 16 || 8 4 2 1 . 128 64 32 16 8 4 2 1
Subnet mask is 240 (128+64+32+16) "Magic number" is 16
160.12.16.0… 160.12.31.255
160.12.32.0, 160.12.48.0, 160.12.64.0, etc

**Class C Subnetting - Subnetting for Networks**
We are given 201.9.6.0 told to make 25 subnets- add 2 and it's 27.
27 = 5 bits - count from the left
201.9.6. 0 0 0 0 1
128 64 32 16 8 || 4 2 1
CIDR is /29 and 255.255.255.248 mask
Network IDs are 201.9.6.8, 201.9.6.16, 201.9.6.24, 201.9.6.32, 201.9.6.40 ... etc.

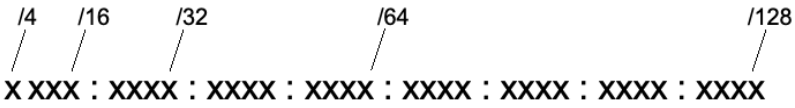
**Class C Subnetting - Subnetting for Hosts**
We have 195.12.8.0 and need 40 hosts +2 = 42 = 6 bits to hold
This time count FROM THE RIGHT to place the divider:
195.12.8.
128 64 || 32 16 8 4 2 1
Subnet mask is 255.255.255.192 (128+64), CIDR is /26
Network IDs 195.12.8.64, 195.12.8.128. That's it since 192 is subnet for these subnets (and it's only a class C

| /9 | 128 | 128 | 24 | $2^{23}$ | 8,388,608 |
|---|---|---|---|---|---|
| /10 | 192 | 64 | 23 | $2^{22}$ | 4,194,304 |
| /11 | 224 | 32 | 22 | $2^{21}$ | 2,097,152 |
| /12 | 240 | 16 | 21 | $2^{20}$ | 1,048,576 |
| /13 | 248 | 8 | 20 | $2^{19}$ | 524,288 |
| /14 | 252 | 4 | 19 | $2^{18}$ | 262,144 |
| /15 | 254 | 2 | 18 | $2^{17}$ | 131,072 |
| /16 | 255 | 1 | 17 | $2^{16}$ | 65,536 |
| | | | . | | |
| /17 | 128 | 128 | 16 | $2^{15}$ | 32,768 |
| /18 | 192 | 64 | 15 | $2^{14}$ | 16,384 |
| /19 | 224 | 32 | 14 | $2^{13}$ | 8,192 |
| /20 | 240 | 16 | 13 | $2^{12}$ | 4,096 |
| /21 | 248 | 8 | 12 | $2^{11}$ | 2,048 |
| /22 | 252 | 4 | 11 | $2^{10}$ | 1,024 |
| /23 | 254 | 2 | 10 | $2^{9}$ | 512 |
| /24 | 255 | 1 | 9 | $2^{8}$ | 256 |
| | | | . | | |
| /25 | 128 | 128 | 8 | $2^{7}$ | 128 |
| /26 | 192 | 64 | 7 | $2^{6}$ | 64 |
| /27 | 224 | 32 | 6 | $2^{5}$ | 32 |
| /28 | 240 | 16 | 5 | $2^{4}$ | 16 |
| /29 | 248 | 8 | 4 | $2^{3}$ | 8 |
| /30 | 252 | 4 | 3 | $2^{2}$ | 4 |
| /31 | 254 | 2 | 2 | $2^{1}$ | 2 |
| /32 | 255 | 1 | 1 | $2^{0}$ | 1 |
| ← CIDR (network bits) | ← subnet mask | ← Binary bit by octet | ← Binary bit placement | ← Power of 2 | ← # of hosts by position |

# IPv6 Overview

- IPv6 core protocol definition is RFC 2460
- RIPng, EIGRPv6, OSPFv3, ICMPv6, traceroute6, ping6, MP-BGP-4 (mulitprotocol BGP v4)
- ARP is replaced with Neighbor discovery, Neighbor solicitation/ duplicated address detection, multiprotocol router discovery and router advertisement.

**IPv6 Numbering Summary:**
Each character is 4 bits, in hexadecimal, so every block of 4 (a quartet) of those is 16 bits. Typically, IPv6 addresses refer to a certain length or place using the backslash notation. This diagram shows the structure of an IPv6 address to demonstrate bit placement/ordering 0-128 bits.



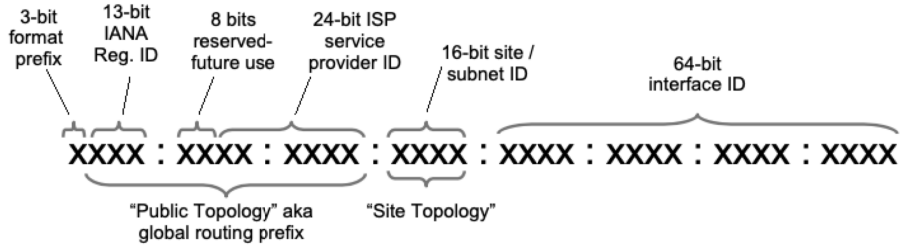

So, a prefix ending at the /48th bit, we can say something like 2001:1234:5678::/48 (how many bits are used)

You'll see IPv6 addresses shortened often. Here's how that works:
Start with expanded address: 2001:0DB8:0000:0000:0000:FF00:0042:8329
- Remove any leading zeros - All leading zeros within each group of 4 hexadecimal digits can go. So, 0DB8 becomes DB8, 0000 becomes 0, and 0042 becomes 42 - 2001:DB8:0:0:0:FF00:42:8329
- Replace consecutive groups of zeros with double colons (::). You can only do this once per address, so do it to the largest grouping or the leftmost if there are a few the same size. Now we have 2001:DB8::FF00:42:8329
- This is more a style thing instead of shortening: make all letters lowercase. It helps with both ease of reading, but also with case-sensitive searching or filtering. We finally get: 2001:db8::ff00:42:8329
- Expanding is easy: inverse the rules and pad out so there are 8 quartets of 4 bits each.

## Address Types

If asked what the 3 types of IPv6 addresses there typically the expected answer is unicast, multicast and anycast. It's most efficient to start of with unicast, of which there are also 3 types, one routable on the internet and the other two that have a local scope. It is also the one that has the most components so it makes the others easier.

*Global Unicast (GUA)*



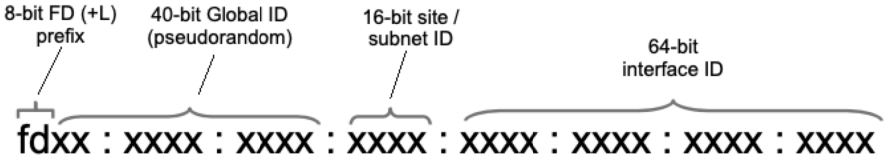

So, using the global unicast address as a starting point, take 2001:0db8:56a4:0001:0000:0000:0000:0002 - or - 2001:db8:56a4:1::2 In this example, the generalized prefix would be 2001:0db8:56a4::/48, the subnet would be 0001, and interface ID (simplified) 2

Global unicast addresses are routable and what you need to use to do anything internet. The other two can't leave your local nets. The also can have a variety of prefixes (the other 2 have the same ones and are easy to spot). As of Feb 2023, IANA lists 2001:0000::/23 – 2c00:0000::/12 as being currently registered/available global unicast prefix blocks ffrom regional registries (RIRs). For more info, see https://www.iana.org/assignments/ipv6-unicast-address-assignments.

If your company was only given one /64, you will likely have a subnet ID assigned by the ISP along with the rest of the prefix illustrated above. The interface ID represents a single interface on any given device. Typically you shouldn't need to worry about altering anything in the prefix your ISP gave you including the subnet ID. You may have been given a larger address space to work with like /32 or /48. There is a section on subnetting those after talking about addresses and such.

2001:1234:3333::/48 -Company B
2001:1234:3333:0001::/64 -Company B's subnet 1
2001:1234:3333:0002::/64 -Company B's subnet 2
2001:1234:3333:2::1 -A host in Company B's subnet 2

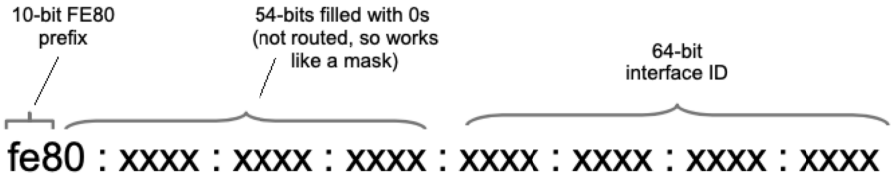
*Unique Local Addresses (ULA) Unicast*




Unique Locals are like the reserved IPv4 addresses for private networks (192.168.0.0/16, 172.16.0.0/12, and 10.0.0.0/8). They aren't routable over the internet (we'll talk about NAT later) but they can move around the local network. They replace an old one named “site local”. These don't use the IANA/ISP's prefix like global unicast, and instead is a “random” replacement (still often called a global ID... out of habit?). This can be autogenerated by the device or done manually, which will be discussed soon.

fc00::/8 is for unique local and assigned from say a DHCP or other provider server
fd00::/8 is only for a locally assigned (IE fd00:: + 40 random bits) without a provider of some kind to generate/hand out addresses.

*Link Local (Unicast), fe80::/10*




Link locals are auto-generated and only useful inside one subnet- can't be routed. They mainly help with the functions of neighbor discovery and next hop configuration. A great way to check out all 3 of these unicast types is to run ifconfig on your workstation.

*Multicast address ff00::/8*
Identifies a group of nodes or interfaces with traffic forwarded to all the nodes in the group. Addresses are all assigned out of the FF00::/8 block; also have a scope associated: link local being just like the scope of unicast LL, 'organization' with same scope as unique local, and global.

Anycast address
Identifies a group of nodes or interfaces, with traffic forwarded to the nearest node in the group. An anycast address is essentially a unicast address assigned to multiple devices with a host ID = 0000:0000:0000:0000. (Anycast addresses are not widely used today.)

*Key IPv6 Multicast Addresses*

| | | |
|---|---|---|
| ff01::1 | All nodes in the interface-local | |
| ff02::1 | All nodes in the link-local, RA/RS | |
| ff0X::1 | All nodes address, identify the group of all IPv6 nodes | |
| ff0X::2 | All routers | 1 (interface-local), 2 (link-local), 5 (site-local) |
| ff01::2 | All routers in the interface-local | |
| ff02::2 | All routers in the link-local | |
| ff05::2 | All routers in the site-local | |
| ff02::5 | OSPFv3 | 2 (link-local) |
| ff02::6 | OSPFv3 Designated Routers | 2 (link-local) |
| ff02::9 | RIP Routers | 2 (link-local) |
| ff02::a | EIGRPv6 Routers | 2 (link-local) |
| ff02::d | All PIM Routers | 2 (link-local) |
| ff02::1a | All RPL Routers | 2 (link-local) |
| ff0X::fb | mDNSv6 | All scopes |
| ff0X::101 | All NTP servers | All scopes |
| ff02::1:1 | Link Name | 2 (link-local) |
| ff02::1:2 | All-dhcp-agents | 2 (link-local) |
| ff02::1:3 | Link-local Multicast Name Resolution | 2 (link-local) |
| ff05::1:3 | All-dhcp-servers | 5 (site-local) |
| ff02::1:ff00:0/104 | Solicited-node multicast address | 2 (link-local) |
| ff02::2:ff00:0/104 | Node Information Queries | 2 (link-local) |

**Host Addressing: SLAAC and EUI-64 (RFC 2373)**
Stateless Address Auto-Configuration (SLAAC) enables hosts to generate a unique **routable** address on their own. For this the router sends out a Router Advertisement (RA) periodically, and a host can send a Router Solicitation(RS) in order to trigger an RA. The RA has the prefix and length to use, default gateway. Before using the address it makes, the host uses Duplicate Address Detection (DAD) to make sure it's unique. Since things like DNS servers aren't in RAs, DHCPv6 still has to be used. (DAD takes the place of ARP in IPv6)

For the interface ID, the host either uses a EUI-64 (Extended Unique Identifier) made from the interface's MAC address (most likely), or can optionally generate it randomly. The EUI-64 creation works like this:
1. Split the 6-byte (12-hex digit) MAC address in two down the middle.
2. Insert FFFE in between the two halves.
3. Invert the seventh bit from the beginning.

*Examples:*
Interface MAC address is aa12:bcbc:1234
10101010 represents the first 8 bits of the MAC address (aa) - when inverting the 7th bit it becomes 10101000.
The answer is A8 and the EUI-64 address: 2001:0db8:0:1:a812:bcff:febc:1234 EUI-64

MAC address 0b34:ba12:1234
0b in binary is 00001011, the first 8 bits of the MAC address, which then becomes 00001001
The answer is 09, and the IPv6 EUI-64 address: 2001:0db8:0:1:0934:baff:fe12:1234 EUI-64

*Generating EUI-64 is usually done for us:*
```
ipv6 address 2001:db8:1111:1::/64 eui-64

interface GigabitEthernet0/0
  ipv6 address 2001:db8:1111:1::/64 eui-64
interface serial0/0/0
  ipv6 address 2001:db8:1111:2::/64 eui-64
show ipv6 interface brief
    GigabitEthernet0/0 [up/up]
        fe80::1ff:fe01:101                  <---link local, employing eui64
        2001:db8:1111:1:0:1ff:fe01:101    <---EUI64 global unicast address
    Serial0/0 [up/up]
        fe80::1ff:fe01:101
        2001:db8:1111:2:0:1ff:fe01:101
```

*Cisco router, setting up SLAAC*
```
Router>enable
Router#configure terminal
Router(config)#ipv6 unicast-routing
Router(config)#interface interface
Router(config-if)#ipv6 address ipv6-address/prefix-length
Router(config-if)#no shutdown
```

*Problems with EUI-64*
It has been pointed out that a user cannot connect anonymously to any network if someone knows the EUI-64 interface identifier of that device, which could be exploited such as websites and apps associating different IPv6 addresses to a particular device or user (whether malicious, monitoring, or benign). It is recommended that if the host OS or router does not support autoconfig with Random Interface Identifiers, that static IPv6 address should be used. (MS Windows generates by default a random interface ID for SLAAC)

Best answer to how to make clients go random instead of using EUI-64:
https://superuser.com/questions/243669/how-to-avoid-exposing-my-mac-address-when-using-ipv6

*Stateless and Stateful DHCPv6 in Cisco IOS*
DHCPv6 Stateless mode
- provides network info not in an RA, (no IPv6 address since already provided by SLAAC).
- DNS domain name and server(s), other DHCP options.

```
ipv6 unicast-routing
ipv6 dhcp pool IPV6_DHCPPOOL
address prefix 2001:db8:5:10::/64
domain-name cisco.com
dns-server 2001:db8:6:6::1
interface Vlan20
description IPv6-DHCP-Stateless
ip address 192.168.20.1 255.255.255.0
ipv6 nd other-config-flag
ipv6 dhcp server IPV6_DHCPPOOL
ipv6 address 2001:DB8:0:20::1/64
```

DHCPv6 Stateful aka managed mode assigns unique addresses instead of the client generating one

```
ipv6 unicast-routing
ipv6 dhcp pool IPV6_DHCPPOOL
address prefix 2001:db8:5:10::/64
domain-name cisco.com
dns-server 2001:db8:6:6::1
interface Vlan20
description IPv6-DHCP-Stateful
ip address 192.168.20.1 255.255.255.0
ipv6 address 2001:DB8:0:20::1/64
ipv6 nd prefix 2001:DB8:0:20::/64 no-advertise
ipv6 nd managed-config-flag
ipv6 nd other-config-flag
ipv6 dhcp server IPV6_DHCPPOOL
```

This interface configuration is for a Cisco IOS IPv6 router implementing stateful DHCPv6 on an external DHCP server:

```
ipv6 unicast-routing
domain-name cisco.com
dns-server 2001:db8:6:6::1
interface Vlan20
description IPv6-DHCP-Stateful
ip address 192.168.20.1 255.255.255.0
ipv6 address 2001:DB8:0:20::1/64
ipv6 nd prefix 2001:DB8:0:20::/64 no-advertise
ipv6 nd managed-config-flag
ipv6 nd other-config-flag
ipv6 dhcp_relay destination 2001:DB8:0:20::2
```

**Cisco IOS Routing in IPv6**

*Static routing in IPv6*

```
ipv6 unicast-routing
interface serial0/0/0
  ipv6 address 2001:5432:1111:4::1/64
interface serial0/0/1
  ipv6 address 2001:db8:1111:5::1/64
interface gigabitethernet0/0
  ipv6 address 2001:db8:1111:1::1/64
```

*Static IPv6 with Next-Hop Address*

```
!First command is on R1, listing R2's global unicast
R1(config)#ipv6 route 2001:db8:1111:2::/64 2001:db8:1111:4::2
!This command is on R2, listed R1's global unicast
R2(config)#ipv6 route 2001:db8:1111:1::/64 2001:db8:1111:4::1
!Verify routes with show ipv6 route static
```

*Default Route*

```
B1(config)#ipv6 route ::0 S0/0/1
```

```
show ipv6 route
show ipv6 route static
show ipv6 route local
```

*OSPFv3 Routing - area goes on interfaces*

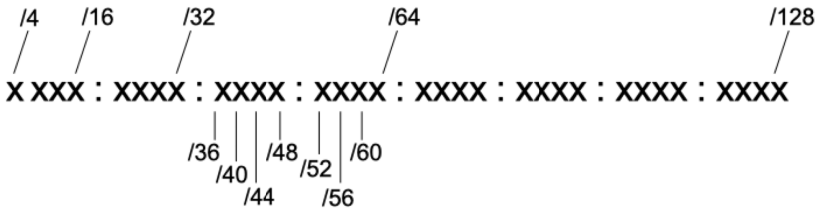
```
ipv6 unicast-routing
interface serial0/0/0
  no ip address
  ipv6 address 2001:abcd:1234:4::1/64
interface s0/0/1
  no ip address
  ipv6 address 2001: abcd 1234:5::1/64
interface GigabitEthernet0/0
  no ip address
  ipv6 address 2001:abcd:1234:5::1/64
ipv6 router ospf 1
    router-id 1.1.1.1
interface s0/0/0
    ipv6 ospf 1 area 0
int gi0/0
    ipv6 ospf 1 area 0

ipv6 unicast-routing
ipv6 router ospf 2
  router-id 2.2.2.2
int s0/0/1
  ipv6 address 2001: abcd:1234:4::2
  ipv6 ospf 2 area 0
int gi0/0
  ipv6 address 2001:abcd:1234:2::2
  ipv6 ospf 2 area 0

show ipv6 ospf
show ipv6 protocols
show ipv6 ospf interface
show ipv6 ospf interface brief
show ipv6 ospf neighbor
show ipv6 ospf database
show ipv6 route ospf
```

**Subnetting IPv6**

In IPv4, subnetting is done to get more addresses for hosts,optimize network distribution to help with that.
In IPv6 we have tons of addresses, so it becomes more of an issue of things like network organization.
Bulk blocks of IPv6 space purchased from RIRs are bought in lengths of /32 - /48 that can be subnetted into multiple /64 subnets. Subnets sized over /64 (like /84 or /96) are not advised since IPv6 features like addressing autoconfiguration rely on being used in a /64 subnet.

| Prefix | Appearance | # of hosts | Number of /64s |
|---|---|---|---|
| /32 | xxxx:xxxx:: | $2^{96}$ | 4294967296 |
| /36 | xxxx:xxxx:x:: | $2^{92}$ | 268435456 |
| /40 | xxxx:xxxx:xx:: | $2^{88}$ | 16777216 |
| /44 | xxxx:xxxx:xxx:: | $2^{84}$ | 1048576 |
| /48 | xxxx:xxxx:xxxx:: | $2^{80}$ | 65536 |
| /52 | xxxx:xxxx:xxxx:x:: | $2^{76}$ | 4096 |
| /56 | xxxx:xxxx:xxxx:xx:: | $2^{72}$ | 256 |
| /60 | xxxx:xxxx:xxxx:xxx:: | $2^{68}$ | 16 |
| /64 | xxxx:xxxx:xxxx:xxxx:: | $2^{64}$ | 1 |

A /64 holds 18,446,744,073,709,551,616 addresses



# *Switchports for Trunking and Access; related display commands*

switchport mode {access | trunk}
switchport access vlan vlan-number -- defines the VLAN interface resides in.
switchport trunk encapsulation [dot1q | isl] -- almost always use dot1q - you may have to do this first.
switchport mode dynamic auto | dynamic desirable
    dynamic auto - becomes a trunk if the neighboring interface is set to trunk or desirable- not if auto!
    dynamic desireable - becomes a trunk if neighboring interface is set to ANY trunk mode *(default!)*
switchport nonegotiate - prevents generating DTP frames, or converting dynamically to anything
switchport trunk allowed vlan 4,6,12,15 --this should eliminate vlans not specifically listed
switchport trunk allowed vlan [remove 4-8 | all | none]
no switchport trunk native vlan
no switchport trunk vlan 4
switchport trunk pruning vlan {add | except | none | remove} vlan-list  Specify VLANs eligible for pruning

**The proper way to remove VLANs from a switch**
If you delete a VLAN from a switch with "no vlan 30", it won't make it disappear from the interface configs that it was previously added to- it will remain in the swtchport configurations! (and cause confusion) First, run on those interfaces "switchport trunk allowed vlan remove 30' or "switchport trunk allowed vlan 10,20,40,50' Then run "no vlan 30" on the switch to remove it.

show interfaces [type,#] switchport - settings, status, trunking, access/voice/native VLAN
show interfaces [type,#] trunk - lists info on trunks (or the specific trunk) and the VLANs
show vlan brief, show vlan - each VLAN and all assigned interfaces, but no trunks!
show vlan [vlan] - access and trunk ports in the VLAN.
show vtp status - VTP mode, configuration and status info

*Expected Trunking Operational Mode Based on the Configured Administrative Modes w/ DTP*

| ***Administrative Mode*** | ***Access*** | ***Dynamic Auto*** | ***Trunk*** | ***Dynamic Desirable*** |
|---|---|---|---|---|
| **access** | Access | Access | Access | Access |
| **dynamic auto** | Access | Access | Trunk | Trunk |
| **trunk** | Access | Trunk | Trunk | Trunk |
| **dynamic desirable** | Access | Trunk | Trunk | Trunk |

**Port Security**
As soon as you enable port-security, it defaults to violation shutdown and a maximum of 1 MAC address.
switchport port-security
switchport port-security violation restrict
switchport port-security mac-address aa.bb.cc.dd.ee.ff [sticky, often used with maximum]
switchport port-security maximum value
    the max number of MAC addresses that can be assigned - default is one.
    A max value can also be set if it's a switch connected - recieving frames for multiple MACs
switchport port-security violation {protect | restrict | shutdown}
        protect: unauthorized frames would just be dropped
        restrict: authorized frames would be dropped and violations count toggled
        shutdown: disable the interface (err-disabled - this is the default action)
show port-security interface

To open an interface shut down with port-security, you first issue "shutdown", then "no shutdown"

**Inter-VLAN Routing (IVR) and Switched Virtual Interfaces (SVIs)**
VLANs can't bridge without a router or Inter-VLAN Routing (IVR). Implementing trunking and Inter-VLAN routing on a layer 3 switch uses Switched Virtual Interfaces (SVIs)

If each router interface is plugged into an access link, each of the routers' interfaces would be the default gateway address for each host in each respective VLAN. IVR in the "router on a stick" (ROAS) implementation puts VLANs into a trunk to a layer 3 device. which performs the routing on logical interfaces and send back out the trunk to the switch on the proper VLAN. This can be done without the external router with a layer 3 switch, which seems a bit more efficient, and relying on the one external router creates a potential bottleneck, as well as a single point of failure.

# *Basic VLAN Management*

Best practice dictates to always have separate voice VLANs, data VLANs, management VLAN, native VLAN, black hole VLAN, and the control plane VLAN (VLAN1). First, we need to discus what each of these are.

**"Default" VLAN and the Control Plane**

The default VLAN is simply the VLAN that all the ports on a switch are members of when a switch is reset to factory defaults. All switch ports are members of the default VLAN after the initial boot of the switch. Default really means exactly that and nothing more- factory defaults.

The default VLAN for Cisco switches is VLAN 1 and you cannot rename or delete it. However, since it defaults to being both Management and Native VLAN, best practices recommend moving those roles to other separate independent VLANs. After doing so, applying shutdown will prevent all data traffic from VLAN 1, so only control protocols are permitted on VLAN 1 (DTP, VTP ,STP BPDU's, PAgP, LACP, CDP, etc) Even if you prune VLAN 1 from trunks, these protocols always use VLAN 1 for controls communication.
Consider VLAN 1 to serve as a conduit only for Layer 2 control plane traffic, supporting no other traffic.

**Management VLAN**

A management VLAN is defined to access the management capabilities of a switch via HTTP, Telnet, SSH, but also includes SNMP and Syslog. It should be obvious why it's segregation is important security-wise

**Native VLAN - Untagged Frames**

Native VLAN is an 802.1q trunk concept for how to transport untagged traffic, and also serves as a common identifier on opposing ends of a trunk link. Traffic from access ports unassigned to a VLAN will be untagged, as well as some legacy LAN traffic. It should be separate and distinct from all other VLANs defined in the switched LAN

By default the native VLAN is always untagged, however there is a command available on some switches where you can tell the switch to tag all VLANs including the native

- Can be manually set on either end of the trunk, using the **switchport trunk native vlan** vlan-id
- If native VLANs differ on either end, it will accidentally cause frames to leave one VLAN and enter another.
  *So the number of Native VLAN's can be equal to the number of trunk ports if you have a REALLY messed up network!*
- Native has to do with the trunk itself, not the switch. We can only configure one native vlan per port. However, a switch with multiple ports, could have different vlans specified as the native vlan for that port. In other words, trunkport 2 could have a native vlan of 20, and trunkport 3 could have a native vlan of 30 (if you choose to be really complicated in your design). The switches on either end of the trunk just need to agree so strange things don't happen. [ This can get really weird: imagine mismatched native vlans on each end of a trunk. In that case, the native vlan on one side becomes part of the same broadcast domain of a differently labeled vlan of the other end ]

Consider: Switch A is connected to Switch B with a trunk. SA tags its native VLAN while SB doesn't, although it indeed uses that same VLAN number. Frames on the that vlan would flow properly from SA to SB, but not from SB to SA. While SB does not expect its traffic destined for its native vlan to be tagged, it will not reject it. Since SA is configured to expect traffic received for its native vlan to be tagged, it will discard the incoming untagged traffic. This is not quite a native vlan mismatch, but traffic will only flow in one direction.

The definition of a native vlan in 802.1q is indeed an untagged vlan. There are some exploits that can take advantage of this by stacking two sets of tags. If a user builds a frame that has an outer dot1q tag for a known native vlan and an inner tag of a vlan he wishes to attack, the first tag will be removed when the frame traverses the first trunk. The next trunk that is encountered will put the frame on the vlan that is the attack destination. As a result, there is a new command introduced to tag all frames on a trunk. This global command is "vlan tag dot1q native"

interface fa0/1
 **switchport mode access**
 **switchport access vlan 10**
 switchport voice vlan 20
 switchport trunk native vlan 30
Here PVID is 10, as untagged frames will get into VLAN 10.

interface fa0/2
 **switchport mode trunk**
 switchport access vlan 10
 switchport voice vlan 20
 **switchport trunk native vlan 30**
Here PVID is 30, as untagged frames will get into VLAN 30

Consider having a dedicated native vlan-id used on all trunks and never used on access ports to defeat double VLAN hopping attacks. On trunks, using a native vlan that is then not used can provide some L2 security advantages this makes all traffic on the trunk ports tagged. Another advantage is there is no concern about native VLAN mismatch if untagged frames are not allowed.

```
interface GigabitEthernet1/0/23
switchport mode trunk
switchport trunk encapsulation dot1q
switchport trunk native vlan 800
switchport trunk allowed vlan 252
```

Another way is to create (for example) VLAN 100 as the native vlan. then shut it down and make all trunk ports native vlan 100.

Finally, **vlan dot1q tag native** is a global command to tag native VLAN traffic, and admit only 802.1Q tagged frames on 802.1Q trunks, dropping any untagged traffic, including untagged traffic in the native VLAN.

**Black Hole VLAN - Suspended Ports**
A black hole (AKA parking or holding) VLAN is defined to assign all unused ports to it so that any device traffic connecting is not allowed on trunk links, thus preventing communicating beyond the switch. It is an extra way of ordering a port inoperable. The **state suspend** command in VLAN configuration mode will cause all received frames to be dropped. Syntax - state {active | suspend}

```
S1(config)#vlan 10
S1(config-vlan)#name Data
S1(config-vlan)#vlan 20
S1(config-vlan)#name Voice
S1(config-vlan)# vlan 30
S1(config-vlan)# name Native
S1(config-vlan)#vlan 99
S1(config-vlan)#name Management
S1(config-vlan)#vlan 5
S1(config-vlan)#name Suspended
S1(config-vlan)#state suspend
S1(config-vlan)#exit
S1(config)#interface vlan 99
S1(config-if)#ip add 192.168.128.10 255.255.255.0
S1(config-if)#no shut
S1(config-if)#exit
S1(config)#ip default-gateway  192.168.128.1
S1(config)#interface range  fa0/5-24
S1(config-if-range)#switchport mode access
S1(config-if-range)#switchport access vlan 5
S1(config-if-range)#shutdown
S1(config)#interface  fa0/1
S1(config-if)#switchport mode trunk
S1(config-if)#switchport trunk native vlan 30
S1(config-if)#no shutdown
S1(config)#interface range  fa0/2-4
S1(config-if-range)#switchport mode access
S1(config-if-range)#switchport access vlan 10
S1(config-if-range)#switchport voice vlan 20
S1(config-if-range)#no shutdown
```

# *VLAN Trunking Protocol (VTP)*

Designed as a method to manage VLANs across a large numbers of switches: addition, deletion, and renaming from a central point of control, and all switches participating can use any related VLANs. Add a VLAN on the server and it gets set up on others. Not used as much in modern networks, Cisco says best practice is having switches in "off" or "transparent" modes; you may never see it but may find it and want to disable or otherwise manage it.

A switch can belong to just one management domain, and there is no communication between domains.
VTP advertisements contain info about the domain itself (including revision number), it's VLANs and their info.

**VTP Modes: vtp mode {server | client | transparent | off}**

Server mode (is default)

- full control over management of it's domain. Each domain should have at least one server
- advertises updates to other switches in the domain, receives info to synchronize domain members
- The first server defined in a network defines the domain that will be used by future VTP servers and clients.
- Multiple VTP servers can coexist in a domain, and is recommended for redundancy.
- There is no election for primary or secondary server
- If one server is configured with a new VLAN or VTP parameter, other servers synchronize just as any client

Client mode:

- listen to VTP advertisements from other switches and modify configurations accordingly.
- forward/relay VTP messages out trunk links to neighboring switches in the domain

Transparent mode:

- do not participate in VTP, does not advertise or synchronize its VLAN database with received advertisements.
- can create and delete VLANs that are local only to itself without changes being advertised
- Works only as a relaying member.
- [VTP v1, transparent mode does not relay VTP info unless its domain and VTP version numbers match]
- [VTP v2 and 3, transparent mode does forward VTP advertisements regardless of the VTP domain name]

Off mode simply disables all VTP activity on the switch.

**Revision Numbers - watch out when adding hardware!**
VTP uses a revision number to track the most recent information.
Starts of at 0 and is incremented by the VTP server with each change in domain info it will advertise
An advertisement with a greater revision number than before says it has new and updated information.
That advertisement is stored and overwrites any previously stored VLAN information.

*Important!*

VTP VLAN data is saved in vlan.dat file in flash memory is retained even when the switch power is off.
It ensures a switch can recover last known VTP/VLAN configuration from its VTP database after it reboots. Even a device previously configured in client mode will send a summary advertisement using info from it's vlan.dat after powering up and discovering it has a higher revision number.
*- Care must be taken to not plug in a VTP enabled switch containing a higher revision number, as it will obliterate existing VTP database information throughout the domain it matches to!*

- Always force revision number 0 before being attaching *EVEN if previously configured as a only a VTP client*
- On a new device, set to VTP transparent and then later change back to server- or change the VTP domain to a bogus name
- For critical portions of your network, consider using VTP transparent or off mode to prevent synchronization problems
- Eliminate the chance for duplicate, overlapping VLANs in a large network with transparent mode. For example, two administrators might configure VLANs on switches in their respective areas but use the same VLAN identification or VLAN number. They could overlap if both administrators advertised them using VTP servers

Domains use unsecure advertisements (default), but a password can be required to participants (secure mode)

Cisco Catalyst switches - default is VTP mode server - Don't forget to disable!
It turns out that by default Cisco switches operate in VTP server mode for the management domain NULL (a blank string), no password. If it hears a VTP summary advertisement it automatically learns the VTP domain name, VLANs, revision number. This makes it easy to bring up a new switch in an existing VTP domain.
Be sure to remember that the new switch stays in VTP server mode until you change it..

**show vtp status**

```
Switch# show vtp status
VTP Version capable             : 1 to 3
VTP version running             : 1
VTP Domain Name                 :
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : aca0.164f.3f80
Configuration last modified by 0.0.0.0 at 0-0-00 00:00:00
Local updater ID is 0.0.0.0 (no valid interface found)
Feature VLAN:
--------------
VTP Operating Mode                : Server
Maximum VLANs supported locally   : 1005
Number of existing VLANs          : 5
Configuration Revision            : 0
MD5 digest                        : 0x57 0xCD 0x40 0x65 0x63 0x59 0x47 0xBD
                                    0x56 0x9D 0x4A 0x3E 0xA5 0x69 0x35 0xBC
```

**Advertisement types**

Summary advertisements

- VTP domain servers send every 300 seconds and every time a VLAN database change occurs
- VTP version, domain name, revision number, MD5 hash, and number of subset advertisements to follow.
- When config changes, one or more subset advertisements with more details are sent afterwards

Subset advertisements

- list specific changes: add/ delete/ suspend/ activate a VLAN, changing of VLAN name, number, MTU.
- Even change in VLAN type (such as Ethernet or Token Ring), or security association identifier (SAID- 802.10)
- each VLAN gets it's own individual sequential subset advertisement per change, as needed.

VTP advertisements are multicast 01-00-0C-CC-CCCC and an SNAP type value of 0×2003.

Cisco switches default to v1. Versions are not fully backward compatible with each other
Versions 1 and 2 support VLAN numbers 1 to 1005. Only VTP v3 supports extended VLAN range 1-4094.

VTP v3 Features

Extended VLAN range VLANs 1 through 4094 can be advertised throughout a VTPv3 domain
Enhanced authentication - the password can be hidden (only a hash of the password is saved in the running configuration) or secret (the password is saved in the running configuration).
Database propagation - databases other than VTP can be advertised
By default, all VTPv3 switches operate as secondary servers and can send updates throughout the domain.
A primary server is only needed to take control of a domain.
Per-port VTP - VTPv3 can be enabled on a per-trunk port basis, rather than a switch as a whole

```
Switch(config)# vtp version 3
Switch(config)# vtp domain MyCompany
Switch(config)# vtp mode server
Switch(config)# vtp password bigsecret
```

**VLAN Pruning**

VTP pruning makes more efficient use of trunk bandwidth by reducing unnecessary flooded traffic.
Broadcast, multicast, and unknown unicast frames on a VLAN are forwarded over a trunk link only if the switch on the receiving end of the trunk has ports in that VLAN.
When a switch has an active port associated with a VLAN, the switch advertises that to its neighbor switches.
The neighbors then to decide whether flooded traffic from a VLAN should be allowed on certain trunk links.

Even when VTP pruning has determined that a VLAN is not needed on a trunk, an instance of the Spanning Tree will run for every VLAN that is **allowed** on the trunk link. To reduce the number of STP instances, you should manually “prune” unneeded VLANs from the trunk and allow only the needed ones. Use the **switchport trunk allowed vlan** command to identify the VLANs that should be added or removed from a trunk.

VTP pruning is disabled by default. To enable pruning, use **vtp pruning**
On a VTP server, it says pruning needs to be enabled for the entire domain (all will also enable pruning.
When pruning is enabled, all general-purpose VLANs become eligible for pruning on all trunk links, if needed. However, you can modify the default list of pruning eligibility with the following interface-configuration command:

**switchport trunk pruning vlan {{{add | except | remove} vlan-list} | none}**

| | |
|---|---|
| vlan-list | List of eligible VLAN numbers (2-1001), separate by commas or dashes, no spaces. |
| add | Will add to the already configured list |
| except | All VLANs are eligible except for the VLAN numbers |
| remove | VLAN numbers to remove from the already configured list |
| none | No VLAN will be eligible for pruning. |

Obviously, VTP pruning has no effect on switches in the VTP transparent mode.
Those switches must be configured manually to “prune” VLANs from trunk links (same command)
VLAN 1 is never eligible for pruning, same with 1002-1005 (reserved for Token Ring and FDDI VLANs)

Troubleshooting VTP - Not updating information from a VTP server?
- Is the switch in VTP transparent mode
- If a VTP client, there might not be an VTP server. In this case, just make it a VTP server itself.
- Is the link to the VTP server a trunk? VTP advertisements are sent only over trunks.
- Is the VTP domain name is configured to match the one on the VTP server.
- Check if the VTP version is compatible matches the VTP domain.
- Does the VTP domain use a password? If the server doesn't, make sure the password is disabled or cleared.

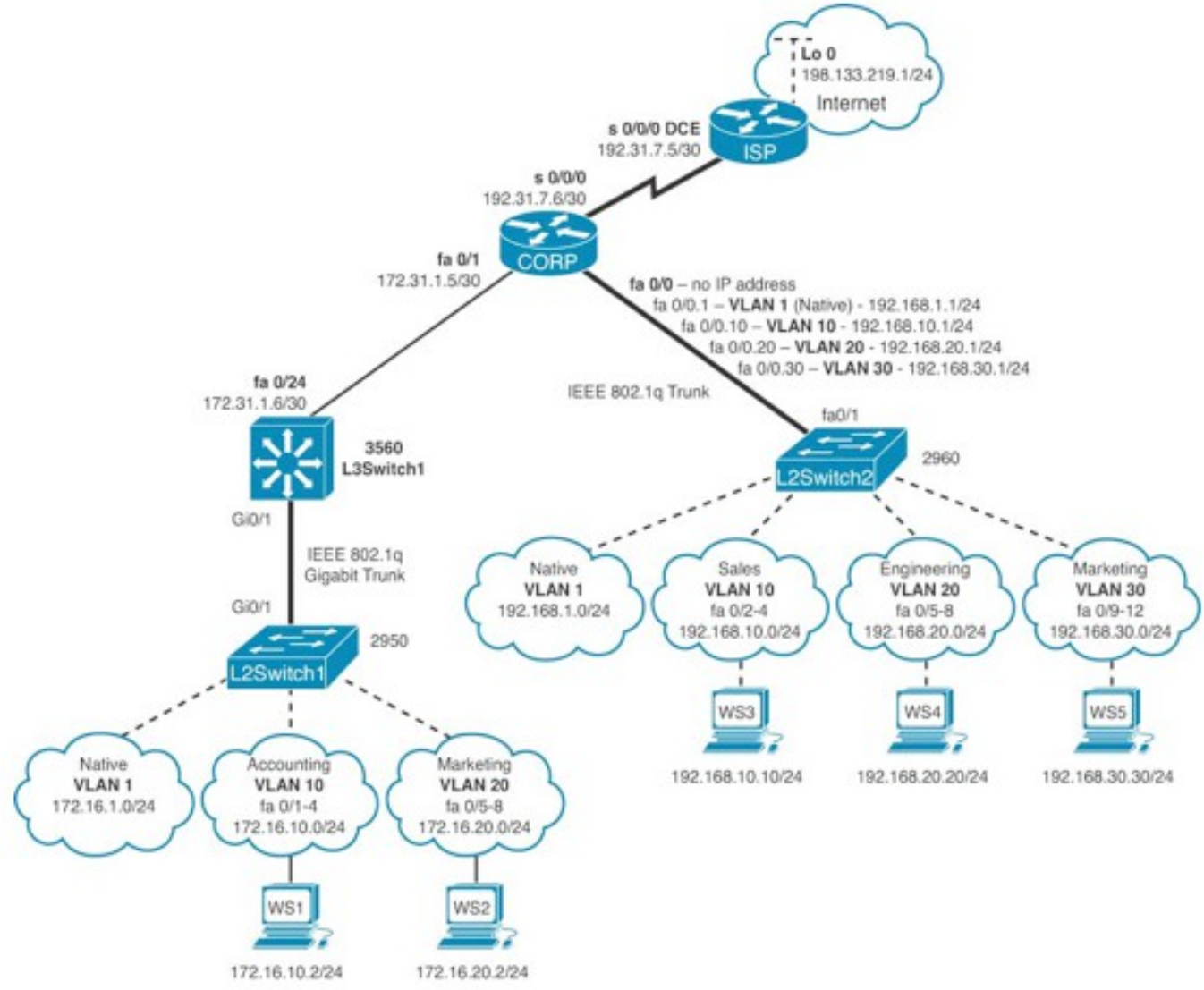
**VTP Troubleshooting Commands**

| | |
|---|---|
| show vtp status | Current VTP parameters, incl. the last VTP server |
| show vlan brief | Displays defined VLANs |
| show interface type member/module/number switchport | Displays trunk status, including pruning eligibility |
| show interface type member/module/number pruning | Displays VTP pruning state |

**VTP Configuration Commands**

| | |
|---|---|
| vtp domain domain-name | Define the VTP domain. |
| vtp mode {server \| client \| transparent \| off} | Set the VTP mode. |
| vtp password password [hidden \| secret] | Define an optional VTP password. |
| vtp version {1 \| 2 \| 3} | Configure VTP version. |
| vtp pruning | Enable VTP pruning. |
| switchport trunk pruning vlan {add \| except \| none \| remove} vlan-list | Specify VLANs eligible for pruning |

## *Configuration Example- Inter-VLAN Communication - pp113, Empson*




**ISP Router**
Router>enable.
Router>#configure terminal
Router(config)#hostname ISP

ISP(config)#interface loopback 0
ISP(config-if)#description simulated address for remote website
ISP(config-if)#ip address 198.133.219.1 255.255.255.0
ISP(config-if)#interface serial 0/0/0
ISP(config-if)#description WAN link to the Corporate Router
ISP(config-if)#ip address 192.31.7.5 255.255.255.252
ISP(config-if)#clock rate 56000 *--DCE cable so don't forget the clock rate*
ISP(config-if)#no shutdown

ISP(config-if)#router eigrp 10 *--- initialize EIGRP for AS*
ISP(config-router)#network 198.133.219.0
ISP(config-router)#network 192.31.7.0
ISP(config-router)#no auto-summary
ISP#copy running-config startup-config

**CORP Router**
Router>enable.
Router>#configure terminal
Router(config)#hostname CORP

```
CORP(config)#interface serial 0/0/0
CORP(config-if)#description link to ISP Router
CORP(config-if)#ip address 192.31.7.6 255.255.255.252
CORP(config-if)#no shutdown
CORP(config)#interface fastethernet 0/1
CORP(config-if)#description link to 3560 Switch
CORP(config-if)#ip address 172.31.1.5 255.255.255.252
CORP(config-if)#no shutdown
CORP(config)#interface fastethernet 0/0
CORP(config-if)#description link to L2Switch2
CORP(config-if)#duplex full  ---full-duplex to ensure trunking will take effect with L2Switch2.
CORP(config-if)#no shutdown
CORP(config-if)#interface fastethernet 0/0.1  ---Creates a virtual subinterface
CORP(config-subif)#description Management VLAN 1 – Native VLAN
CORP(config-subif)#encapsulation dot1q 1 native
CORP(config-subif)#ip address 192.168.1.1 255.255.255.0
CORP(config-subif)#interface fastethernet 0/0.10
CORP(config-subif)#description Sales VLAN 10
CORP(config-subif)#encapsulation dot1q 10
CORP(config-subif)#ip address 192.168.10.1 255.255.255.0
CORP(config-subif)#interface fastethernet 0/0.20
CORP(config-subif)#description Engineering VLAN 20
CORP(config-subif)#encapsulation dot1q 20
CORP(config-subif)#ip address 192.168.20.1 255.255.255.0
CORP(config-subif)#interface fastethernet 0/0.30
CORP(config-subif)#description Marketing VLAN 30
CORP(config-subif)#encapsulation dot1q 30
CORP(config-subif)#ip add 192.168.30.1 255.255.255.0
CORP(config-subif)#exit
CORP(config-if)#exit
CORP(config)#router eigrp 10
CORP(config-router)#network 192.168.1.0   ---Advertise our networks
CORP(config-router)#network 192.168.10.0
CORP(config-router)#network 192.168.20.0
CORP(config-router)#network 192.168.30.0
CORP(config-router)#network 172.31.0.0
CORP(config-router)#network 192.31.7.0
CORP(config-router)#no auto-summary
```

**L2Switch2 (Catalyst 2960)**

```
Switch(config)#hostname L2Switch2
L2Switch2(config)#vlan 10
L2Switch2(config-vlan)#name Sales  ---Assign names to the VLANs
L2Switch2(config)#vlan 20
L2Switch2(config-vlan)#name Engineering
L2Switch2(config-vlan)#vlan 30
L2Switch2(config-vlan)#name Marketing
L2Switch2(config)#interface range fastethernet 0/2 - 4   ---Assign access ports to the VLANs
L2Switch2(config-if-range)#switchport mode access
L2Switch2(config-if-range)#switchport access vlan 10
L2Switch2(config-if-range)#interface range fastethernet 0/5 - 8
L2Switch2(config-if-range)#switchport mode access
L2Switch2(config-if-range)#switchport access vlan 20
L2Switch2(config-if-range)#interface range fastethernet 0/9 - 12
L2Switch2(config-if-range)#switchport mode access
L2Switch2(config-if-range)#switchport access vlan 30
L2Switch2(config-if-range)#exit
L2Switch2(config)#interface fastethernet 0/1
L2Switch2(config-if)#description Trunk Link to CORP Router
L2Switch2(config-if)#switchport mode trunk
L2Switch2(config-if)#exit
L2Switch2(config)#interface vlan 1  ---Creates virtual interface for VLAN 1
L2Switch2(config-if)#ip address 192.168.1.2 255.255.255.0
L2Switch2(config-if)#no shutdown
L2Switch2(config-if)#exit
L2Switch2(config)#ip default-gateway 192.168.1.1  ---Assigns the default gateway address.
L2Switch2(config)#exit
L2Switch2#copy running-config startup-config
```

**L3Switch1 (Catalyst 3560)**

```
Switch(config)#hostname L3Switch1
L3Switch1(config)#vtp mode server  ---Changes the switch to VTP server mode.
L3Switch1(config)#vtp domain testdomain  ---Configures the VTP domain name to testdomain.
L3Switch1(config)#vlan 10
L3Switch1(config-vlan)#name Accounting
L3Switch1(config-vlan)#exit
L3Switch1(config)#vlan 20
L3Switch1(config-vlan)#name Marketing
L3Switch1(config-vlan)#exit
L3Switch1(config)#interface gigabitethernet 0/1
L3Switch1(config-if)#description Gigabit Trunk to L2Switch1
L3Switch1(config-if)#switchport trunk encapsulation dot1q
L3Switch1(config-if)#switchport mode trunk
L3Switch1(config-if)#exit
L3Switch1(config)#ip routing  ---Enables IP routing on this device.
L3Switch1(config)#interface vlan 1  ---Creates a virtual interface for VLAN 1
L3Switch1(config-if)#ip address 172.16.1.1 255.255.255.0
L3Switch1(config-if)#no shutdown
L3Switch1(config-if)#interface vlan 10
L3Switch1(config-if)#ip address 172.16.10.1 255.255.255.0
L3Switch1(config-if)#no shutdown
L3Switch1(config-if)#interface vlan 20
L3Switch1(config-if)#ip address 172.16.20.1 255.255.255.0
L3Switch1(config-if)#no shutdown
L3Switch1(config-if)#exit
L3Switch1(config)#interface fastethernet 0/24
L3Switch1(config-if)#description Link to CORP
L3Switch1(config-if)#no switchport  ---Creates a Layer 3 port on the switch.
L3Switch1(config-if)#ip address 172.31.1.6 255.255.255.252
L3Switch1(config-if)#exit
L3Switch1(config)#router eigrp 10
L3Switch1(config-router)#network 172.16.0.0
L3Switch1(config-router)#network 172.31.0.0
L3Switch1(config-router)#no auto-summary
L3Switch1(config-router)#exit
L3Switch1(config)#exit
L3Switch1#copy running-config startup-config
```

**L2Switch1 (Catalyst 2960)**

```
Switch(config)#hostname L2Switch1
L2Switch1(config)#vtp domain testdomain  ---VTP domain name to testdomain.
L2Switch1(config)#vtp mode client  ---Changes the switch to VTP client mode.
L2Switch1(config)#interface range fastethernet 0/1 - 4
L2Switch1(config-if-range)#switchport mode access
L2Switch1(config-if-range)#switchport access vlan 10
L2Switch1(config-if-range)#interface range fastethernet 0/5 - 8
L2Switch1(config-if-range)#switchport mode access
L2Switch1(config-if-range)#switchport access vlan 20
L2Switch1(config-if-range)#exit
L2Switch1(config)#interface gigabitethernet 0/1
L2Switch1(config-if)#switchport mode trunk
L2Switch1(config-if)#exit
L2Switch1(config)#interface vlan 1
L2Switch1(config-if)#ip address 172.16.1.2 255.255.255.0  --VLAN1 on diagram is 172.16.1.0/24 - typo?
L2Switch1(config-if)#no shutdown
L2Switch1(config-if)#exit
L2Switch1(config)#ip default-gateway 172.16.1.1
L2Switch1(config)#exit .
L2Switch1#copy running-config startup-config
```

# ***Spanning Tree (802.1D)***

Spanning Tree is a managed system where each port is in a forwarding or blocking state. Blocking state means the port doesn't process any frames except STP messages- the switch physically receives the frame on blocked port, but ignores it. If topology changes (a switch goes down), STP convergence updates port states accordingly. This system has a "root bridge" and uses bridge IDs for each switch containing a priority # and MAC address. Distance to the root switch/bridge are measured as a "root cost" from each switch and port, and governs the forwarding and blocking of ports, and ultimately, the paths Layer 2 stuff can take around the network.

**The Root Bridge/ Root Switch and the System's Basics**

- The root switch is always the designated switch on all directly connected segments
- All ports on the root switch are designated ports, in a forwarding state
- Nonroot switches have a root port (with lowest cost to the root switch); if directly connected, it's the root port.
- Each nonroot switch has at least one designated port (DP) to the other non-root switches.
- Similarly, a designated switch is chosen by lowest "root cost" among other paths to root, other non-root switches connect to the root through their neighboring designated switch.
- Throughout this system of non-root switches, up/up ints become either DPs or blocked
- The designated port's job is to forward STP messages (BDPUs) back and forth from the root

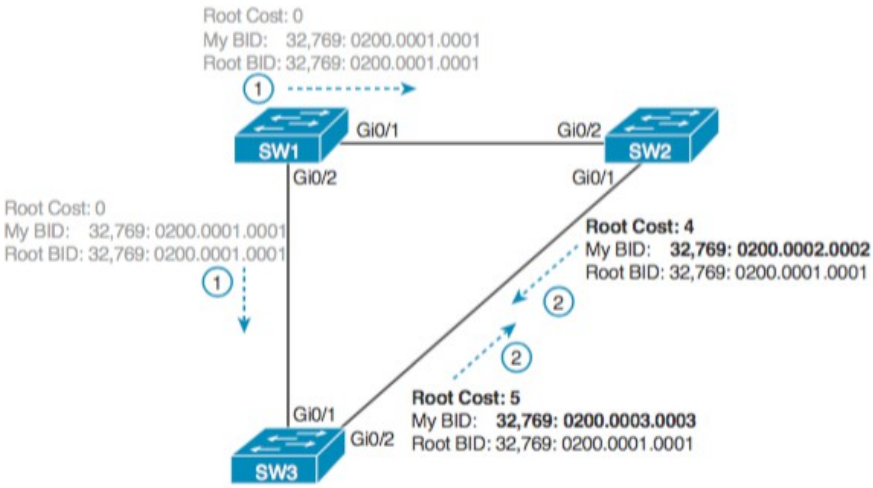
**Electing the Root Switch [more explanation in Bridge ID (BID) section]**

Root switch has the lowest value priority field in their BID

If SW1 Priority = 4096 and SW2 Priority = 8192, then SW1 is Root

If a priority tie occurs, switch with lowest MAC address field in BID is Root

If SW1 Pri= 4096 & MAC= 0200.xxxx.xxxx and SW2 Pri= 4096 & MAC= 0911.xxxx.xxxx, SW1 is Root




**Bridge Protocol Data Units (BPDU) exchange info among switches**

Hello messages are a BPDU containing:

- Root Bridge ID, Sender's Bridge ID and root cost
- Timer values on the root switch [hello, MaxAge, and forward delay timers]

Configuration BPDUs originate from Root Bridge. These are the majority of BPDUs on a healthy network

Topology Change Notification (TCN) BPDUs sent to the Root Bridge to alert that active topology has changed. (notification and acknowledgement subtypes)

**Timers**

Hello - 2 seconds default - Time period between recieving hellos created by the root

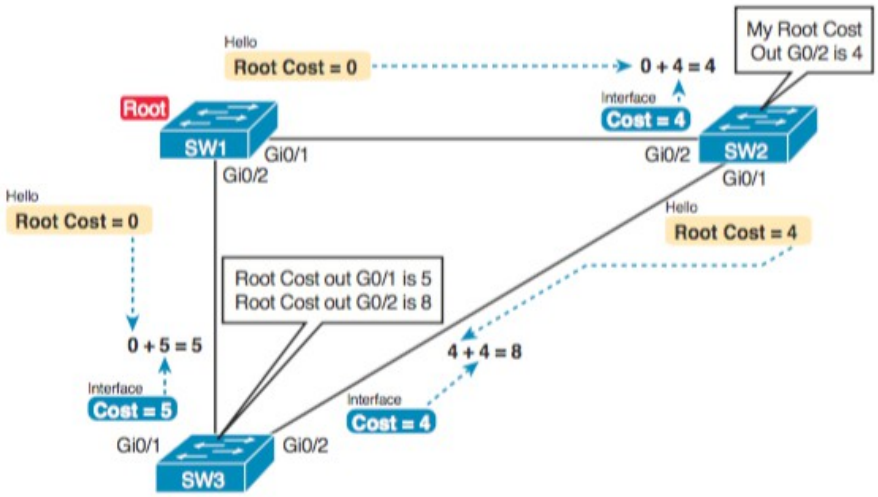
MaxAge- 10x hello timer (20 sec default) -How long switch waits after no hello BDPUs, before changing states

Forward delay- 15 secs - time interface goes from blocking to forwarding in each of the listening, learning states

**Electing Designated Ports - the port with the lowest-cost hello on a LAN segment.**

(Hello msgs have a cost of 0 when they leave the root switch) Cost by line type, 2nd # is "long path cost":

| | | |
|---|---|---|
| 10 Mbps | 100 | 2000000 |
| 100 Mbps | 19 | 200000 |
| 1 Gbps | 4 | 20000 |
| 10 Gbps | 2 | 2000 |
| 100 Gbps | | 200 |




*Both SW2 and SW3 list their respective cost to the root switch (cost 4 on SW2 and cost 5 on SW3). SW2 lists the lower cost, so SW2's Gi0/1 port is the designated port on that LAN segment.* ***The order is lowest root cost, then lowest BID, then lowest neighbor port priority, then finally, lowest neighbor internal port number***

**50-second convergence delay:** STP has the interface in both *learning and listening states* for a time equal to the forward delay timer, which defaults to 15 seconds each. A convergence event that causes an interface to change from blocking to forwarding requires 30 sec to transition those PLUS the MaxAge to first move from blocking

**Temporary states when going from blocking to forwarding:**
- Listening- doesn't forward- does MAC table cleanup
- Learning- also doesn't forward- learn the MAC address of received frames

| State | Forwards? | Learns MACs? | Transitory or Stable State? |
|---|---|---|---|
| Blocking | No | No | Stable |
| Listening | No | No | Transitory |
| Learning | No | Yes | Transitory |
| Forwarding | Yes | Yes | Stable |
| Disabled | No | No | Stable |

**Rapid STP (IEEE 802.1w - RSTP):** (is backward-compatible w/ non-RSTP switches)
- Calls blocking "discarding", no listening state. Only has learning, forwarding or discarding states
- Hello time = 2 secs, Max Age = 6 secs (3 x hello, 3 missed BDPUs)
***- improves convergence from about 50 seconds to about 10 seconds***
- adds two port types allowed to immediately enter the forwarding state rather than passively wait for the network to converge. In addition to forwarding, root, designated, and disabled:
    - Alternate port - best alternate path to the root bridge if there is a failure on the designated port
    - Backup port. Applies to scenarios with a hub. Antiquated.

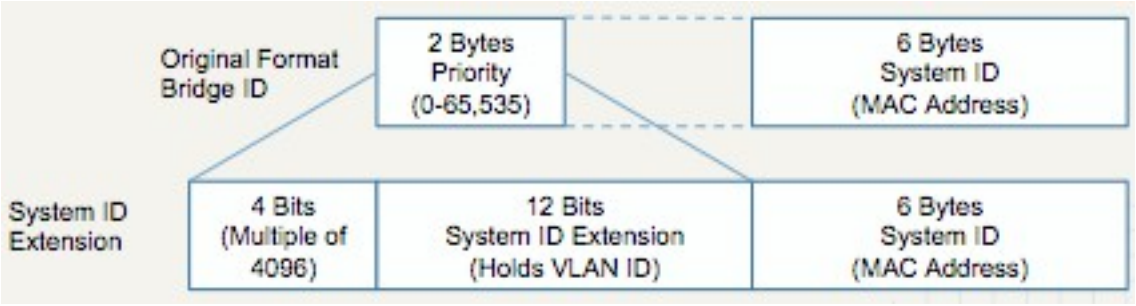
**Check Before Connecting New HW**
When plugged into the network, switches send out hello BPDUs listing their own BID as the root BID, then, if it hears a hello that lists a better (lower) BID, that switch stops advertising itself as root and starts forwarding the superior hello. This is how an unauthorized or improperly vetted device can screw up a STP topology- if one shows up with a lower BID, convergence spreads it as a new root switch and everything goes haywire.

**PortFast**
- Allows immediately change from blocking to forwarding. No listening or learning states
- For endpoint/edge devices (PCs) - NOT bridges/switches
- *Don't put on interfaces receiving BPDU (not another switch) - w/o BDPU Guard can create loops*
- If voice VLAN set, PortFast turns on automatically- will stay on even if voice VLAN is unset

**BPDU Guard**
- BDPU Guard disables a port if any BPDUs are received on it and prevents PortFast problems
- STP opens up the following security exposures:
    Attacker adds switch with low STP priority and becomes the Root Switch
    Attacker could plug into multiple ports/switches, become root, & forward (or TCPDump) LAN traffic
    Users can harm the LAN when they connect a non-STP switch (can be counted in elections, etc.)

**The Bridge ID - { [ Priority(4096-61440) + VLAN ID(1-4094) ] + the MAC address}**
STP Bridge ID (BID) is a unique 8-byte value- basically a priority field slapped onto a MAC address
- a 2-byte priority field The last 12 bits holds a VLAN ID 0-4096 (it's just basically a 16-bit number)
- a 6-byte system ID - the 48-bit MAC address

The 2-byte Priority field is separated into 2 parts
4-bit priority field; the part that can be changed with the "priority" directive
12-bit *system ID extension* - basically, just space for the VLAN ID)




*When the bits are written out, those leading 4 bits reside starting at the 4096 bit (the 13th bit), so each number starting there would naturally be a multiple of 4096. Since VLAN IDs can range from 1 to 4094, it all makes perfect sense and looks a lot less complicated like this:*

[ Priority 4096-61440 + VLAN 1-4094 ] + tack on the MAC address.
Means, [0000 + 000000000000] + your MAC address
0000 - [111111111111] <---- 4095 in binary (enough bits reserved to plop a "priotrity" onto)
*We configure the priority field below in square brackets:*
[1000] - 000000000000 <---- Default base priority of 32768 in binary
[0110] - 000000000000 <---- Secondary root bridge designation number in binary
[????] - 000000000000 <-----you can set a multiple here for a priority (from 4096 x ? up to 61440)

**The Easy Way**
**- spanning-tree vlan** vlan-id **root primary** [auto-set this vlan's priority low enough to become root now]
Cisco switches use a default base priority of 32,768
If current root base priority > 24,576, make base priority 24,576.
If 24,576 or lower, sets base priority to highest multiple of 4096 that results in becoming root.
For our numbers below, 28,672 is 0110 000000000000

**- spanning-tree vlan** vlan-id **root secondary** [again, this auto-sets it for us]
A priority value worse than the primary switch but better than all the other switches.
Sets the switch's base priority to 28,672 *regardless* of the current root's priority value.
For our numbers below, 28,672 is 0111 000000000000

**The Hard Way**
It is much easier to just designate using root primary and secondary to avoid doing it yourself. If you insist on doing "spanning-tree vlan vlan-id priority x":

**- spanning-tree vlan** vlan-id **priority** x
A switch configured with VLANs 1 through 4, with a default base priority
of 32,768, has a default STP priority of 32,769 in VLAN 1, 32,770 in VLAN 2, 32,771 in VLAN 3, and so on. So, you can view the 16-bit priority as a base priority (as configured on the **spanning-tree vlan** vlan-id **priority** x command) plus the VLAN ID."
0001:000000000001 = 4097
0010:000000000001 = 8193
0100 = 16384, 1000 = 32768

Vlan 1 would be 32769, Vlan 2 32770, Vlan 10 32778, Vlan 100 32868
1111= 61,440 <--- if there are more than one bit turned on, things get weird. One trick is to pretend the VLAN-ID part is all 1's so it flattens out to multiples of 4096 only, when trying to calculate stuff)
*Final word: If binary makes you uncomfortable, forget it. Use "root primary" and "root secondary"*

***Determining the Root Switch and the Root Port on Nonroot Switches***

Use **show spanning-tree**, **show spanning-tree root** to rule out any switches that have an RP, because root switches do not have an RP. **show spanning-tree** identifies the local switch as root directly: "This switch is the root". In **show spanning-tree root**, the RP column is empty if the local switch is the root.

Using **show spanning-tree vlan** x on a few switches, and recording the root switch, RP, and DP ports can quickly show you most STP facts. Chase the RPs. If starting with SW1, and SW1's G0/1 is an RP, try the switch on the other end of SW1's G0/1 port.

***STP Tiebreakers When Choosing the Root Port***

*The three tiebreakers are, in the order: lowest neighbor bridge ID, lowest neighbor port priority, finally, lowest neighbor internal port number. Only root paths that tie are considered when thinking about tiebreakers.*

Below, SW3 is not root and that its two paths to reach the root tie with their root costs of 8. The first tiebreaker is the lowest neighbor's BID. SW1's BID value is lower than SW2's, so SW3 chooses its G0/1 interface as its RP.

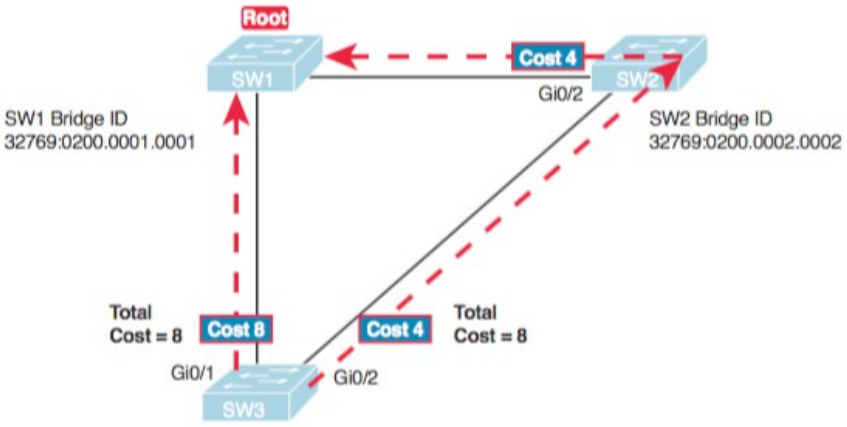



**Figure 2-8** *SW3's Root Cost Calculation Ends in a Tie*

The last two RP tiebreakers come into play only when two switches connect to each other with multiple links, as shown in above. In that case, a switch receives hellos on more than one port from the same neighboring switch, so the BIDs tie.

So, SW2 becomes root, and SW1 needs to choose its RP. SW1's root cost over each path will tie at 19. SW2 sends hellos over each link to SW1, so SW1 cannot break the tie based on SW1's neighbor BID because both list SW2's BID. So, SW1 has to turn to the other two tiebreakers.

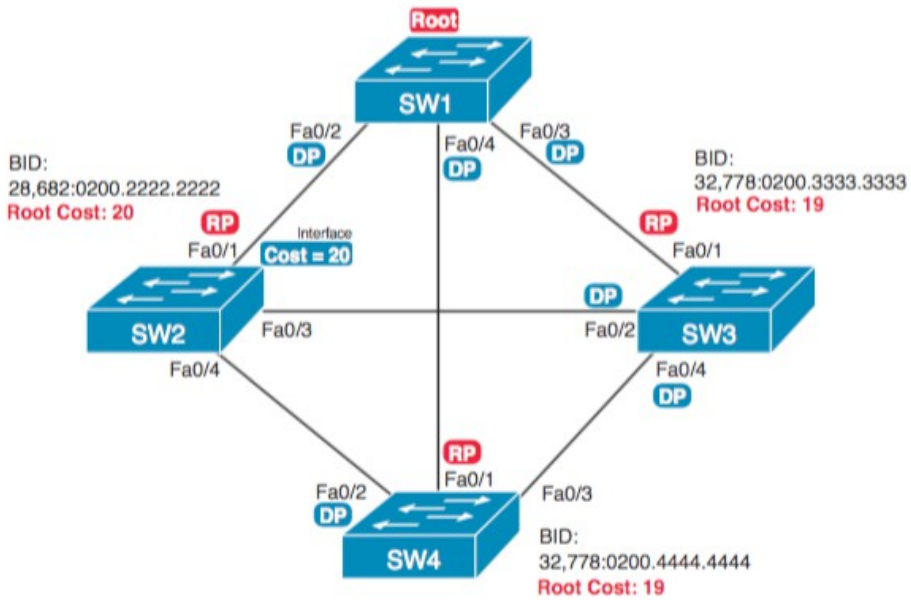

**Figure 2-9** *Topology Required for the Last Two Tiebreakers for Root Port*

The next tiebreaker is configurable: the neighboring switch's port priority on each neighboring switch interface. Cisco switch ports default to a setting of 128, with a range of values from 0 through 255, with lower being better (as usual). Here, SW2's F0/16 was manually set with **spanning-tree vlan 10 port-priority 112**. SW1 learns that the neighbor has a port priority of 112 on the top link and 128 on the bottom, so SW1 uses its top (F0/14) interface as the root port.

If the port priority ties, which it often does due to the default values, STP relies on an internal port numbering on the neighbor. Cisco switches assign an internal integer to identify each interface on the switch. The nonroot looks for the neighbor's lowest internal port number (as listed in the hello messages) and chooses its RP based on the lower number.

With Fa0/1 having the lowest number, then Fa0/2, then Fa0/3, and so on- SW2's Fa0/16 would have a lower internal port number than Fa0/17; SW1 would learn those numbers in the hello; and SW1 would use its Fa0/14 port as its RP. (In real life, most engineers would put these two links into an EtherChannel)

This figure notes the root, RPs, and DPs and each switch's least cost to reach the root over its respective RP.




*Remember the order of the criteria:*
*Lowest root cost, lowest BID, lowest neighbor port priority, lowest neighbor internal port number*

Focus on the segments that connect the nonroot switches for a moment.
**SW2–SW4 segment:** SW4 wins because of its root cost of 19, compared to SW2's root cost of 20.
**SW2-SW3 segment:** SW3 wins because of its root cost of 19, compared to SW2's root cost of 20.
**SW3-SW4 segment:** SW3 and SW4 tie on root cost, both with 19. SW3 wins due to its better (lower) BID value.
SW2 loses and does not become DP on the links to SW3 and SW4 even though SW2 has the better (lower) BID value. The first DP criteria is the lowest root cost, and SW2's root cost happens to be higher than SW3's and SW4's.

---

**show spanning-tree vlan** Bridge and RootID info, timers; port roles for local ports, switch role for device.
**show spanning-tree vlan 10 root** - lists root's BID for each VLAN, local switch's root cost and root port
**show spanning-tree vlan 10 bridge** - lists the local switch's BID, MAC, timers
**debug spanning-tree events**

**spanning-tree portfast** - **spanning-tree bpduguard enable** (to be run on individual interfaces)
Default is no portfast and no BPDU Guard. Change the opposite to make all on the new default:
**spanning-tree portfast default** - **spanning-tree portfast bpduguard default** (Global config)
**spanning-tree portfast disable** - **spanning-tree bpduguard disable** (Run per interface config)

UplinkFast: pg 197 - UplinkFast: Access Layer Uplinks
BackboneFast pg 198 - BackboneFast: Redundant Backbone Paths

**Commands for Spanning Tree - PortFast - BDPUGuard - UplinkFast - BackboneFast**

| | |
|---|---|
| **spanning-tree** vlan-id | Enable STP. |
| **spanning-tree mode** { **pvst \| rapid-pvst \| mst** } | Set the STP mode. |
| **spanning-tree vlan** vlan-number **root primary** | Changes this switch to the root. |
| **spanning-tree vlan** vlan-number **root secondar y** | Sets this switch's STP base |
| **spanning-tree** [**vlan** vlan-id] {**priority** priority} | Changes the bridge priority of this switch for this VLAN. |
| **spanning-tree** [**vlan** vlan-number] **cost** cost | Changes the STP cost . |
| **spanning-tree** [**vlan** vlan-number] **port-priority** priority | Changes STP port priority in VLAN (0 to 240, count by 16). |
| **spanning-tree vlan** vlan-id **priority** bridge-priority | Set bridge priority. |
| **spanning-tree vlan** vlan-id **root** {**primary** \| **secondary**} [**diameter** diameter] | Set root bridge (macro). |
| **spanning-tree** [**vlan** vlan-id] **hello-time** seconds<br>**spanning-tree** [**vlan** vlan-id] **forward-time** seconds<br>**spanning-tree** [**vlan** vlan-id] **max-age** seconds | Set STP timers. |
| **show spanning-tree** | STP info all VLANs, ports summarized |
| **show spanning-tree detail** | STP info all VLANs, ports detailed. |
| **show spanning-tree interface** interface-id | Lists STP info for the specified port |
| **show spanning-tree vlan** vlan-id | Lists STP info for the specified VLAN |
| **show spanning-tree** [vlan vlan-id] **summary** | Show switch ports currently in each of the STP states. |
| **show spanning-tree** [**vlan** vlan-id] **root** | Show a VLAN's root bridge ID, root port, and root path cost. |
| **show spanning-tree** [**vlan** vlan-id] **bridge** | Show this switch's bridge ID and STP timers |
| **show spanning-tree interface** type number **portfast** | 1-line status message about PortFast |
| **channel-group** channel-group-number **mode** {**auto** \| **desirable** \| **active** \| **passive** \| **on**} | Enables EtherChannel. |
| **show etherchannel** [chan-grp-#] {**brief** \| **detail** \| **port** \| **port-channel** \| **summary**} | Info on state of EtherChannels |
| **debug spanning-tree events** | Provide info messages about changes in the STP topology |
| **spanning-tree portfast (disable)** | Enables/disables PortFast. |
| **spanning-tree bpduguard enable \| disable** | Enables/disables BPDU Guard |
| **spanning-tree portfast default** | Changes switch default for PortFast to enabled. |
| **spanning-tree portfast bpduguard default** | Changes switch default for BPDU Guard to enabled. |
| **spanning-tree uplinkfast** [**max-update-rate** pkts-per-sec] | Set UplinkFast on a switch. |
| **spanning-tree backbonefast** | Set BackboneFast on a switch. |
| **show spanning-tree uplinkfast** | Show the STP UplinkFast status. |
| **show spanning-tree backbonefast** | Show the STP BackboneFast status. |
| **show spanning-tree summary** | Display global BPDU Guard, BPDU filter, and Loop Guard states. |
| **show spanning-tree interface** type mod/num [**detail**] | Look for detailed reasons for inconsistencies. |
| **show spanning-tree inconsistentports** | List ports labeled in an inconsistent state. |
| **show udld** [type mod/num] | Display the UDLD status on one or all ports. |
| **udld reset** | Reenable ports that UDLD aggressive mode errdisabled. |

| **Task** | **Global Command Syntax** | **Interface Command Syntax** |
|---|---|---|
| Enable Root Guard. | -- | **spanning- tree guard root** |
| Enable BPDU Guard. | **spanning-tree portfast bpduguard default** | **spanning- tree bpduguard enable** |
| Enable Loop Guard. | **spanning-tree loopguard default** | **spanning- tree guard loop** |
| Enable UDLD. | **udld {enable \| aggressive \| message time}** | **udld {enable \| aggressive \| disable}** |
| Enable BPDU filtering. | **spanning-tree bpdufilter default** | **spanning- tree bpdufilter enable** |

Switch(config-if)**# spanning-tree portfast** Define an edge port
Switch(config-if)**# spanning-tree link-type point-to-point** Override a port type
Switch(config)**# spanning-tree mode mst** Enable MST on a switch
Switch(config)**# spanning-tree mst configuration** Enter MST configuration mode
Switch(config-mst)**# name** name Name the MST region.
Switch(config-mst)**# revision** version Set the configuration revision number

**spanning-tree mst** instance-id **root {primary | secondary**} [**diameter** diameter] Set root bridge (macro).
**spanning-tree mst** instance-id **priority** bridge- priority Set bridge priority.
**spanning-tree mst** instance-id **cost** cost Set port cost.
**spanning-tree mst** instance-id **port-priority** port-priority Set port priority.

Set STP timers.
**spanning-tree mst hello-time** seconds
**spanning-tree mst forward-time** seconds
**spanning-tree mst max-age** seconds

**STP and RTSP Port Path Cost values**

| **Data rate** | **STP 802.1D-1998** | **RSTP 802.1W-2004** |
|---|---|---|
| Base Metric | **1 Gigabit per sec** | **2 Terabit per sec** |
| Calculation | **1,000,000/(*N* Kb/s)** | **20,000,000,000/ (*N* Kb/s)** |
| | (1998 is nonlinear) | |
| | (1990 was linear) | |
| <=100 Kb/s | | 200,000,000 |
| 1 Mbit/s | | 20,000,000 |
| 4 Mbit/s | 250 | 5,000,000 |
| 10 Mbit/s | 100 | 2,000,000 |
| 16 Mbit/s | 62 (was 63) | 1,250,000 |
| 45 Mbit/s | 39 (was 22) | 444,445 |
| 100 Mbit/s | 19 (was 10) | 200,000 |
| 155 Mbit/s | 14 (was 6) | 129,032 |
| 622 Mbit/s | 6 (was 2) | 32,154 |
| 1 Gbit/s | 4 (was 1) | 20,000 |
| 2 Gbit/s | 3 | 10,000 |
| 10 Gbit/s | 2 | 2,000 |
| 100 Gbit/s | | 200 |
| 1 Tbit/s | | 20 |
| 10 Tbit/s | | 2 |

**802.1W-2004:** Limiting the range of the Path Cost parameter to 1–200 000 000 ensures that the accumulated Path Cost cannot exceed 32 bits over a concatenation of 20 hops
The grey cells indicate values presented by Cisco in the Switch 300-115 Official Cert Guide and are not in the IEEE 802.1W-2004.
Cisco also says in the CIEE 5.0 Routing and Switching book the short format is Cisco's, and not IEEE.
Indeed, in IEEE 802.1W-2004, it is the long format which remains (ed.: I am thankful since a nonlinear format is nonsensical)

# *Switching: EtherChannel*

*Cisco's Port Aggregation Protocol (PAgP) and Link Aggregation Control Protocol (LACP - 802.3ad)*
- Combines multiple parallel segments of equal speed into an EtherChannel (single interface)
- Multiple segments of equal speed up to 8 links. If one fails, the rest stay up.
- Prevents STP convergence when only a single failure occurs; with at least one up, no STP convergence
- Aggregated links provide some load balancing on switches (and better use of bandwidth)

Create the port-channel interface, exit, then enter interface config mode for ports and add them with a mode:

```
Cat2950(config)# interface port-channel 1   <exit>
Cat2950(config)# interface fa0/2
Cat2950(config-if)# channel-group 1 mode on
```

PAgP: **channel-group 1 mode {desirable | auto}**
LACP: **channel-group 1 mode {active | passive}**
Manually run with **channel-group** 1 **mode on** (has no encapsulation; set "on" on both ends)
- One side must be either desirable or active to get the other end to begin negotiations (like trunks)
- Number must remain the same on the current device. Another device could call it 4 or 5

*Three terms as synonyms: EtherChannel, PortChannel, and Channel-group.*
The **channel-group** configuration command
**show etherchannel** shows status of a “PortChannel” (instead of EtherChannel or Channel-group)
**show spanning-tree** also says Port-channel

Misconfiguration of channel-group commands:
- All members of a switch's specific etherchannel need to be using the same channel-group number.
- The neighboring switches can refer to that etherchannel as any number they want.
- If using the "on" keyword, it has to be on the other end too (and remember no encaps'ing).
- If using the "desirable" keyword, it's PAgP and the other must use either "desirable" or "auto".
- If using "active" keyword, it's LACP; the other must use either "active" or "passive".
- No "auto" or "passive" keyword on both switches! Then both wait on the other to begin negotiations.
- Don't mix LACP and PAgP commands on one end or the other- the protocols need to match.

**show etherchannel** [chan-grp-#] {**brief** | **detail** | **port** | **port-channel** | **summary**}

```
SW1# show etherchannel 1 summary
Flags:  D - down        P - bundled in port-channel
        I - stand-alone s - suspended
        H - Hot-standby (LACP only)
        R - Layer3      S - Layer2
        U - in use      f - failed to allocate aggregator
        M - not in use, minimum links not met
        u - unsuitable for bundling
        w - waiting to be aggregated
        d - default port
Number of channel-groups in use: 1
Number of aggregators:           1
Group  Port-channel  Protocol    Ports
1      Po1(SU)           -        Fa0/14(P)   Fa0/15(P)
```

The **show etherchannel summary** command. The D code letter means that the channel itself is down, with S meaning a Layer 2 EtherChannel. The bottom shows Portchannel (Po1) as L2 EtherChannel in a down state (SD)
"Down" only refers to the EtherChannel state - the two physical interfaces are still connected (**sh int status**)

**Configuration Checks for EtherChannel**
Things need to match or be compatible. Watch for common sense stuff (no shutdown, speed, duplex)
- Operational access or trunking state (all must be access, or all must be trunks)
- On an access port, check the access VLAN;
- Trunk port? Check allowed VLAN list - **switchport trunk allowed** - also check the native VLAN
- Check STP interface settings (ie, cost)

For example, if you change the STP port cost on one of the interfaces but not the other you get this:
ERR_Disable: channel-misconfig (STP) error detected on Po1, putting Fa0/14 in err-disable state
ERR_Disable: channel-misconfig (STP) error detected on Po1, putting Fa0/15 in err-disable state
ERR_Disable: channel-misconfig (STP) error detected on Po1, putting Po1 in err-disable state
Make them match and it should bring everything back up again

***EtherChannel Stuff from CCNP***

Load balancing uses a hashing operation on either MAC or IP addresses and can be based solely on source or destination addresses, or both. To configure frame distribution for all EtherChannel switch links:

Switch(config)**# port-channel load-balance** *method*

The method is set with a global configuration command. It's set globally for the switch, not on a per-port basis.

**Table 10-3** Types of EtherChannel Load-Balancing Methods

| Method Value | Hash Input |
|---|---|
| src-ip | Source IP |
| dst-ip | Destination IP |
| src-dst-ip | Source and destination IP |
| src-mac | Source MAC |
| dst-mac | Destination MAC |
| src-dst-mac | Source and destination MAC |
| src-port | Source port number |
| dst-port | Destination port number |
| src-dst-port | Source and destination port number |

- All hash operations are performed on bits, except src-dst combo methods use XOR
- Port number only specified in CCNP Switch book as available in models 4500, 6500

Switch**# show etherchannel load-balance**
EtherChannel Load-Balancing Configuration:
src-mac

EtherChannel Load-Balancing Addresses Used Per-Protocol:
Non-IP: Source MAC address
IPv4: Source MAC address
IPv6: Source MAC address

To verify efficiency of load-balancing method use the **show etherchannel port-channel** command

**EtherChannel Troubleshooting Commands**

| | |
|---|---|
| Current EtherChannel status of each member port | **show etherchannel summary**<br>**show etherchannel port** |
| Time stamps of EtherChannel changes | **show etherchannel port-channel** |
| Detailed status about each EtherChannel component | **show etherchannel detail** |
| Load-balancing hashing algorithm | **show etherchannel load-balance** |
| Load-balancing port index used by hashing algorithm | **show etherchannel port-channel** |
| EtherChannel neighbors on each port | **show** {**pagp** \| **lacp**} **neighbor** |
| LACP system ID | **show lacp sys-id** |

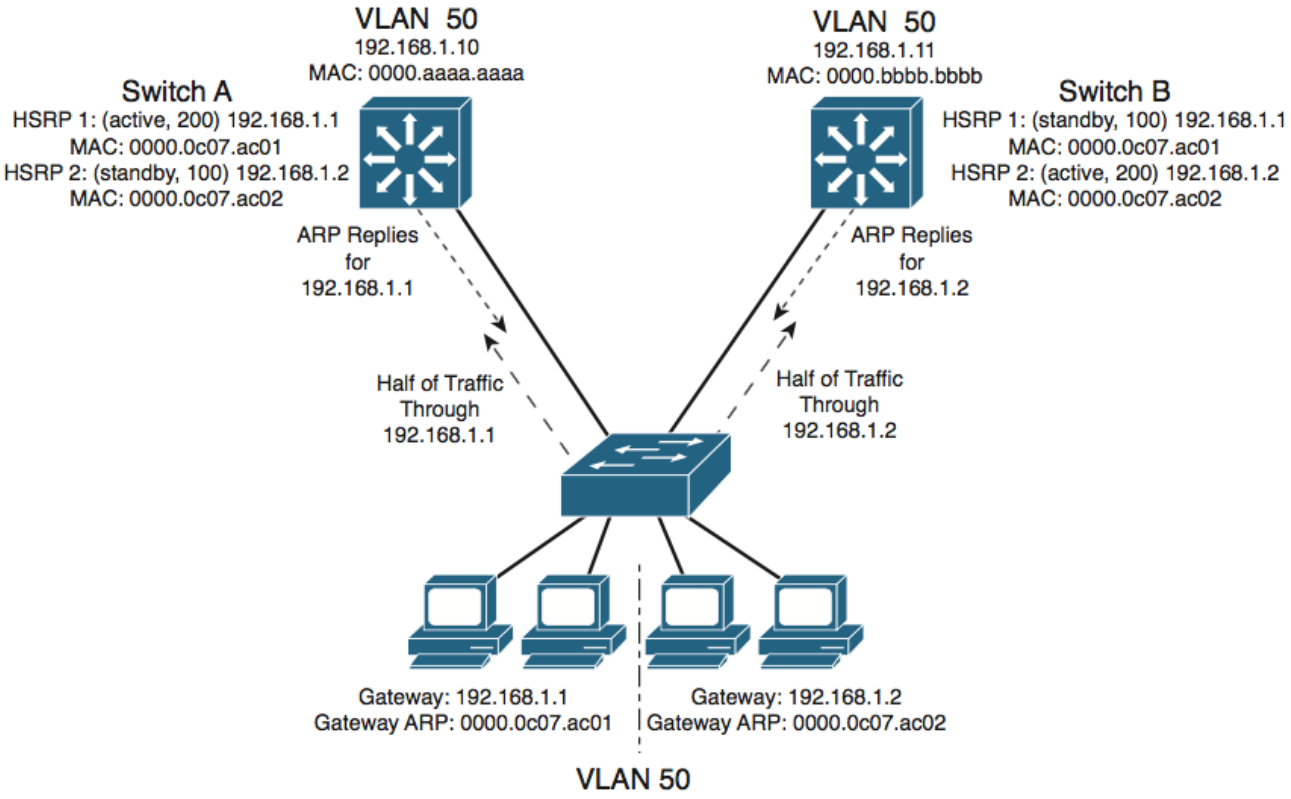
**EtherChannel Configuration Commands**

| | |
|---|---|
| Select load-balancing method for switch. | **port-channel load-balance** method |
| Use a PAgP mode on an interface. | **channel-protocol pagp**<br>**channel-group** number **mode** {**on \|** {{**auto** \| **desirable**} [**non-silent**]} |
| Assign the LACP system priority. | **lacp system-priority** priority |
| Use an LACP mode on an interface. | **channel-protocol lacp**<br>**channel-group** number **mode** {**on \| passive \| active**}<br>**lacp port-priority** priority |
| Configure EtherChannel Guard | [**no**] **spanning-tree etherchannel guard misconfig** |

From SVI section
Keep in mind that a Layer 3 port assigns a network address to one specific physical interface. If several interfaces are bundled as an EtherChannel, the EtherChannel can also become a Layer 3 port. In that case, the network address is assigned to the port-channel interface—not to the individual physical links within the channel.

# *HSRP and VRRP Load Balancing Example*
### *[ two HSRP groups, one VLAN ]*

Load balancing traffic across two uplinks to two HSRP routers with a single HSRP group is not possible.
We must use two HSRP groups where each group assigns an active router to one of the switches
- Make each switch function as the standby router for its partner's HSRP group.
- (Each router is active for one group and standby for the other group)
- Switch A is active router for Group 1 (192.168.1.1), and standby for Group 2 (192.168.1.2).
- Switch B is configured similarly, but with its roles reversed.
- Configure half of the PCs with the Group 1 virtual router address and the other half with the Group 2 address.
- Each half of the hosts uses one switch as its default gateway over one uplink

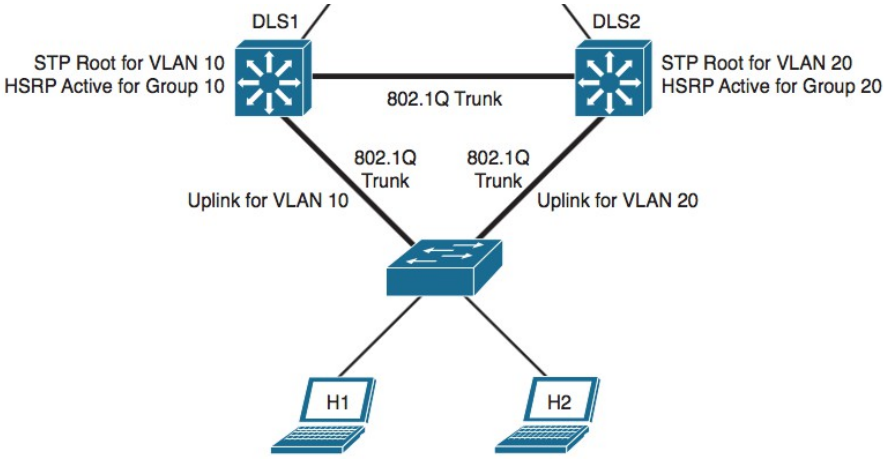



| **HSRP Version** | **VRRP Version (minus authentication)** |
|---|---|
| Switch-A(config)# **interface vlan 50**<br>Switch-A(config-if)# **ip address 192.168.1.10 255.255.255.0**<br>Switch-A(config-if)# **standby 1 priority 200**<br>Switch-A(config-if)# **standby 1 preempt**<br>Switch-A(config-if)# **standby 1 ip 192.168.1.1**<br>Switch-A(config-if)# **standby 2 priority 100**<br>Switch-A(config-if)# **standby 2 ip 192.168.1.2** | Switch-A(config)# **interface vlan 50**<br>Switch-A(config-if)# **ip address 192.168.1.10 255.255.255.0**<br>Switch-A(config-if)# **vrrp 1 priority 200**<br>Switch-A(config-if)# **vrrp 1 ip 192.168.1.1**<br>Switch-A(config-if)# **vrrp 2 priority 100**<br>Switch-A(config-if)# **no vrrp 2 preempt**<br>Switch-A(config-if)# **vrrp 2 ip 192.168.1.2** |
| Switch-B(config)# **interface vlan 50**<br>Switch-B(config-if)# **ip address 192.168.1.11 255.255.255.0**<br>Switch-B(config-if)# **standby 1 priority 100**<br>Switch-B(config-if)# **standby 1 ip 192.168.1.1**<br>Switch-B(config-if)# **standby 2 priority 200**<br>Switch-B(config-if)# **standby 2 preempt**<br>Switch-B(config-if)# **standby 2 ip 192.168.1.2** | Switch-B(config)# **interface vlan 50**<br>Switch-B(config-if)# **ip address 192.168.1.11 255.255.255.0**<br>Switch-B(config-if)# **vrrp 1 priority 100**<br>Switch-B(config-if)# **no vrrp 1 preempt**<br>Switch-B(config-if)# **vrrp 1 ip 192.168.1.1**<br>Switch-B(config-if)# **vrrp 2 priority 200**<br>Switch-B(config-if)# **vrrp 2 ip 192.168.1.2** |

## *HSRP Load Balancing Example 2*

***[ two HSRP groups, two VLANs, Spanning Tree roots (primary and secondary ]***

- configure DLS1 as STP root, HSRP active for VLAN 10 - also backup root and HSRP standby for VLAN 20
- configure DLS2 as STP root, HSRP active for VLAN 20 - also backup root and HSRP standby for VLAN 10.
- The configuration for DLS2 below is basically the opposite of what DLS1 has.




STP forwarding to the correct VLANs ensured by setting spanning-tree primary root for those VLANs.

| | |
|---|---|
| DLS1(config)**#spanning-tree vlan 10 root primary** | Configures STP root primary for VLAN 10. |
| DLS1(config)**#spanning-tree vlan 20 root secondary** | Configures STP root secondary for VLAN 20. |
| DLS1(config)**#interface vlan10** | Moves to interface configuration mode. |
| DLS1(config-if)**#ip address 10.1.10.2 255.255.255.0** | Assigns IP address and netmask. |
| DLS1(config-if)**#standby 10 ip 10.1.10.1** | Activates HSRP group 10, virtual IP of 10.1.10.1 |
| DLS1(config-if)**#standby 10 priority 110** | Assigns a priority value of 110 to standby group 10. This will be the active forwarder for VLAN 10. |
| DLS1(config-if)**#standby 10 preempt** | Switch will take control of VLAN 10 forwarding if local priority is higher than active switch VLAN 10 priority. |
| DLS1(config-if)**#interface vlan20** | Moves to interface configuration mode. |
| DLS1(config-if)**#ip address 10.1.20.2 255.255.255.0** | Assigns IP address and netmask. |
| DLS1(config-if)**#standby 20 ip 10.1.20.1** | Activate HSRP group 20, virtual IP address 10.1.20.1. |
| DLS1(config-if)**#standby 20 priority 90** | Assigns a priority value of 90 to standby group 20. This switch will be the standby device for VLAN 20. |
| DLS1(config-if)**#standby 20 preempt** | This switch will take control of VLAN 20 forwarding if local priority is higher than active switch for VLAN 20 |

| | |
|---|---|
| DLS2(config)**#spanning-tree vlan 20 root primary** | Configures STP root primary for VLAN 20. |
| DLS2(config)**#spanning-tree vlan 10 root secondary** | Configures STP root secondary for VLAN 10. |
| DLS2(config)**#interface vlan20** | Moves to interface configuration mode. |
| DLS2(config-if)**#ip address 10.1.20.3 255.255.255.0** | Assigns IP address and netmask. |
| DLS2(config-if)**#standby 20 ip 10.1.20.1** | Activates HSRP group 10, virtual IP of 10.1.20.1 |
| DLS2(config-if)**#standby 20 priority 110** | Assigns a priority value of 110 to standby group 20. This will be the active forwarder for VLAN 10. |
| DLS2(config-if)**#standby 20 preempt** | Switch will take control of VLAN 20 forwarding if local priority is higher than active switch VLAN 20 priority. |
| DLS2(config-if)**#interface vlan10** | Moves to interface configuration mode. |
| DLS2(config-if)**#ip address 10.1.10.3 255.255.255.0** | Assigns IP address and netmask. |
| DLS2(config-if)**#standby 10 ip 10.1.10.1** | Activate HSRP group 10, virtual IP address 10.1.10.1. |
| DLS2(config-if)**#standby 10 priority 90** | Assigns a priority value of 90 to standby group 10. This switch will be the standby device for VLAN 10. |
| DLS2(config-if)**#standby 10 preempt** | This switch will take control of VLAN 20 forwarding if local priority is higher than active switch for VLAN 10 |

***HSRP Load Balancing Example 1 (cont.)***
***( output of "show standby" command )***

```
Switch-A# show standby vlan 50 brief
                     P indicates configured to preempt.
                     |
Interface   Grp Prio P State    Active addr     Standby addr    Group addr
Vl50        1   200  P Active   local           192.168.1.11    192.168.1.1
Vl50        2   100    Standby  192.168.1.11    local           192.168.1.2
Switch-A#
Switch-A# show standby vlan 50
Vlan50 - Group 1

  Local state is Active, priority 200, may preempt
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 2.248
  Virtual IP address is 192.168.1.1 configured
  Active router is local
  Standby router is 192.168.1.11 expires in 9.860
  Virtual mac address is 0000.0c07.ac01
  Authentication text "MyKey"
  2 state changes, last state change 00:11:58
  IP redundancy name is "hsrp-Vl50-1" (default)
Vlan50 - Group 2
  Local state is Standby, priority 100
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 1.302
  Virtual IP address is 192.168.1.2 configured
  Active router is 192.168.1.11, priority 200 expires in 7.812
  Standby router is local
  Authentication text "MyKey"
  4 state changes, last state change 00:10:04
  IP redundancy name is "hsrp-Vl50-2" (default)
```

***Switch B is the same, just flipped.***

*VRRP Load Balancing example - output of "show" commands*

**Table 18-3** *Verifying VRRP Status for multiple VRRP Groups*

**Switch A**

```
Switch-A# show vrrp
Vlan50 - Group 1
  State is Master
  Virtual IP address is 192.168.1.1
  Virtual MAC address is 0000.5e00.0101
  Advertisement interval is 1.000 sec
  Preemption is enabled
    min delay is 0.000 sec
  Priority is 200
  Authentication is enabled
  Master Router is 192.168.1.10 (local),
    priority is 200
  Master Advertisement interval is 1.000
    sec
  Master Down interval is 3.218 sec

Vlan50 - Group 2
  State is Backup
  Virtual IP address is 192.168.1.2
  Virtual MAC address is 0000.5e00.0102
  Advertisement interval is 1.000 sec
  Preemption is disabled
  Priority is 100
  Authentication is enabled
  Master Router is 192.168.1.11, priority
    is 200
  Master Advertisement interval is 1.000
    sec
  Master Down interval is 3.609 sec
(expires in 2.977 sec)
Switch-A#
```

**Switch B**

```
Switch-B# show vrrp
Vlan50 - Group 1
State is Backup
Virtual IP address is 192.168.1.1
Virtual MAC address is 0000.5e00.0101
Advertisement interval is 1.000 sec
Preemption is disabled
Priority is 100
Authentication is enabled
Master Router is 192.168.1.10, priority
is 200
Master Advertisement interval is 1.000
sec
Master Down interval is 3.609 sec
(expires in 2.833 sec)

Vlan50 - Group 2
State is Master
Virtual IP address is 192.168.1.2
Virtual MAC address is 0000.5e00.0102
Advertisement interval is 1.000 sec
Preemption is enabled
min delay is 0.000 sec
Priority is 200
Authentication is enabled
Master Router is 192.168.1.11 (local),
priority is 200
Master Advertisement interval is 1.000
sec
Master Down interval is 3.218 sec
Switch-B#
```

```
Switch-A# show vrrp brief
Interface          Grp Pri Time  Own Pre State   Master addr     Group addr
Vlan50             1   200 3218        Y  Master  192.168.1.10    192.168.1.1
Vlan50             2   100 3609           Backup  192.168.1.11    192.168.1.2
Switch-A#

Switch-B# show vrrp brief
Interface          Grp Pri Time  Own Pre State   Master addr     Group addr
Vlan50              1   100 3609           Backup  192.168.1.10    192.168.1.1
Vlan50              2   200 3218        Y  Master  192.168.1.11    192.168.1.2
Switch-B#
```

## IEEE Standard: VRRP - Virtual Router Redundancy Protocol (RFC 2338)

Almost the same as HSRP. Here, "master" takes the place of "active" in HSRP terms, and the rest are termed "backup." The VRRP preempt option is enabled by default (the switch that is the IP address owner will preempt, regardless of this explicit setting). Default preempt delay is 0 sec (easily extended)
- Group number can be from 1- 255; Priority range 1-254. The default is 100.
- The virtual router MAC is of the form 0000.5e00.01xx, where xx is a two-digit hex group number.
- Sends advertisements to multicast 224.0.0.18 (VRRP), using IP protocol 112
- Uses a timer for advertisements by the virtual switch master.
- All switches in a group must use the same timer values, or the group will not communicate.
- Default interval value is 1 sec, range is 1-255 sec. If you use the msec argument, you change the timer globally to measure in milliseconds. The range then is 50 to 999ms.

| | |
|---|---|
| Switch(config)**#interface vlan10** | (all of these are interface config mode) |
| **ip address 172.16.100.5 255.255.255.0** | Assigns IP address and netmask to VLAN interface. |
| **vrrp 10 ip 172.16.100.1** | Enables VRRP group 10 virtual IP of 172.16.100.1 (using a real interface IP here will make the router with that address become the master) |
| **vrrp 10 description Sales Group** | Assigns a text description to the group. |
| **vrrp 10 priority 110** | Sets the priority level for this VLAN. |
| **vrrp 10 preempt** | This switch will take over as virtual switch master for group 10 if it has a higher priority than the current one |
| **vrrp 10 preempt delay minimum 60** | This switch will preempt, only after a delay of 60 sec |
| **vrrp 10 timers advertise 15** | Configures the interval between successful advertisements by the virtual switch master. |
| **vrrp 10 timers learn** | Configures a virtual switch backup, to learn the advertisement interval from the virtual switch master. |
| (**no**) **vrrp 10 shutdown** | Disables/ re-enables VRRP on the interface, in a way that the configuration is still retained. |
| **no vrrp 10 shutdown** | Reenables the VRRP group using the previous configuration. |
| **vrrp 10 authentication text ottawa** | Plain-text auth for group 10 using the key **ottawa**. |
| **vrrp 10 authentication md5 key-string winnipeg** | MD5 auth for group 10 using the key **winnipeg**. |

**~~Interface~~ Object Tracking**
VRRP does not have a native interface tracking mechanism, but even better, has the ability to track objects. This allows the VRRP master to lose its status if a tracked object (interface, IP SLA, and so on) fails.

| | |
|---|---|
| **track 10 interface fastethernet0/0 line-protocol** | Creates a tracked object, where the status of the uplink interface is tracked |
| **interface fastethernet0/1** | |
| **vrrp 1 track 10 decrement 30** | Track previously created object and decrease the VRRP priority by 30 should the uplink interface fail |

**Verifying VRRP (same for IPv6 and IPv4)**

| | |
|---|---|
| Switch**#show vrrp** | Displays VRRP information |
| Switch**#show vrrp brief** | Displays a brief status of all VRRP groups |
| Switch**#show vrrp 10** | Displays detailed information about VRRP group 10 |
| Switch**#show vrrp interface vlan10** | Displays info about VRRP enabled on int Vlan10 |
| Switch**#show vrrp interface vlan10 brief** | Displays a brief summary about VRRP on int Vlan10 |

**Debugging VRRP**

| | |
|---|---|
| Switch**#debug vrrp all** | Displays all VRRP messages |
| Switch**#debug vrrp error** | Displays all VRRP error messages |
| Switch**#debug vrrp events** | Displays all VRRP event messages |
| Switch**#debug vrrp packets** | Displays messages about packets sent and received |
| Switch**#debug vrrp state** | Displays messages about state transitions |

VRRP is only partially supported on some Cisco hardware. Verify VRRP capabilities by platform datasheets and appropriate Cisco IOS command and configuration guides.

## Cisco Proprietary: HSRP - Hot Standby Redundancy Protocol

| | |
|---|---|
| **interface vlan10** | Make VLAN an interface - activate switch virtual interface (SVI) |
| **ip address 172.16.0.10 255.255.255.0** | Assigns IP address and netmask. |
| **standby 1 ip 172.16.0.1** [**secondary**] | Activates HSRP group 1 and creates virtual IP address of 172.16.0.1<br>If interface has secondary IP addresses, you can add **secondary** so HSRP has a redundant secondary gateway address. |
| **standby 1 priority 120** | Assigns a priority value of 120 to standby group 1. |

- Group number can be from 0 to 255. The default is 0.
- Some Catalyst switches limit to 16 unique group numbers- just make the group number the same (that is, 1) for every VLAN interface. HSRP groups are locally significant only on an interface: HSRP Group 1 on interface VLAN 10 is unique and independent from HSRP Group 1 on interface VLAN 11.
- The actual interface address and the virtual (standby) address must be configured to be in the same subnet.
- Priority value 1-255, default is 100. A higher priority will result in that switch being elected the active switch.
- If priorities of all switches in the group are equal, switch with highest IP address becomes active switch.
- HSRP hello messages - multicast 224.0.0.2 (“all routers”) - UDP port 1985.
- Only the standby router (with the second-highest priority) monitors hello messages from the active router.
- Decreasing hello time allows a router failure to be detected more quickly, yet increases traffic on the interface.

Router interface states before becoming active:

Disabled **=>** Init**=>** Listen **=>** Speak **=>** Standby **=>** Active

- HSRP defines a virtual MAC address in the form 0000.0c07.acxx, where xx represents the group number as a two-digit hex value. (Group 1 appears as 0000.0c07.ac01, Group 16 appears as 0000.0c07.ac1, etc.)

| | |
|---|---|
| **show standby** | Displays HSRP information |
| **show standby brief** | Displays single-line output summary of each standby group |
| **show standby vlan 1** | Displays HSRP information on the VLAN 1 group |

**Defaults for HSRP**

| | |
|---|---|
| HSRP version | Version 1 (v1 and v2 have different packet structure) |
| HSRP groups | None configured. |
| Standby group number | 0 |
| Standby MAC address | System assigned as 0000.0c07.ac*XX*, where *XX* is the HSRP group number. For HSRPv2, the MAC address will be 0000.0C9F.FXXX. |
| Standby priority | 100 |
| Standby delay | 0 (no delay) |
| Standby track interface priority | 10 |
| Standby hello time | 3 seconds |
| Standby holdtime | 10 seconds |

**Preempt -** switch will take control of the active switch if local priority is higher than priority of active switch

| **interface vlan10** | |
|---|---|
| **standby 1 preempt** | Designate this switch to preempt |
| **standby 1 preempt delay minimum 180 reload 140** | Set to preempt 180 sec since that switch was last restarted or 140 sec since switch was last reloaded. |
| **standby delay minimum 30 reload 60** | Delay for HSRP group initialization 30 sec when interface comes up and 60 sec after switch reloads. |
| **no standby 1 preempt delay** | Disables the preemption delay |
| **no standby 1 preempt** | Disable the preempt option completely |

If a router is not already active, it cannot become active again until the current active router fails- even if its priority is higher than that of the active router. When routers are just being powered up or added to a network, the first router to bring up its interface becomes the HSRP active router, even if it has the lowest priority of all. This is where setting up preempt comes to the rescue.

- Use **reload** to force router to wait after it has been reloaded or restarted before preempt (you should consider and allow time for routing protocols (e.g.) that need time to converge). Use of **minimum** only refers to time after interface is ready for HSRP

**Timers** - Hello timer 1-254, default is 3; Hold timer 1-255, default is 10. The default unit of time is seconds.

| **interface vlan10** | |
|---|---|
| **standby 1 timers 5 15** | Sets the hello to 5 sec and hold to 15 sec |
| | Hold normally set to be ≧ 3X hello |
| **standby 1 timers msec 200 msec 600** | Sets hello to 200 milliseconds, hold to 600 msec. |

If the **msec** argument is used, the timers can be an integer from 15 to 999

**Track -** assigns a value that the priority will be decreased if the tracked interface goes down

| Switch(config)**#interface vlan10** | |
|---|---|
| Switch(config-if)**#standby 1 track f0/0 25** | HSRP will track the availability of interface FastEthernet0/0. If it goes down, the priority of the switch in group 1 will be decremented by 25. |

Default value of the **track** is 10. In the example, assuming default priority of 100, the new priority will be 75. Using track facilitates in election eligibility when failures occur and replacement candidates must be considered.

**Authentication**

| Switch(config)**#key chain HSRP** | Creates authentication key chain called **HSRP**. |
|---|---|
| Switch(config-keychain)**#key 1** | Adds a first key to the key chain. |
| Switch(config-keychain-key)**#key-string australia** | Configures a key string of **australia**. |
| Switch(config)**#interface vlan10** | |
| Switch(config-if)**#standby 1 authentication text canada** | Configures **canada** as plain-text authentication string used by group 1. |
| Switch(config-if)**#standby 2 authentication md5 key-string england** | Configures **england** as MD5 key string for group 2. |
| Switch(config-if)**#standby 3 authentication md5 key-chain HSRP** | Configures MD5 using key chain **HSRP**. Queries key chain for current live key and key ID. |

**HSRPv2 for IPv6**

HSRPv2 must be enabled on an interface before HSRP for IPv6 can be configured.
When configuring the IPv6 virtual address, if an IPv6 global address is used, it must include an IPv6 prefix length. If a link-local address is used, it does not have a prefix

| **standby version 2** | Enables HSRPv2 on an interface |
|---|---|
| **standby 1 ipv6 autoconfig** | Use a virtual link-local address that will be generated automatically from the link-local prefix and a modified EUI-64 format interface identifier, where the EUI-64 interface identifier is created from the relevant HSRP virtual MAC address |
| **standby 1 ipv6 FE80::1:1** | Use an explicitly configured link-local address to be used as the virtual IPv6 address for group 1 |
| **standby 1 ipv6 2001::0DB8:2/64** | Use a global IPv6 address as the virtual address for group 1 |

All other relevant HSRP commands (preempt, priority, authentication, tracking, and so on) are identical in HSRPv1 and HSRPv2.

**Debugging HSRP**

| **debug standby** | All HSRP debugging information, including state changes and transmission/ reception of HSRP packets |
|---|---|
| **debug standby terse** | All HSRP errors, events, and packets, except for hellos and advertisements |
| **debug standby errors** | Displays HSRP error messages |
| **debug standby events** | Displays HSRP event messages |
| **debug standby events terse** | Displays all HSRP events except for hellos and advertisements |
| **debug standby events track** | Displays all HSRP tracking events |
| **debug standby packets** | Displays HSRP packet messages |

## Cisco Proprietary: GLBP - Gateway Load Balancing Protocol

Introduced in Cisco IOS 12.2(14)S for routers, but is not consistently supported across all switching platforms.
- Group members elect one gateway to be the **active virtual gateway (AVG)**.
- Members are **active virtual forwarders (AVF)**, backup for AVG in the event it becomes unavailable.
- The AVG assigns a different virtual MAC address to each AVF (which becomes their identifier).
- Each AVF assumes responsibility for forwarding packets sent to the virtual MAC address assigned
- If an AVF fails, one of others assumes responsibility for the virtual MAC address.
- Precedence is highest priority value, or the highest IP address in the group
- Automatic selection and simultaneous use of multiple available gateways; automatic failover
- No configuring multiple groups or multiple default gateway configurations like HSRP/VRRP
- Maximum of four routers in each forwarding group. Technically the AVG is also an AVF (it is just "special")
- Up to 1024 virtual routers as GLBP groups on each router's physical interface (number range 0-1023)
- Up to four AVFs per group at a time, secondaries can be backups if one fails

- Priority is 1- 255, default 100. A higher number is preferred.
- Preemption is disabled by default for AVG; AVF preempt has default delay of 30 seconds.
- Hello timer default 3 sec; range 1-60 sec. If msec argument is used, the timer will switch to msec, range of 50- 60,000 msec.
- Hold timer default 10 sec; range 19-180 sec. If msec argument is used, the timer will switch to msec, range of 18,020-180,000 msec.
- Hello messages - multicast 224.0.0.102 - UDP 3222.
- Virtual MAC have the form 0007.b4xx.xxyy. The 16-bit value denoted by xx.xx represents six 0 bits followed by a 10-bit GLBP group number. The 8-bit yy value is the virtual forwarder number.

**Three different types of load balancing**

**Host-dependent** uses MAC address of a host to determine which AVF MAC address to use.
- The option if using stateful NAT (needs each host to be returned to the same virtual MAC address each time it sends an ARP request for the virtual IP address)
- Not recommended for where there are fewer than 20 end hosts.

**Weighted** - place a weight on each device when calculating the amount of load sharing.
- If router A has twice the forwarding capacity of router B, weighting value should be configured accordingly
- Use the **glbp x weighting y** where **x** is the GLBP group number, and **y** is the weighting value (1-254)

**Round-robin (default)** each AVF MAC is used sequentially in ARP replies for virtual IP address.
- If no load balancing is used, GLBP will operate like HSRP, where the AVG will only respond to ARP requests with its own AVF MAC address, and all traffic will be directed to the AVG. Use **no glbp load-balancing**.

It is recommended that unless you are extremely familiar with your network design and mechanisms of GLBP that you don't change the timers. Reset timers back to default **no glbp x timers**, where x is group number.

| Router(config)**#interface fastethernet0/0** | (all of these are interface config mode) |
|---|---|
| **ip address 172.16.100.5 255.255.255.0** | Assigns IP address and netmask. |
| **glbp 10 ip 172.16.100.1** | Put group 10 on this int w/ virtual IP of 172.16.100.1. |
| **glbp 10 preempt** | Take over as AVG for group 10 if this router has higher priority than current AVG |
| **glbp 10 preempt delay minimum 60** | Wait before preempt AVG- delay of 60 seconds |
| **glbp 10 forwarder preempt** | Take over as AVF for group 10 if this router has higher priority than current AVF. |
| **glbp 10 forwarder preempt delay minimum 60** | Wait before preempt AVF- delay of 60 seconds |
| **glbp 10 priority 150** | Sets the priority level of the router. |
| **glbp 10 timers 5 15** | Configures the hello timer 5 sec and the hold 15 sec |
| **glbp 10 timers msec 20200 msec 60600** | Hello timer 20,200 msec hold timer to 60,600 millisec |
| **glbp 10 authentication text edmonton** | Configures GLBP for plain text authentication |
| **glbp 10 authentication md5 key-chain vancouver** | Configures GLBP for MD5 authentication |
| **glbp 10 load- balancing host-dependent** | Load balance using the host-dependent method. |
| **glbp 10 load- balancing weighted** | Load balance using the weighted method. |
| **glbp 10 weighting 80** | Set maximum weighting value for this interface |
| **glbp 10 load balancing round robin** | Load balance using the round-robin method. |

**Interface Tracking**

| | |
|---|---|
| Router(config)**#track 2 interface fa0/1 line-protocol** | Config FastEthernet0/1 interface to be tracked. The **line-protocol** tracks whether the interface is up |
| Router(config-track)**#exit** | Notice prompt "(config-track)" Use ? for more options |
| Router(config)**#interface fastethernet0/0** | |
| Router(config-if)**#glbp 10 weighting 110 lower 20 upper 50** | Initial weighting value; upper/lower thresholds, for GW |
| Router(config-if)**#glbp 10 weighting track 2 decrement 50** | Track the object and decrement the weight by 50 when the Fast Ethernet 0/1 interface fails |

**Verifying GLBP**

| | |
|---|---|
| Router**#show glbp** | Displays GLBP information |
| Router**#show glbp brief** | Displays a brief status of all GLBP groups |
| Router**#show glbp 10** | Displays information about GLBP group 10 |
| Router**#show glbp vlan10** | Displays GLBP information on interface Vlan10 |
| Router**#show glbp vlan20 10** | Displays GLBP group 10 info on interface Vlan20 |

**Debugging GLBP**

| | |
|---|---|
| Router**#debug condition glbp** | Displays GLBP condition messages |
| Router**#debug glbp errors** | Displays all GLBP error messages |
| Router**#debug glbp events** | Displays all GLBP event messages |
| Router**#debug glbp packets** | Displays messages about packets sent and received |
| Router**#debug glbp terse** | Displays a limited range of debugging messages |

A redirect timer is used to determine when the AVG will stop using a stale virtual MAC address in ARP replies. The AVF corresponding to the old address continues to act as a stand-in gateway for any clients that try to use it.

When the timeout timer expires, the old MAC address and the virtual forwarder using it are flushed from all the GLBP peers. The AVG assumes that the previously failed AVF will not return to service, so the resources assigned to it must be reclaimed.
At this point, clients still using the old MAC address in their ARP caches must refresh the entry to obtain the new virtual MAC address.

The redirect timer defaults to 600 seconds (10 minutes) and can range from 0 to 3600 seconds (1 hour). The timeout timer defaults to 14,400 seconds (4 hours) and can range from 700 to 64,800 seconds (18 hours). You can adjust these timers with the following interface configuration command:
Switch(config-if)# **glbp** *group* **timers redirect** *redirect timeout*

Switch(config)# **track** *object-number* **interface** *type member/module/number* {**line- protocol** | **ip routing**}
The object-number is an arbitrary index (1 to 500) that is used for weight adjustment. The condition that triggers an adjustment can be **line-protocol** (the interface line protocol is up) or **ip routing**. (IP routing is enabled, the interface has an IP address, and the interface is up.)

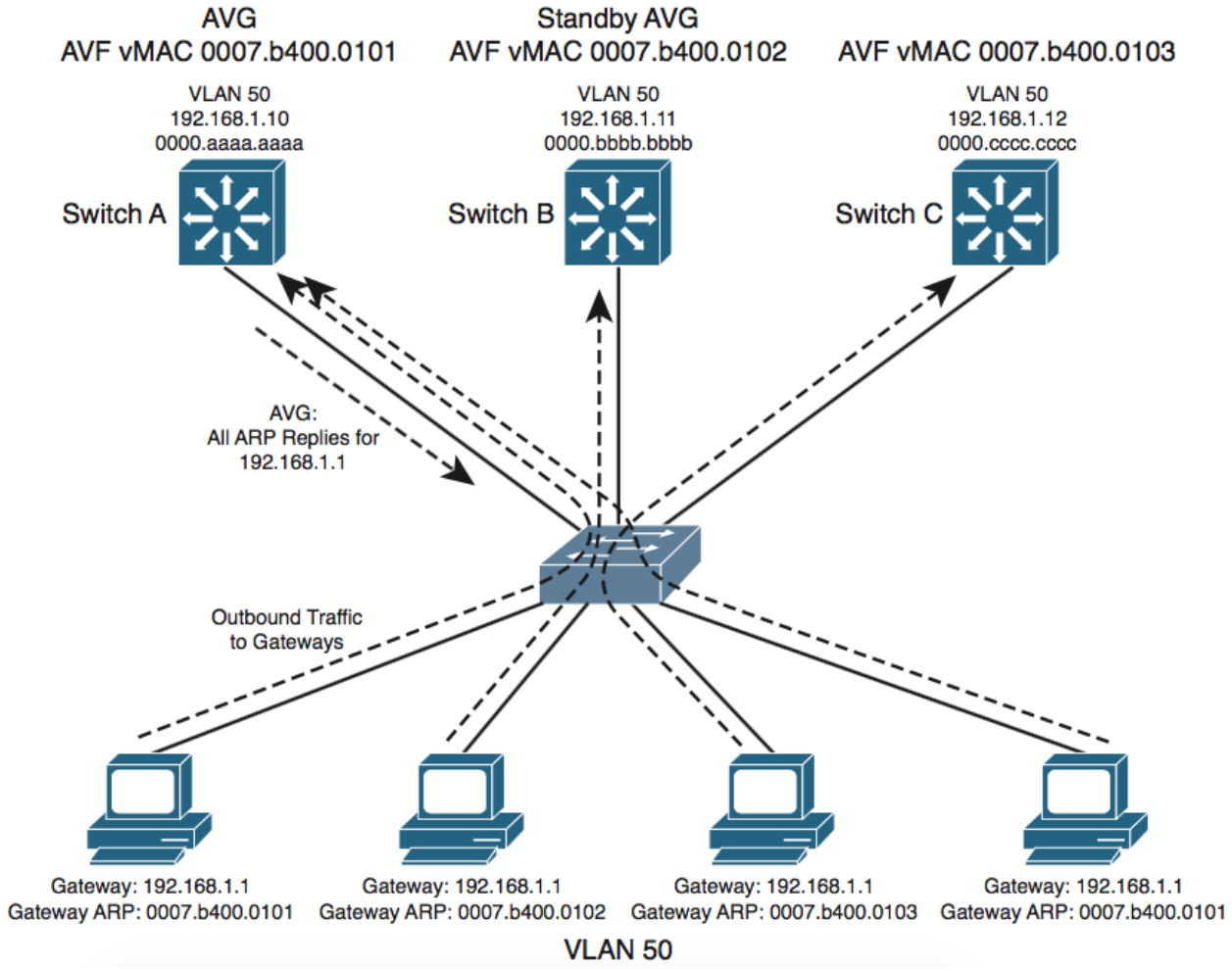
GLBP can also be used with IPv6. Rather than specifying an IPv6 address, use the following command to autoconfigure the address:
Switch(config-if)# **glbp** *group* **ipv6 autoconfigure**

## *Typical 3 Switch GLBP Setup Example*

Three multilayer switches participating in a common GLBP group. with switch A elected AVG, to coordinate the GLBP process, answer all ARP requests for the virtual router 192.168.1.1.

It has identified itself, and other two switches as AVFs for the group, using default round-robin load balancing, so each of the client PCs sends an ARP request to look for the virtual router address (192.168.1.1) in turn, from left to right. Each time the AVG replies, the next sequential virtual MAC is sent back to a client. After the fourth PC sends a request, all three virtual MAC addresses (and AVF routers) have been used, so the AVG cycles back to the first virtual MAC address.

One GLBP group has been configured, clients know of only one gateway IP address: 192.168.1.1, all uplinks are being used, and all routers are proportionately forwarding traffic.




Switch-A(config)# **interface vlan 50**
Switch-A(config-if)# **ip address 192.168.1.10 255.255.255.0**
Switch-A(config-if)# **glbp 1 priority 200**
Switch-A(config-if)# **glbp 1 preempt**
Switch-A(config-if)# **glbp 1 ip 192.168.1.1**

Switch-B(config)# **interface vlan 50**
Switch-B(config-if)# **ip address 192.168.1.11 255.255.255.0**
Switch-B(config-if)# **glbp 1 priority 150**
Switch-B(config-if)# **glbp 1 preempt**
Switch-B(config-if)# **glbp 1 ip 192.168.1.1**

Switch-C(config)# **interface vlan 50**
Switch-C(config-if)# **ip address 192.168.1.12 255.255.255.0**
Switch-C(config-if)# **glbp 1 priority 100**
Switch-C(config-if)# **glbp 1 ip 192.168.1.1**

You can verify GLBP operation with the **show glbp** [**brief**] command. With the **brief** keyword, the GLBP roles are summarized showing the interface, GLBP group number (Grp), virtual forwarder number (Fwd), GLBP priority (Pri), state, and addresses.

```
Switch-A# show glbp brief
Interface   Grp  Fwd Pri State    Address         Active router   Standby router
Vl50        1    -   200 Active   192.168.1.1     local           192.168.1.11
Vl50        1    1   7   Active   0007.b400.0101  local           -
Vl50        1    2   7   Listen   0007.b400.0102  192.168.1.11    -
Vl50        1    3   7   Listen   0007.b400.0103  192.168.1.12    -
Switch-A#

Switch-B# show glbp brief
Interface   Grp  Fwd Pri State    Address         Active router   Standby router
Vl50        1    -   150 Standby  192.168.1.1     192.168.1.10    local
Vl50        1    1   7   Listen   0007.b400.0101  192.168.1.10    -
Vl50        1    2   7   Active   0007.b400.0102  local           -
Vl50        1    3   7   Listen   0007.b400.0103  192.168.1.12    -
Switch-B#

Switch-C# show glbp brief
Interface   Grp  Fwd Pri State    Address         Active router   Standby router
Vl50        1    -   100 Listen   192.168.1.1     192.168.1.10    192.168.1.11
Vl50        1    1   7   Listen   0007.b400.0101  192.168.1.10    -
Vl50        1    2   7   Listen   0007.b400.0102  192.168.1.11    -
Vl50        1    3   7   Active   0007.b400.0103  local          -
```

Notice that Switch A is shown to be the AVG because it has a dash in the Fwd column and is in the Active state. It also is acting as AVF for virtual forwarder number 1.

Because the GLBP group has three routers, there are three virtual forwarders and virtual MAC addresses.

Switch A is in the Listen state for forwarders number 2 and 3, waiting to be given an active role in case one of those AVFs fails.

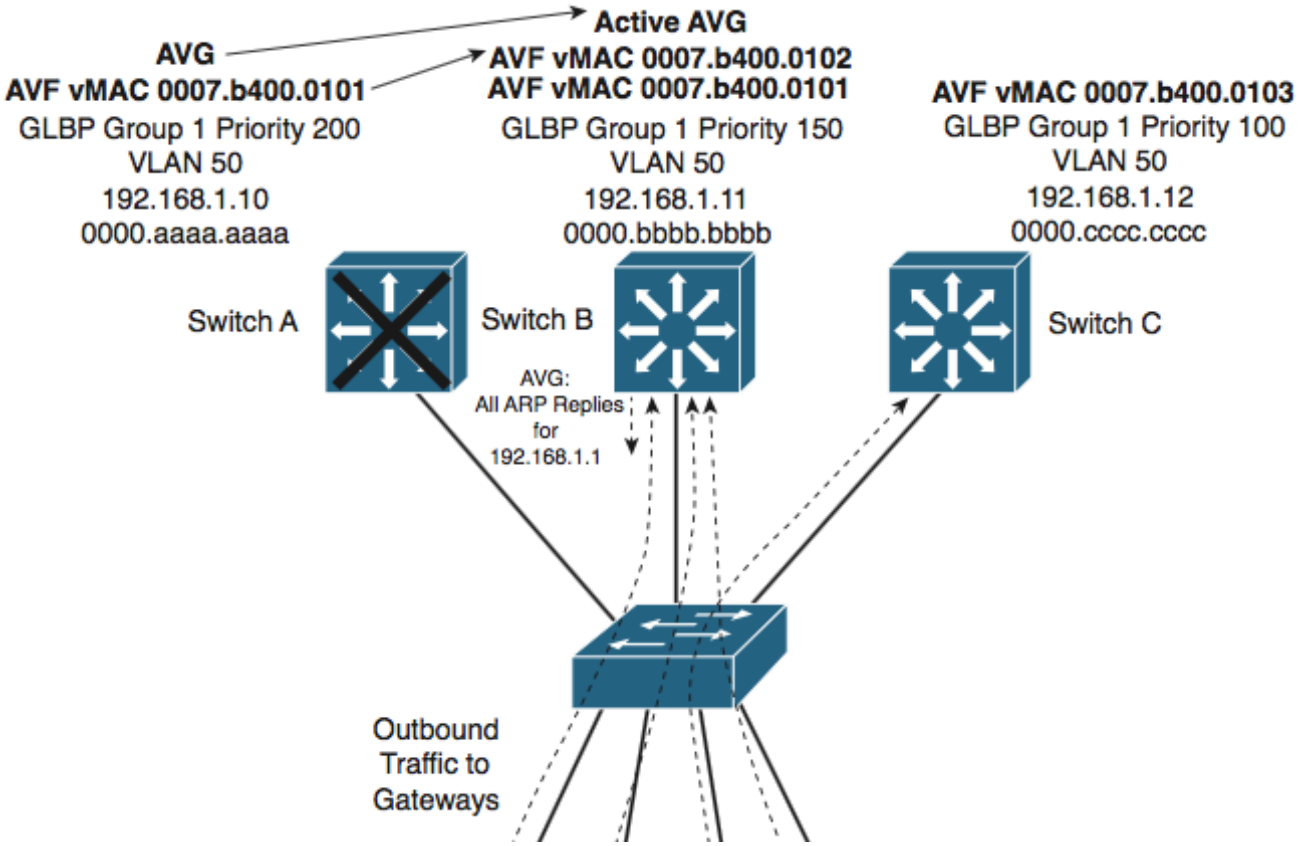
Switch B is shown to have the Standby role, waiting to take over in case the AVG fails. It is the AVF for virtual forwarder number 2.

Finally, Switch C has the lowest GLBP priority, so it stays in the Listen state, waiting for the active or standby AVG to fail. It is also the AVF for virtual forwarder number 3.

Taking off the **brief** keyword and just using **show glbp** displays more detailed info about config and status

The AVG here shows the virtual forwarder roles it has assigned to each of the routers in the GLBP group.

```
Switch-A# show glbp
Vlan50 - Group 1
  State is Active
    7 state changes, last state change 03:28:05
  Virtual IP address is 192.168.1.1
  Hello time 3 sec, hold time 10 sec
    Next hello sent in 1.672 secs
  Redirect time 600 sec, forwarder time-out 14400 sec
  Preemption enabled, min delay 0 sec
  Active is local
  Standby is 192.168.1.11, priority 150 (expires in 9.632 sec)
  Priority 200 (configured)
  Weighting 100 (default 100), thresholds: lower 1, upper 100
  Load balancing: round-robin
  There are 3 forwarders (1 active)
  Forwarder 1
    State is Active
      3 state changes, last state change 03:27:37
    MAC address is 0007.b400.0101 (default)
    Owner ID is 00d0.0229.b80a
    Redirection enabled
    Preemption enabled, min delay 30 sec
    Active is local, weighting 100
  Forwarder 2
    State is Listen
    MAC address is 0007.b400.0102 (learnt)
    Owner ID is 0007.b372.dc4a
    Redirection enabled, 598.308 sec remaining (maximum 600 sec)
    Time to live: 14398.308 sec (maximum 14400 sec)
    Preemption enabled, min delay 30 sec
    Active is 192.168.1.11 (primary), weighting 100 (expires in 8.308 sec)
  Forwarder 3
    State is Listen
    MAC address is 0007.b400.0103 (learnt)
    Owner ID is 00d0.ff8a.2c0a
    Redirection enabled, 599.892 sec remaining (maximum 600 sec)
    Time to live: 14399.892 sec (maximum 14400 sec)
    Preemption enabled, min delay 30 sec
    Active is 192.168.1.12 (primary), weighting 100 (expires in 9.892 sec)
```




Redundancy is also inherent in the GLBP group: Switch A is the AVG, but the next-highest priority router can take over if the AVG fails.

All routers have been given an AVF role for a unique virtual MAC address in the group.

If one AVF fails, some clients remember the last-known virtual MAC address that was handed out. Therefore, another of the routers also takes over the AVF role for the failed router, causing the virtual MAC address to remain alive at all times.

Above is illustrated how these redundancy features react when the current active AVG fails. Before its failure, Switch A was the AVG because of its higher GLBP priority.

After it failed, Switch B became the AVG, answering ARP requests with the appropriate virtual MAC address for gateway 192.168.1.1.

Switch A also had been acting as an AVF, participating in the gateway load balancing.

Switch B also picks up this responsibility, usingits virtual MAC address 0007.b400.0102 along with the one Switch A had been using, 0007.b400.0101.

Therefore, any hosts that know the gateway by any of its virtual MAC addresses still can reach a live gateway or AVF.

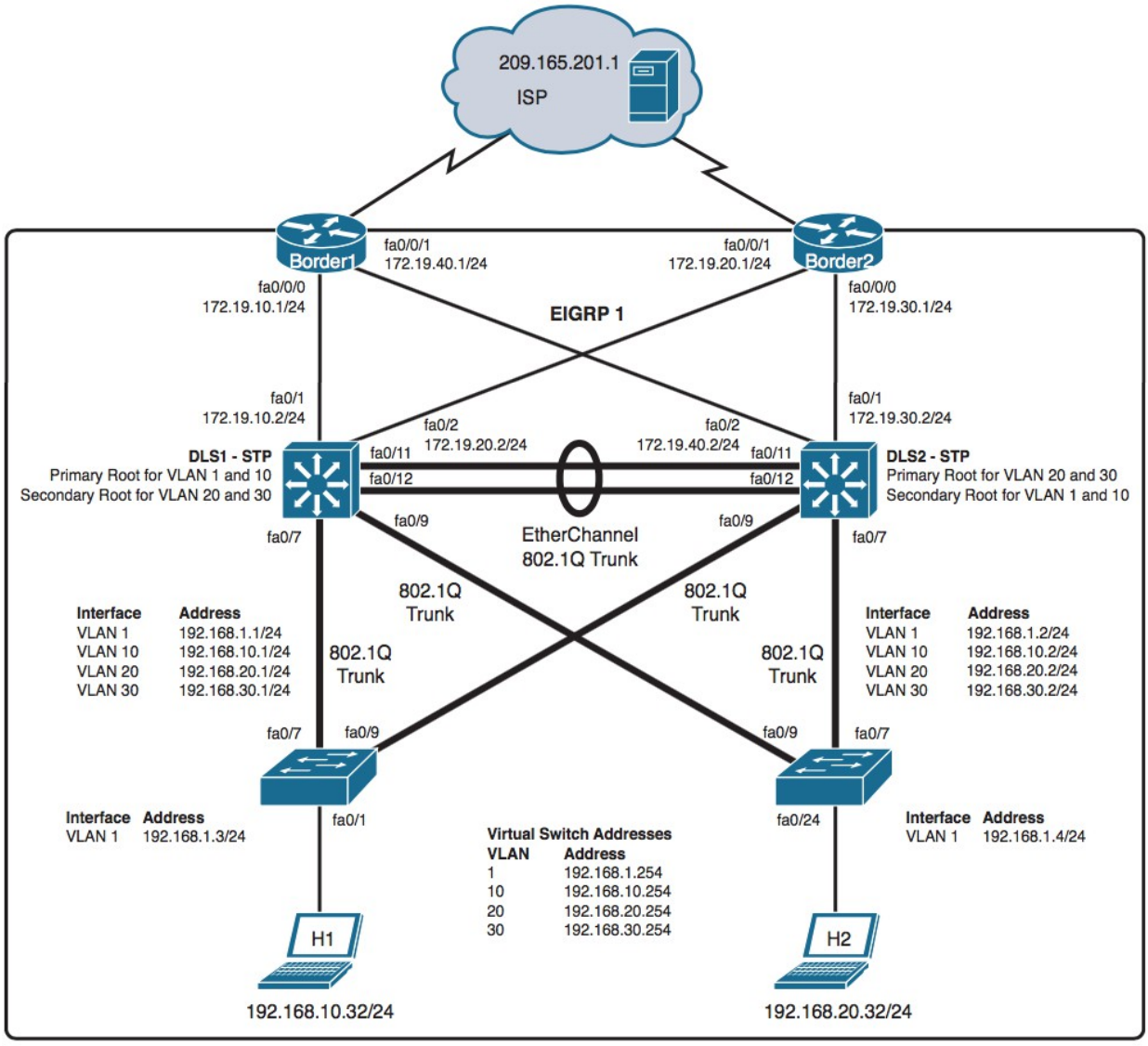



**Figure 13-2** Network Topology for HSRP Configuration Example

# *DHCP Setup on Cisco Devices (IPv4)*

Switch(config)#service dhcp -- makes sure it's enabled - "no service dhcp" turns it off
Switch(config)#ip dhcp pool Sales_Wireless --create the pool
Switch(dhcp-config)#network 192.168.10.0 255.255.255.0 -- netID and mask for pool
Switch(dhcp-config)#default-router 192.168.10.1 --default gateway
Switch(dhcp-config)#dns-server 4.4.4.4
Switch(dhcp-config)#lease 3 12 15 --days, hours, minutes - default 24hours
Switch(dhcp-config)#exit
Switch(config)#ip dhcp excluded-address 192.168.10.1 192.168.10.10

*DHCP Relay*
If you need to provide addresses from a DHCP server to hosts that aren't on the same LAN as the DHCP server, you can configure your router interface to relay DHCP client requests.
Router(config)#interface fa0/0
Router(config-if)#ip helper-address 10.10.10.254

The router and fa0/0 are on 192.168.10.1. 10.10.10.254 is obviously on a different subnet, but is the DHCP server. ip helper-address forwards more than just DHCP client requests.

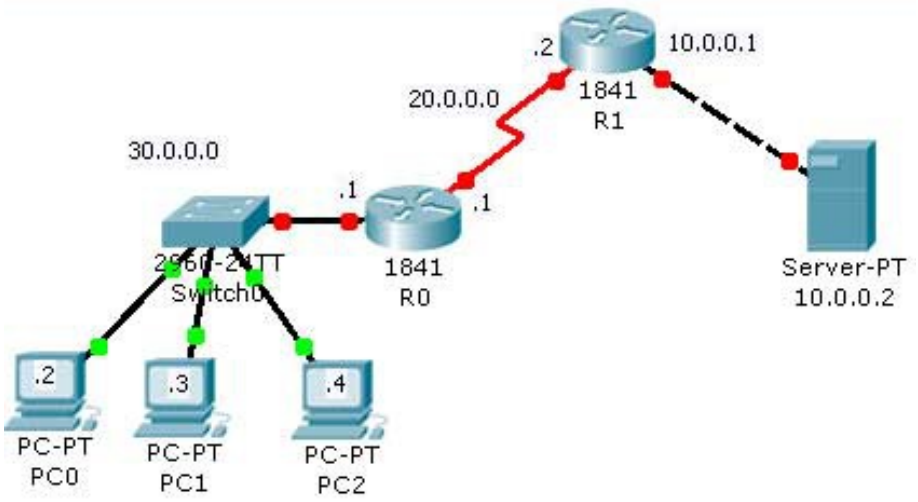
*Verifying DHCP*
show ip dhcp binding - Lists state information about each IP address currently leased
show ip dhcp pool [poolname] - Lists scope of addresses, plus stats for leased addresses
show ip dhcp server statistics - Lists DHCP server statistics
show ip dhcp conflict - show duplicate addresses

*More Pools Example*
Corp#config t
Corp(config)#ip dhcp excluded-address 192.168.10.1
Corp(config)#ip dhcp excluded-address 192.168.20.1
Corp(config)#ip dhcp pool SF_LAN
Corp(dhcp-config)#network 192.168.10.0 255.255.255.0
Corp(dhcp-config)#default-router 192.168.10.1
Corp(dhcp-config)#dns-server 4.4.4.4
Corp(dhcp-config)#exit
Corp(config)#ip dhcp pool LA_LAN
Corp(dhcp-config)#network 192.168.20.0 255.255.255.0
Corp(dhcp-config)#default-router 192.168.20.1
Corp(dhcp-config)#dns-server 4.4.4.4

Add these to the two routers so they will forward DHCP:
LA(config)#int f0/0
LA(config-if)#ip helper-address 172.16.10.5

SF(config)#int f0/0
SF(config-if)#ip helper-address 172.16.10.1

## *Network Address Translation : Static and Dynamic NAT and PAT*




**IPv4 Static NAT**

R1(config-if)#ip address 10.0.0.1 255.0.0.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface serial 0/0/0
R1(config-if)#ip address 20.0.0.2 255.0.0.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#ip route 30.0.0.0 255.0.0.0 20.0.0.1
**R1(config)#ip nat inside source static 10.0.0.2 50.0.0.1**
**R1(config)#interface fastEthernet 0/0**
**R1(config-if)#ip nat inside**
R1(config-if)#exit
**R1(config)#interface serial 0/0/0**
**R1(config-if)#ip nat outside**
R1(config-if)#exit

R0(config)#interface fastethernet 0/0
R0(config-if)#ip address 30.0.0.1 255.0.0.0
R0(config-if)#no shutdown
R0(config-if)#exit
R0(config)#interface serial 0/0/0
R0(config-if)#ip address 20.0.0.1 255.0.0.0
R0(config-if)#clock rate 64000
R0(config-if)#bandwidth 64
R0(config-if)#no shutdown
R0(config-if)#exit
**R0(config)#ip route 50.0.0.0 255.0.0.0 20.0.0.2**

There is not direct route for 10.0.0.2. So PC from network of 30.0.0.0 will never know about it. They will access 50.0.0.1 as the web server IP. Ping 50.0.0.1 and it works. Ping 10.0.0.2 and you will get destination host unreachable error.

**IP4 Dynamic NAT**
This is using 192.168.0.0 network as internal. We have five public ip address 50.0.0.1 to 50.0.0.5 to use. Router1(1841 Router0) is going to be NAT device. Starting off by configuring Router1(1841 Router0):

R1(config)#interface fastethernet 0/0
R1(config-if)#ip address 192.168.0.1 255.0.0.0
R1(config-if)#no shutdown
R1(config)#interface serial 0/0/0
R1(config-if)#ip address 30.0.0.1 255.0.0.0
R1(config-if)#clock rate 64000
R1(config-if)#bandwidth 64
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#ip route 0.0.0.0 0.0.0.0 serial 0/0/0
**R1(config)#access-list 1 permit 192.168.0.0 0.0.0.255**
**R1(config)#ip nat pool test 50.0.0.1 50.0.0.5 netmask 255.0.0.0**
**R1(config)#ip nat inside source list 1 pool test**
**R1(config)#interface fastEthernet 0/0**
**R1(config-if)#ip nat inside**
R1(config-if)#exit
**R1(config)#interface serial 0/0/0**
**R1(config-if)#ip nat outside**
R1(config-if)#exit

R1#debug ip nat
IP NAT debugging is on
NAT: s=192.168.0.7->50.0.0.1, d=20.0.0.2[1]
NAT*: s=20.0.0.2, d=50.0.0.1->192.168.0.7[1]
R1#no debug ip nat
IP NAT debugging is off

As you can see in output 192.168.0.5 is translated with 50.0.0.1 before leaving the router
Webpage loads as well.

**IP4 PAT**
If you have few global IP address and hundred of inside local address to translate. In such a situation you need to use PAT. This time we are using only one global IP address 50.0.0.1

**R1(config)#access-list 1 permit 192.168.0.0 0.0.0.255**
**R1(config)#ip nat pool test 50.0.0.1 50.0.0.1 netmask 255.0.0.0**
**R1(config)#ip nat inside source list 1 pool test overload**
**R1(config)#interface fastEthernet 0/0**
**R1(config-if)#ip nat inside**
R1(config-if)#exit
**R1(config)#interface serial 0/0/0**
**R1(config-if)#ip nat outside**
R1(config-if)#exit

To verify PAT go on R1 and run show ip nat translations
R1#show ip nat translations

| Pro | Inside global | Inside local | Outside local | Outside global |
|---|---|---|---|---|
| icmp | 50.0.0.1:1 | 192.168.0.7:1 | 20.0.0.2:1 | 20.0.0.2:1 |
| icmp | 50.0.0.1:2 | 192.168.0.7:2 | 20.0.0.2:2 | 20.0.0.2:2 |

# Access Control Lists on Cisco Devices

Set ACL in global config, then apply to interfaces.
Put the right ACL on the right port! Outbound or inbound?
Put the ACL on the right device!
- standard ACLs should be close to destination, and extended ACLs close to source
- Prevent unecessary network traffic that will just be denied by the subnet on the other side.
- Use as-needed granularity on the client end prevent a mass denial of traffic.

Corp(config)#access-list {1-99 | 1300-1999} - for standard
Corp(config)#access-list {100-199 | 2000-2699} - for IP extended access list

None of these source addresses should be ever be allowed to enter a production network:
Deny any source addresses from your internal networks.
Deny any local host (127.0.0.0/8) and reserved private addresses (RFC 1918).
Deny any addresses in the IP multicast address range (224.0.0.0/4).

Put most specific (single IPs) first. Subnet or range of subnets lower on list
Put permits first then place deny entries lower on the lists to filter properly
e.g.: Place 10.1.1.1 allow first, and have a later entry for deny 10.1.1.x for the rest of that subnet.

After adding restrictions (deny statements) append an explicit "permit any" at end of rules to counteract implicit deny. See the set ACLs listed in the running-config
You need to disable an ACL before making changes: no ip access-list 3

Corp(config)#access-list 10 [deny | permit | remark] [any | subnet | host] [wildcard (for subnets)]
Wildcard masks: for a block of addresses. Each block size must start at 0 or a multiple of the block size.

Corp(config)#access-list 10 deny 172.16.0.0 0.0.255.255
Match the first two octets and that the last two octets can be any value
Corp(config)#access-list 10 deny 172.16.16.0 0.0.3.255
This configuration tells the router to start at network 172.16.16.0 and use a block size of 4
The range would then be 172.16.16.0 through 172.16.19.255
Corp(config)#access-list 10 deny 172.16.16.0 0.0.7.255
172.16.16.0 going up a block size of 8 to 172.16.23.255
Corp(config)#access-list 10 deny 172.16.32.0 0.0.15.255
172.16.32.0 and goes up a block size of 16 to 172.16.47.255
Corp(config)#access-list 10 deny 172.16.64.0 0.0.63.255
172.16.64.0 and goes up a block size of 64 to 172.16.127.255
Corp(config)#access-list 10 deny 192.168.160.0 0.0.31.255
192.168.160.0 and goes up a block size of 32 to 192.168.191.255
---
Router(config)#access-list 1 deny 172.16.128.0 0.0.31.255 --- /19 is a 32 block means 144 is in 128-160.0
Router(config)#access-list 1 deny 172.16.48.0 0.0.15.255 --- /20 is a 16 block means 50 is in 48-60.0
Router(config)#access-list 1 deny 172.16.192.0 0.0.63.255 ---/18 is a 64 block means 198 is in 192-254.0
Router(config)#access-list 1 deny 172.16.88.0 0.0.7.255 ---/21 is an 8 block means 92 is in 88-106.0
Router(config)#access-list 1 permit any
Router(config)#interface serial 0
Router(config-if)#ip access-group 1 out
*It turns out the rules could have been done this with one line:*
Router(config)#access-list 1 deny 172.16.0.0 0.0.255.255
**The value of 0.0.0.0/255.255.255.255 can be specified as any.**
---
Need: a standard access list that permits only the host or hosts you want to be able to telnet into the routers, then apply the access list to the VTY line with the access-class in command.
Lab_A(config)#access-list 50 permit host 172.16.10.3
Lab_A(config)#line vty 0 4
Lab_A(config-line)#access-class 50 in <--- use "-class" when configuring lines instead of "-group"
---
Users on the Sales LAN (172.16.40.0/24 - Fa0/0) should not have access to the Finance LAN (172.16.50.0/24 Fa0/1), but they should be able to access the Internet and the marketing department files. The Marketing LAN needs to access the Finance LAN for application services.
Lab_A(config)#access-list 10 deny 172.16.40.0 0.0.0.255
Lab_A(config)#access-list 10 permit any
*Place it close to the source*

If you place it as an incoming access list on Fa0/0, you might as well shut down the interface because all of the Sales LAN devices will be denied access to all networks attached to the router. The best place to apply this access list is on the Fa0/1 interface as an outbound list:
Lab_A(config)#int fa0/1
Lab_A(config-if)#ip access-group 10 out

## Extended ACLs

access-list 165 [deny | permit | remark] [protocol] {sourceIP maskWC} {destIP maskWC} [logical op] [ port | tos ]
**For a single host, use "host" before the IP address and no wildcard mask is required**

interface Serial 0/0
Access-group 100 in
access-list 100 remark Begin -- Allow BGP IN and OUT
access-list 100 permit tcp host 1.1.1.1 host 2.2.2.2 eq bgp
access-list 100 permit udp any host 2.2.2.2 gt 33000
access-list 100 remark End
access-list 100 deny ip any any log

interface BRI 0/0
Access-group 100 in
Access-group 100 out

access-list 100 permit ip any any log
debug IP packet detail XXX (access list number)

access-list 101 permit tcp any any eq telnet
debug ip packet detail 101
[ IP packet debugging is on (detailed) for access list 101 ]

As always, once our access list is created, we must apply it to an interface with the same command used for the IP standard list:
Corp(config-if)#ip access-group 110 in
Or this:
Corp(config-if)#ip access-group 110 out

**Extended ACLs #2**
Lab_A#config t
Lab_A(config)#access-list 110 deny tcp any host 172.16.50.5 eq 21
Lab_A(config)#access-list 110 deny tcp any host 172.16.50.5 eq 23
Lab_A(config)#access-list 110 permit ip any any
Lab_A(config)#int fa0/1
Lab_A(config-if)#ip access-group 110 out
---
Router(config)#access-list 110 deny tcp any 172.16.48.0 0.0.15.255 eq 23
Router(config)#access-list 110 deny tcp any 172.16.192.0 0.0.63.255 eq 23
Router(config)#access-list 110 permit ip any any
Router(config)#interface Ethernet 1
Router(config-if)#ip access-group 110 out
Router(config-if)#interface Ethernet 2
Router(config-if)#ip access-group 110 out
---
Allow HTTP access to the Finance server from source Host B only. All other traffic will be permitted. We need to be able to configure this in only three test statements:
Lab_A(config)#access-list 110 permit tcp host 192.168.177.2 host 172.22.89.26 eq 80
Lab_A(config)#access-list 110 deny tcp any host 172.22.89.26 eq 80
Lab_A(config)#access-list 110 permit ip any any
Lab_A(config)#interface fastethernet 0/1
Lab_A(config-if)#ip access-group 110 out
**webserver:**
**access-list 253 permit tcp 127.16.5.0 0.0.0.255 eq www any <--permit from 172.16.5.* of www to anyone**

**Named Lists - these start with "ip access-"**

```
Lab_A(config)#ip access-list standard BlockSales
Lab_A(config-std-nacl)#deny 172.16.40.0 0.0.0.255
Lab_A(config-std-nacl)#permit any
Lab_A(config-std-nacl)#exit
Lab_A(config)#int fa0/1
Lab_A(config-if)#ip access-group BlockSales out
```

**Adding #comments#**

```
R2(config)#access-list 110 remark Permit Bob from Sales Only To Finance
R2(config)#access-list 110 permit ip host 172.16.40.1 172.16.50.0 0.0.0.255
R2(config)#access-list 110 deny ip 172.16.40.0 0.0.0.255 172.16.50.0 0.0.0.255
R2(config)#ip access-list extended No_Telnet
R2(config-ext-nacl)#remark Deny all of Sales from Telnetting to Marketing
R2(config-ext-nacl)#deny tcp 172.16.40.0 0.0.0.255 172.16.60.0 0.0.0.255 eq 23
R2(config-ext-nacl)#permit ip any any
```

**Adding a line to the sequence list:**

```
Lab_A(config)#do show access-list
Extended IP access list 110
    10 deny tcp any host 172.16.30.5 eq ftp
    20 deny tcp any host 172.16.30.5 eq telnet
    30 permit ip any any
    40 permit tcp host 192.168.177.2 host 172.22.89.26 eq www
    50 deny tcp any host 172.22.89.26 eq www
Lab_A (config)#ip access-list extended 110
Lab_A (config-ext-nacl)#21 deny udp any host 172.16.30.5 eq 69
```

show access-list - displays all access lists and parameters. Shows stats on usage. Doesn't show interfaces.
show access-list 110 - Reveals parameters for 110. Doesn't show interfaces.
show ip access-list - Shows only the IP access lists configured on the router.
show ip interface - Displays which interfaces have access lists set on them.
show running-config - Shows the access lists and the specific interfaces that have ACLs applied on them

Remember: You can’t have two lists on the same interface, in the same direction- one will overwrite the other. configuration.

This extended ACL is used to permit traffic on the 10.1.1.x network (inside) and to receive ping responses from the outside while it prevents unsolicited pings from people outside, permitting all other traffic.

```
interface Ethernet0/1
ip address 172.16.1.2 255.255.255.0
ip access-group 101 in
access-list 101 deny icmp any 10.1.1.0 0.0.0.255 echo
access-list 101 permit ip any 10.1.1.0 0.0.0.255
```

Note: Some applications such as network management require pings for a keepalive function. If this is the case, you might want to limit blocking inbound pings or be more granular in permitted/denied IPs.

# *OSPF - Link State Routing - Dijkstra's algorithm*

Central1(config)#int loopback 0
Central1(config-if)#ip address 172.31.1.1 255.255.255.255
Central1(config)#router ospf 132
Central1(config-router)#network 10.10.10.1 0.0.0.0 area 0
Central1(config-router)#network 172.16.10.0 0.0.0.3 area 1 <--this mask is 255 minus mask, per octet (here 252)
Central1(config-router)#network 172.16.10.4 0.0.0.3 area 2
Central1(config-router)#passive-interface f0/0 <--- says no neighbors should be discovered on the interface

**Have a default route for your networks (so they know the way out), propagate to other routers.**
Central1(config)#ip route 0.0.0.0 0.0.0.0 Fa0/0
Central1(config)#router ospf 1
Central1(config-router)#default-information originate

**Loopback and Router-IDs in OSPF**
If you don't configure a loopback interface on a router, the highest active IP address on a router will become that router's RID during bootup if it isn't already specified. A loopback interface will not override the router-id command, and we don't have to reboot the router to make it take effect as the RID.

1. Highest IP on an active interface by default.
2. Highest logical interface IP (loopback) overrides a physical interface.
3. The router-id overrides the interface and loopback interface.

Reload or use "clear ip ospf process" command, to force changes to take effect

**IPv6 OSPF Routing - Routing info is also attached to the interface.**
Central1(config)#int f0/0
Central1(config-if)#ipv6 ospf 1 area 0
*The same IPv4 OSPF RID is still there- change the RID under the OSPF process ID in the global config:*
Central1(config)#ipv6 router ospf 1
Central1(config-rtr)#router-id 1.1.1.1
Central1(config-rtr)#do clear ip ospf process

**Designated Router Elections**
- By default, all OSPF routers are assigned a DR priority of 1.
- Elections are first won based upon a router's priority level, then highest RID (often the highest IP address)
- Backup designated router (BDR) is the runner-up in the DR election, a standby for the DR
  - receives all routing updates but does not distribute them; steps in if the DR goes down

*Suppose we wanted R1 to become the DR and R2 to become the BDR:*
R1(config)# **interface f0/0**
R1(config-if)# **ipv6 ospf priority 100**
R2(config)# interface f0/0
R2(config-if)# **ipv6 ospf priority 90**

*Prevent routers from ever becoming a DR or BDR by configuring a priority of zero*
R2(config)# interface f0/0
R2(config-if)# **ipv6 ospf priority 0**

Choosing the best route - cost=reference_bandwidth /interface_bandwidth

| Interface | Default Bandwidth | Formula (Kbps) | OSPF Cost |
|---|---|---|---|
| Serial | 1544 Kbps | 100,000/1544 | 64 |
| Ethernet | 10,000 Kbps | 100,000/10,000 | 10 |
| Fast Ethernet | 100,000 Kbps | 100,000/100,000 | 1 |

The default ref bandwidth assumes the fastest link in the network runs at 100 Mbps.
Use **auto-cost reference-bandwidth 10000 to** accommodate links up to 10 Gbps in speed.

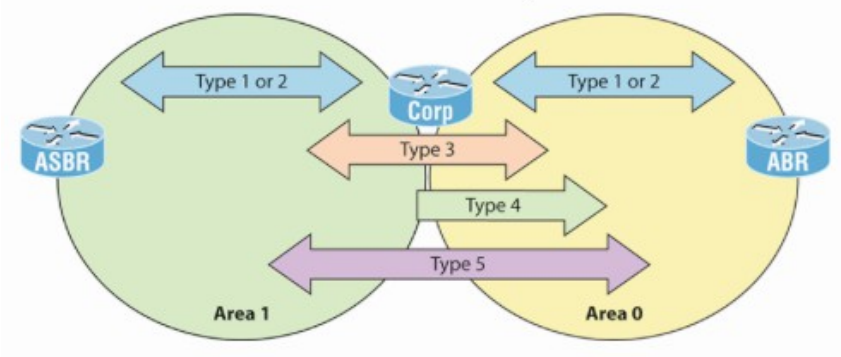
Here are the rules for how a router sets its OSPF interface costs:
- Set the cost explicitly, with **ip ospf cost x interface**, to a value between 1-65,535
- To affect all routes on the device, use **distance 80**. This also affects the AD of non-OSPF routes.
- Change interface bandwidth with the **bandwidth** *speed* command, with speed in Kbps
- Change the reference bandwidth, using subcommand **auto-cost reference-bandwidth** *ref-bw,* in Mbps

Instead of copying routes into the routing table from the LSDB, a router must do the SPF math, choose the best route, and add the route with a subnet number/mask, an outgoing interface, and a next-hop router IP address.

*Left: R1–R7–R8 10 + 180 + 10 = 200*
*Middle: R1–R5–R6–R8 20 + 30 + 40 + 10 = 100*
*Right: R1–R2–R3–R4–R8 30 + 60 + 20 + 5 + 10 = 125*

De facto load balancing occurs when metrics tie for multiple routes to the same subnet, the router can put multiple routes in the routing table (the default is four max- use **maximum-paths** subcommand to change).

**Link-State Advertisements - Basic LSAs**
Type 1 LSA - Router LSA - The common routers describe themselves amongst themselves
Type 2 - Network LSAs - The DR describes itself and it's network to it's area routers
Type 3 LSA - Summary LSAs - Area Border Routers brag to other areas about thier contents
Type 4 LSAs - Area Border Routers give directions to the ASBR
Type 5 LSAs - Autonomous System LSAs - AS external link advertisements




Timers have to match for neighborship
- Flood changed LSAs to neighbors
and re-flood unchanged LSAs when their lifetime expires (default 30 min)
- Maintain neighborship with Hello msgs, listening for Hellos before the dead/hold interval expires.
    **- the dead/hold timer is 4x the hello interval**
    **- 10-sec hello timer and 40-sec hold timer on broadcast and point-to-point links**
    **- 30-sec hello timer and 120-sec hold timer for all other network types**
If a router goes down, the LSDB is populated so all of the routers can recalculate their Shortest Paths

**Becoming neighbors.** When a point-to-point link comes up, the routers on the ends go through interim neighbor states (INIT and 2-Way) exchanging their basic LSAs for discovery, compatibility checks (routerIDs, timers). When both reach 2-Way state, it means they both acknowledge each other as neighbors with matching settings and are prepared to exchange LSDB's. ExStart/Exchange states are when they share DB descriptors, Loading means sending each other Link-State Updates, and Full means their LSDB's match with the same LSA info. These LSA structures comprise the topology. Down means no Hellos are received yet; and Attempt is when NBMA needs manual config.

**LSAs in a Multi-Area Design**
Migrating from a single-area design to a multi-area design has a couple of effects on LSAs:
- Each area has a smaller number of router and network LSAs.
- ABRs have a copy of the LSDB for each area they connect to and submit a router LSA in each area's LSDB.
- Each area has a need for summary (Type 3) LSAs to describe subnets in other areas.

**OSPF: General Verification and Troubleshooting Commands**
**sh int** [**brief**] **x0/0** - up/down, IP address, encapsulation, keepalive, and loopback status
**show ip ospf interface** (**brief**) - address, area, process, router ID, cost, state, priority, timers, DR and BDR
**sh ip ospf** - router ID, area information, SPF statistics, and LSA timer information
**show ip route ospf** - shows the routing table, and displays any injected routes
**show ip ospf neighbor** (**detail**)- neighbor table, related interfaces, hold time, queue, seq number, DR and BDR
**show ip ospf database** - view topology table - output per LSA, organized by type, neighbor RIDs, etc
**show ip ospf topology** (or topology-info)- displays topology table
**show ip ospf traffic** - shows the # and type of traffic received
**show ip protocols** - shows the routing process ID and which protocols are enabled, K values
**show key-chain** - display authentication info
**show ip ospf border-routers**
**show ip ospf data** --lots more on links

- Type 1, or Router LSA, is created and advertised by every router.
- Each internal router has one Router LSA, while each ABR has multiple Router LSAs, one for each area.
- Router LSAs are flooded throughout its (intended) area by sending a copy to all connected neighbors.

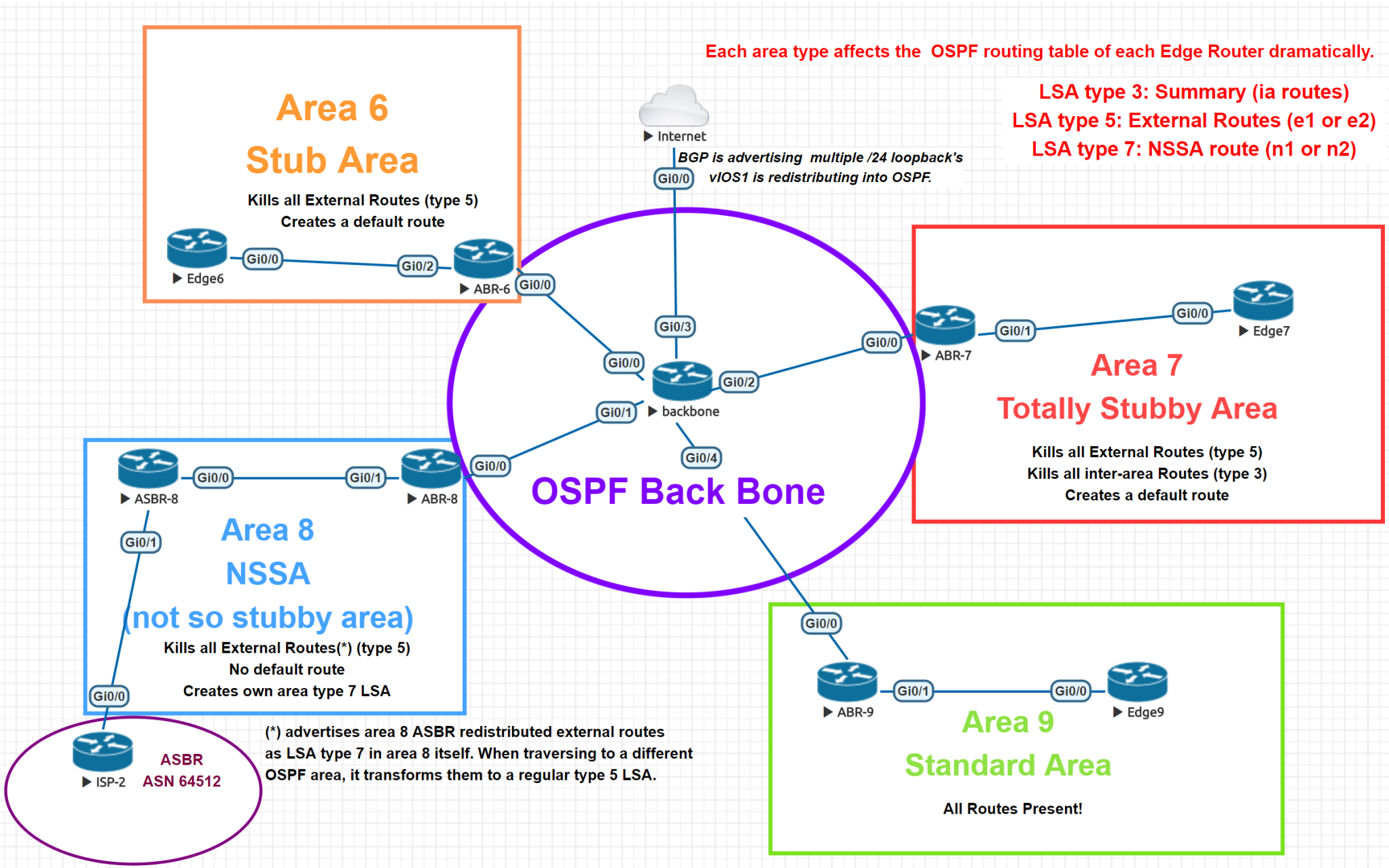
Each one includes: all neighbors directly connected, the router Interface address the LSA was sent

***Basic LSA Layout (This can vary but is a basic look)***

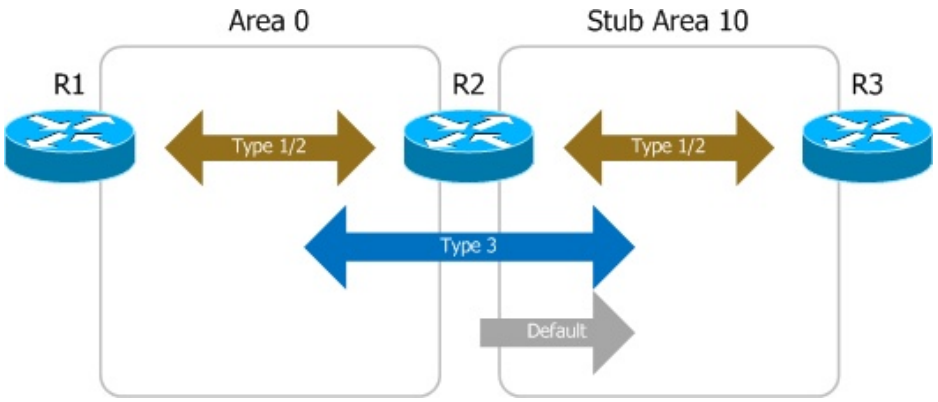
<table>
<tr><td colspan="6">20-byte LSA Header</td></tr>
<tr><td>0 [4 bits]</td><td>V = on if virtual link</td><td>E = on if ASBR</td><td>B = on if ABR</td><td>0 [9 bits]</td><td>Number of Links: count of all router links [16 bits]</td></tr>
<tr><td colspan="6">Link state identifier ID [32 bits]</td></tr>
<tr><td colspan="6">Link Data [32 bits]</td></tr>
<tr><td colspan="2">Type [8 bits]</td><td colspan="2">Number TOS = 0</td><td colspan="2">Metric</td></tr>
<tr><td colspan="6">Starting from Link ID again...</td></tr>
</table>

***An overview of Link State Advertisement Types***

| LSA Type | Description | Routing Table Code |
|---|---|---|
| 1 | Router LSA. Advertises intra-area routes. Generated by each OSPF router to ID itself to peers. Flooded only within the area. Represents stub networks as well | O |
| 2 | Network LSA. Generated by a DR, advertises routers on a multiaccess link.. Flooded only within the area. Represents the subnet and the router interfaces connected to the subnet. One per transit network. | O |
| 3 | Network Summary LSA. Advertises interarea routes. Generated by an ABR, flooded to adjacent areas. Defines the links (subnets) in the origin area, and cost, but no topology data. "I am the ABR and these are my area's subnets" | O IA |
| 4 | ASBR Summary LSA. Advertises the route to reach an ASBR. Generated and advertised by ABR. Flooded to adjacent areas. | O IA |
| 5 | AS External LSA. Advertises external routes (in another routing domain) Generated by an ASBR and flooded to adjacent areas.<br>E1–metric increases at each router as it is passed through the network.<br>E2–metric does not increase (this is the default). | O |
| 6 | Multicast LSA. Used in multicast OSPF (MOSPF) operations, not supported by Cisco IOS. | |
| 7 | Not-so-stubby area (NSSA) LSA. Advertises routes in another routing domain. Generated by an ASBR, advertised within a NSSA (instead of a type-5 LSA)<br>**Totally NSSA bends this rule- that's why it accepts Type 4 LSA.**<br>N1–metric increases as it is passed through the network.<br>N2–metric does not increase (default). | O |
| 8 | External Attributes LSA, aka Link LSA. For OSPF and BGP interaction. Not implemented in Cisco. **Another definition at Cisco says this is an IPv6 tool:**<br>"IPv6 local-link flooding scope (never flooded beyond), provide the link-local address to all other routers attached to the link, also gives a list of IPv6 prefixes to associate with the link. Allows the router to assert a collection of Options bits to associate with the network LSA that will be originated for the link" | |
| 9 | Intra-Area-Prefix LSAs - A router can originate multiple intra-area-prefix LSAs for each router or transit network, each with a unique link-state ID. The link-state ID for each intra-area-prefix LSA describes its association to either the router LSA or the network LSA and contains prefixes for stub and transit networks. | |
| 8,10,11 | Opaque LSAs. Used for specific applications, these are generic LSAs to allow for easy future extension of OSPF; for example, type 10 has been adapted for MPLS traffic, BGP, IPv6 stuff. Others more vendor-specific- are becoming more defined. | |


Each area type affects the OSPF routing table of each Edge Router dramatically.
LSA type 3: Summary (ia routes)
LSA type 5: External Routes (e1 or e2)
LSA type 7: NSSA route (n1 or n2)
Area 6
Stub Area
Kills all External Routes (type 5)
Creates a default route
Edge6
Gi0/0
Gi0/2
ABR-6
Gi0/0
Internet
BGP is advertising multiple /24 loopback's
vIOS1 is redistributing into OSPF.
Gi0/0
Gi0/3
Gi0/0
Gi0/2
Gi0/1
backbone
Gi0/4
OSPF Back Bone
Gi0/0
ABR-7
Gi0/1
Gi0/0
Edge7
Area 7
Totally Stubby Area
Kills all External Routes (type 5)
Kills all inter-area Routes (type 3)
Creates a default route
ASBR-8
Gi0/0
Gi0/1
ABR-8
Gi0/1
Area 8
NSSA
(not so stubby area)
Kills all External Routes(*) (type 5)
No default route
Creates own area type 7 LSA
Gi0/0
ISP-2
ASBR
ASN 64512
(*) advertises area 8 ASBR redistributed external routes
as LSA type 7 in area 8 itself. When traversing to a different
OSPF area, it transforms them to a regular type 5 LSA.
Gi0/0
ABR-9
Gi0/1
Gi0/0
Edge9
Area 9
Standard Area
All Routes Present!

# Stub Areas



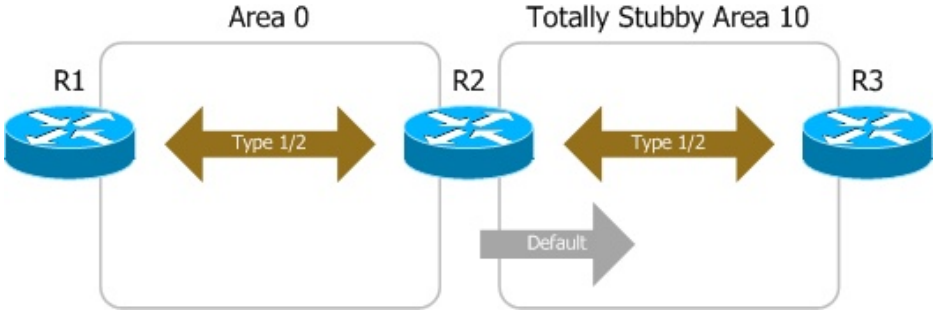

In this next example, R2 and R3 share a common stub area. Instead of propagating external routes (type 5 LSAs) into the area, the ABR injects a type 3 LSA containing a default route into the stub area. This ensures that routers in the stub area will be able to route traffic to external destinations without having to maintain all of the individual external routes. Because external routes are not received by the stub area, ABRs also do not forward type 4 LSAs from other areas into the stub.

For an area to become a stub, all routers belonging to it must be configured to operate as such. Stub routers and non-stub routers will not form adjacencies.

```
Router(config-router)# area 10 stub
```

This idea of substituting a single default route for many specific routes can be applied to internal routes as well, which is the case of *totally stubby areas*.

# Totally Stubby Areas



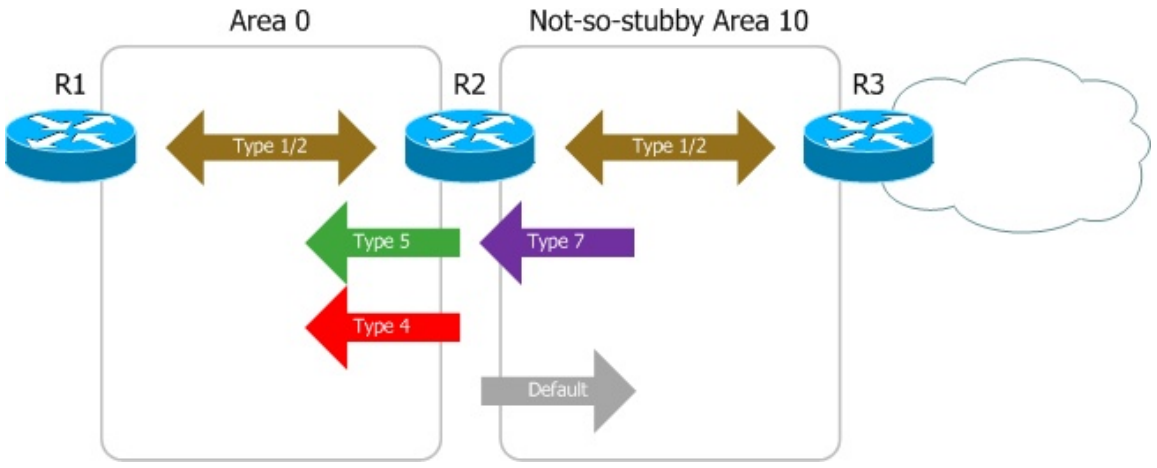

Like stub areas, totally stubby areas do not receive type 4 or 5 LSAs from their ABRs. However, they also do not receive type 3 LSAs; all routing out of the area relies on the single default route injected by the ABR.

A stub area is extended to a totally stubby area by configuring all of its ABRs with the `no-summary` parameter:

```
Router(config-router)# area 10 stub no-summary
```

Stub and totally stubby areas can certainly be convenient to reduce the resource utilization of routers in portions of the network not requiring full routing knowledge. However, neither type can contain an ASBR, as type 4 and 5 LSAs are not permitted inside the area. To solve this problem, and in what is arguably the worst naming decision ever made, Cisco introduced the concept of a *not-so-stubby area (NSSA)*.

## Not-so-stubby Areas




An NSSA makes use of type 7 LSAs, which are essentially type 5 LSAs in disguise. This allows an ASBR to advertise external links to an ABR, which converts the type 7 LSAs into type 5 before flooding them to the rest of the OSPF domain.

An NSSA can function as either a stub or totally stubby area. To designate a normal (stub) NSSA, all routers in the area must be so configured:

```
Router(config-router)# area 10 nssa
```

Type 3 LSAs will pass into and out of the area. Unlike a normal stub area, the ABR will *not* inject a default route into an NSSA unless explicitly configured to do so. As traffic cannot be routed to external destinations without a default route, you'll probably want to include one by appending `default-information-originate` (thanks to Adam for pointing this out).

```
Router(config-router)# area 10 nssa default-information-originate
```

To expand an NSSA to function as a totally stubby area, eliminating type 3 LSAs, all of its ABRs must be configured with the `no-summary` parameter:

```
Router(config-router)# area 10 nssa no-summary
```

The ABR of a totally stubby NSSA (or not-so-totally-stubby area, if you prefer) injects a default route without any further configuration.

## Summary

- **Standard areas** can contain LSAs of type 1, 2, 3, 4, and 5, and may contain an ASBR. The backbone is considered a standard area.
- **Stub areas** can contain type 1, 2, and 3 LSAs. A default route is substituted for external routes.
- **Totally stubby areas** can only contain type 1 and 2 LSAs, and a single type 3 LSA. The type 3 LSA describes a default route, substituted for all external and inter-area routes.
- **Not-so-stubby areas** implement stub or totally stubby functionality yet contain an ASBR. Type 7 LSAs generated by the ASBR are converted to type 5 by ABRs to be flooded to the rest of the OSPF domain.

# ***EIGRP - Enhanced Interior Gateway Routing Protocol***

- A "balanced hybrid" routing protocol, instead of link-state/ distance vector. Sometimes "adv distance vector"
- Sends routing table once, and uses partial updates for changes
- Updates to multicast 224.0.0.10; and the unicast IP of the neighbor if specific.
- Reliable Transport Protocol (RTP) to send updates - resend any EIGRP messages that are not received
- For route poisoning (communicating that a route has "failed" to either warn or stop traffic over the route); EIGRP uses $2^{32}$-1 as "infinity" (just over 4 billion), some new IOS versions raise that value to $2^{56}$-1 (over 10,000 trillion)

**EIGRP default timers** are 5 and 15 seconds for Hello and Hold Intervals (Hold is default 3x Hello interval) Not required, but they should be the same for stability. (if Hold time expires it's a sign the other router is down)

**For Neighborship:** same configured autonomous system number (ASN), the source IP in the neighbor's Hello must be from the same subnet as the interface, matching K values, and pass authentication process if needed.

**3-steps when a router first joins a network, 3 tables to populate**

- Neighbor discovery: Hello messages discover potential neighboring EIGRP routers to add to neighbor table.
- Topology exchange: Exchange full topology update, only partial updates later as needed on changes
- Routing table: Analyzes topology tables, choose lowest-metric route to each subnet.

**Reported Distance (RD) and Feasible Distance (FD), Routing and Topology Tables**

- Reported/advertised distance (RD) is how far a neighbor says it is from a remote network.
- Feasible distance (FD) is calculated by taking that and adding how far away that neighbor is.
- The route with the lowest FD is the route that you'll find in the routing table - it's considered the best path.

D 10.0.0.0/8 [90/2195456] via 172.16.10.2, 00:27:06, Serial0/0
D (for DUAL) says this an EIGRP injected route. Get to the 10.0.0.0 network via its neighbor 172.16.10.2.
The first number (90) is the administrative distance (AD) of the routing protocol, the second number is the feasible distance (FD)- the entire cost for this router to get to network 10.0.0.0/8 through that neighbor, 2195456

- The neighbor router has a reported (advertised) distance (RD) between it and network 10.0.0.0
- Feasible Distance (FD), or total cost to that network adds RD to the calculated distance to that router.

***Successor Routes and Feasible Successors***

Each router keeps information about adjacent neighbors in the neighbor table and the topology tables hold all destinations advertised by the neighbors, with associated distances and other metrics. That best path is called a successor route, and it is determined by which path has the lowest Feasible Distance.

Also elected is a backup "feasible successor" route, which is kept in the topology table if they are needed. If a nonsuccessor route's RD is less than the FD of the successor, the route is a feasible successor route.
Router E has 3 routers/ paths to choose to get to Router A/ Subnet 1:

| **Router E Topology Table:** | | **Router E Routing Table** |
|---|---|---|
| **Next Hop FD RD** | | Subnet 1 **Metric 14,000**, Through D |
| Router B - 19,000 | 15,000 | |
| Router C - 17,500 | **13,000 <-- RD makes it FS** | |
| Router D - **14,000** | 10,000 **<-- FD makes it SR** | |

Rule is, the lowest FD will determine the Successor Route, and Router D wins with a FD of 14000. You might think that the next smaller FD would be the runner up, right? Absolutely not. In the case of a Feasible Successor, we look at the Successor Route 's FD and find a Reported Distance that is smaller than *that*. Router C wins out as Feasible Successor by the rule because it's RD 13,000 < FD of the current Successor at 14,000.

**Verifying EIGRP - The Configuration and the Neighbor, Topology and Routing Tables**

show ip eigrp interfaces
show ip eigrp interfaces detail
show ip eigrp interfaces *type number*
show ip protocols
show ip eigrp neighbors
show ip eigrp neighbors *type number*
show ip eigrp topology
show ip eigrp topology *subnet/prefix*
show ip route
show ip route eigrp
show ip route *subnet mask*
show ip route | section *subnet*

**EIGRP Metric Calculations**

$$\text{Metric} = \left(\left(\frac{10^7}{\text{least-bandwidth}}\right) + \text{cumulative-delay}\right) * 256$$

In this formula, the term least-bandwidth represents the lowest-bandwidth link in the route, using a unit of Kbps. For instance, if the slowest link in a route is a 10Mbps Ethernet link, the first part of the formula is $10^7$ / $10^4$, which equals 1000. You use $10^4$ in the formula because 10Mbps is equal to 10,000 Kbps ($10^4$ Kbps).
Cumulative-delay is the sum of delay values for all outgoing interfaces in the route, *in tens of microseconds* (µsec) (most **show** commands, including **show ip eigrp topology** and **show interfaces**, list delay in µseconds)
You can set both bandwidth and delay for each link with **bandwidth** and **delay** interface commands.

**A Simple Example**
R1 learns about subnet 10.1.3.0/24 from R2's update listing min. BW of 100,000 Kbps, and delay of 100 µsec.
R1's S0/1 has an interface bandwidth set to 1544 Kbps and a delay of 20,000 µsec
- 1.544 Mbps < 100,000 Kbps, or 100 Mbps. R1 needs to use this slower bandwidth in the metric calculation, (data's being squeezed down a smaller pipe)
- For interface delay, the router always adds its interface delay to the delay listed in the EIGRP Update.
- The update lists 100 µseconds (10 tens of µseconds.

Add R1's S0/1 is 2000 tens of = 2010 tens of µsec.
Metric = [( $10^7$ / 1544) + (10 + 2000)] * 256 = 2,172,416
- Later this is revisited, for Reported Distance that R1 was given. It turns out we have the info already:

Metric = [( $10^7$ / 100,000) + (10)] * 256 = 28,160 - So FD for R1 calc'd at 2,172,416; and the RD from R2: 28160
**- On R1, it listed in the sh ip route like this with the AD/FD:**
D 10.1.3.0/24 [90/2172416] via ???.??.??.?, 23:58:00, S0/1 [no IP address for R2's Serial port (ooops)]
sh ip eigrp topology might have something like this, which shows the FD/RD
P 10.1.3.0/24, 1 successors, FD is 2172416 via ?.?.?.? (2172416/28160), Serial0/1

**The Entire Equation**
*Metric = {K1 × Bandwidth + [(K2 × Bandwidth) / (256 – Load)] + K3 × Delay} × [K5 / (Reliability + K4)]*
K1 Bandwidth (Be) = lowest bandwidth of the links along the path - [10000000 / Kbps] × 256]
K2 Load (utilization on path)
K3 Delay (Dc) = sum of all the delays of the links along the path - [in 10s of µsec] × 256
K4 Reliability (r)
K5 MTU

- K values are only multiples of the metric calculation. Default values are: K1=1, K2=0, K3=1, K4=0, K5=0.
- K1 and K3 are static. K2, K4, and K5 are variable, and we just get overhead calculating them and reporting them if they are on. MTU is exchanged between EIGRP neighbors but not used. By default only K1 and K3 are enabled and we don't use K2 or K4. (only bandwidth and delay are used in the formula)

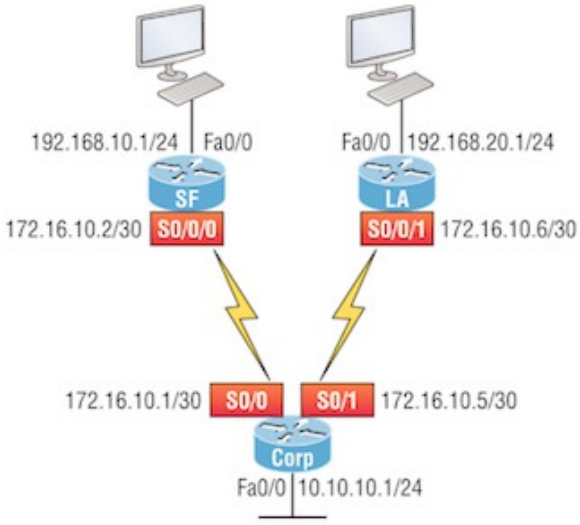
**Serial Links and a Special Problem**
Serial links default to a bandwidth of 1544 and a delay of 20,000 µseconds. A slow 64 Kbps serial link's **bandwidth** command is used to reflect the correct speed (instead of the default 1544

**<u>EIGRP Configuration Commands</u>**
**router eigrp** as-number
**network** ip-address [wildcard-mask] (address should match an int, and often wildcards are unavoidable*)
**eigrp router-id** value
**ip hello-interval eigrp** asn time
[ for interfaces: **ip hold-time eigrp** asn time ]
[ for interfaces: **bandwidth** value
[ for interfaces: **delay** value
**maximum-paths** number
**variance** multiplier to tweak k-values
**auto-summary**

**Failed Routes, DUAL Query and Reply to Update Routing Table**
When a route fails, and there's no feasible successor, EIGRP uses Diffusing Update Algorithm (DUAL) to choose a replacement. DUAL sends queries looking for a loop-free route, and when it's found, DUAL adds it to the routing table. Messages just confirm that a route exists, and would not create a loop. In most networks, convergence can still occur in less than 10 seconds.

# ***Basic IGP Routing in IPv4***

*The three types of routing are static (in which routes are manually configured at the CLI), dynamic (in which the routers share routing information via a routing protocol), and default routing (in which a special route is configured for all traffic without a more specific destination network found in the table)*




*Basic layout to setup*
*Central1 - in diagram is CORP*
Serial 0/0: 172.16.10.1/30
Serial 0/1: 172.16.10.5/30
Fa0/0: 10.10.10.1/24

*DFW - in diagram is SF*
S0/0/0: 172.16.10.2/30
Fa0/0: 192.168.10.1/24

*HOU - in diagram is LA*
S0/0/0: 172.16.10.6/30
Fa0/0: 192.168.20.1/24

***Basic Local IPv4 Routing (Direct Connections)***

*Every interface below should have "no shutdown" added. It has been removed here for brevity*

```
Router(config)#hostname Central1
Central1(config)#int f0/0
Central1(config-if)#desc Connection to LAN
Central1(config-if)#ip address 10.10.10.1 255.255.255.0
Central1(config-if)#int s0/0
Central1(config-if)#desc WAN connection to DFW
Central1(config-if)#ip address 172.16.10.1 255.255.255.252
Central1(config-if)#int s0/1
Central1(config-if)#desc WAN connection to HOU
Central1(config-if)#ip address 172.16.10.5 255.255.255.252
```

**Central1>sh controllers s0/0**

Interface Serial0/0
Hardware is PowerQUICC MPC860
DTE V.35 TX and RX clocks detected.

***Remember to set clock rates on the DCEs!***

```
Router(config)#hostname DFW
DFW(config)#int s0/0/0
DFW(config-if)#desc WAN Connection to Central1
DFW(config-if)#ip address 172.16.10.2 255.255.255.252
DFW(config-if)#clock rate 1000000   - DFW's DCE to Central's DTE -we need a clock rate!
DFW(config-if)#int f0/0
DFW(config-if)#desc DFW LAN
DFW(config-if)#ip address 192.168.10.1 255.255.255.0

Router(config)#hostname HOU
HOU(config)#int s0/0/1
HOU(config-if)#ip address 172.16.10.6 255.255.255.252
HOU(config-if)#clock rate 1000000        - HOU's DCE to Central1's DTE -we need a clock rate
HOU(config-if)#description WAN To Central1
HOU(config-if)#int f0/0
HOU(config-if)#ip address 192.168.20.1 255.255.255.0
HOU(config-if)#description HOU LAN
```

***Static Routing: Manually Adding to Routing Table***

```
Router(config)#ip route 172.16.3.0 255.255.255.0 192.168.2.4
```

172.16.3.0 is the remote network, and 255.255.255.0 is it's mask of the remote network.
192.168.2.4 is the next hop that packets will be sent to. Can also be the interface out to it.

*Below, the 150 is where the administrative distance goes if you want to override the default. This is set here because in the next example, we will add routes with RIP, and using 150 (rather than the default of 1) means we won't have to remove static routes first- RIP (120) will just override them.*

*Administrative Distances (the smaller # wins out):*

| | | | |
|---|---|---|---|
| Connected | 0 | OSPF | 110 |
| Static route | 1 | RIP | 120 |
| EIGRP summary | 5 | iBGP | 200 |
| eBGP | 20 | Unknown | 255 |
| EIGRP | 90 | | |

Central1(config)#ip route 192.168.10.0 255.255.255.0 172.16.10.2 150 --use the dest IP out
Central1(config)#ip route 192.168.20.0 255.255.255.0 s0/1 150 -- use an interface out

DFW(config)#ip route 10.10.10.0 255.255.255.0 172.16.10.1 150
DFW(config)#ip route 172.16.10.4 255.255.255.252 172.16.10.1 150
DFW(config)#ip route 192.168.20.0 255.255.255.0 172.16.10.1 150

HOU#config t
HOU(config)#ip route 10.10.10.0 255.255.255.0 172.16.10.5 150
HOU(config)#ip route 172.16.10.0 255.255.255.252 172.16.10.5 150
HOU(config)#ip route 192.168.10.0 255.255.255.0 172.16.10.5 150

*A stub indicates that the networks in this design have only one way out to reach all other networks; use only a default route. Here's an alt config instead of typing in the static routes. to make this a "stubby network" (first we turn off the routes we just configured):*
HOU(config)#no ip route 10.10.10.0 255.255.255.0 172.16.10.5 150
HOU(config)#no ip route 172.16.10.0 255.255.255.252 172.16.10.5 150
HOU(config)#no ip route 192.168.10.0 255.255.255.0 172.16.10.5 150
HOU(config)#ip route 0.0.0.0 0.0.0.0 172.16.10.5

***Convert from Static IPv4 to Dynamic IPv4 RIPv2***
*As a shortcut, you can verify directly connected networks to know what to configure RIP with:*
Central1#sh ip int brief

| Interface | IP-Address | OK? | Method | Status | Protocol |
|---|---|---|---|---|---|
| FastEthernet0/0 | 10.10.10.1 | YES | manual | up | up |
| Serial0/0 | 172.16.10.1 | YES | manual | up | up |
| FastEthernet0/1 | unassigned | YES | unset | admin. down | down |
| Serial0/1 | 172.16.10.5 | YES | manual | up | up |

> *When we declare our routes in RIP, we need to refer to them in a classful way instead of by the specific subnet. This is illustrated below. RIP then finds the subnets and fills in the routing table (RIP is NOT a classful routing protocol btw).*

Central1(config)#router rip
Central1(config-router)#network 10.0.0.0
Central1(config-router)#network 172.16.0.0
Central1(config-router)#version 2
Central1(config-router)#no auto-summary

*Add the default static route, and enter RIP config to set default-info originate to propagate it.*
Central1(config)#ip route 0.0.0.0 0.0.0.0 Fa0/0
Central1(config)#router rip
Central1(config-router)#default-information originate

DFW(config)#router rip
DFW(config-router)#network 192.168.10.0
DFW(config-router)#network 172.16.0.0
DFW(config-router)#version 2
DFW(config-router)#no auto-summary *---disabling auto-summary makes RIP look at our subnets*
DFW(config-router)#do show ip route

```
C    192.168.10.0/24 is directly connected, FastEthernet0/0
L    192.168.10.1/32 is directly connected, FastEthernet0/0
     172.16.0.0/30 is subnetted, 3 subnets
R       172.16.10.4 [120/1] via 172.16.10.1, 00:00:08, Serial0/0/0
C       172.16.10.0 is directly connected, Serial0/0/0
L       172.16.10.2/32 is directly connected, Serial0/0
S    192.168.20.0/24 [150/0] via 172.16.10.1
     10.0.0.0/24 is subnetted, 1 subnets
R       10.10.10.0 [120/1] via 172.16.10.1, 00:00:08, Serial0/0/0
```

*- Note above the administrative distances set for RIP (R) and our previous static routes (S)*

HOU(config)#no ip route 0.0.0.0 0.0.0.0 *--- get rid of that temporary default route we set for HOU*
HOU(config)#router rip
HOU(config-router)#network 192.168.20.0
HOU(config-router)#network 172.16.0.0
HOU(config-router)#no auto
HOU(config-router)#vers 2

***Switch from IPv4 RIP to Dynamic IPv4 OSPF***
Central1(config)#no router rip
Central1(config)#router ospf 132
Central1(config-router)#network 10.10.10.1 0.0.0.255 area 0
Central1(config-router)#network 172.16.10.1 0.0.0.3 area 0
Central1(config-router)#network 172.16.10.5 0.0.0.3 area 0

DFW(config)#no router rip
DFW(config)#router ospf 300
DFW(config-router)#network 192.168.10.1 0.0.0.255 area 0
DFW(config-router)#network 172.16.10.0 0.0.0.7 area 0
[may need "area 1 range" for this, since it's a summary of two listed on Central1- see below]

HOU(config)#router ospf 100
HOU(config-router)#network 192.168.20.0 0.0.0.255 area 0
HOU(config-router)#network 172.16.10.0 0.0.0.7 area 0

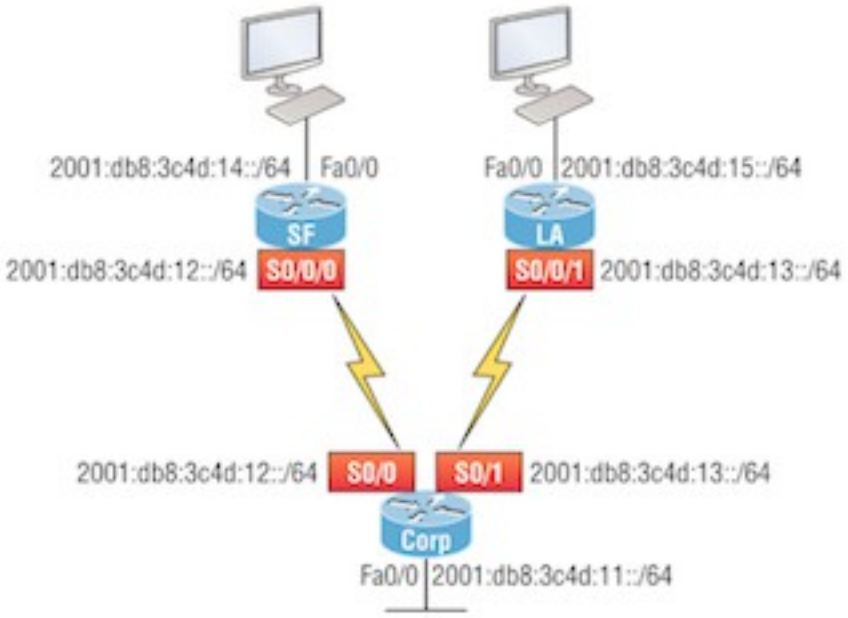
*Adding a non-OSPF network? Use passive-interface:*
HOU(config)#router ospf 100
HOU(config-router)#passive-interface fastEthernet 0/1

*Add a SanAnt router:*
Router(config)#hostname SanAnt
SanAnt(config)#int f0/0
SanAnt(config-if)#ip address 10.10.10.2 255.255.255.0
SanAnt(config-if)#no shut
SanAnt(config-if)#router ospf 2
SanAnt(config-router)#network 10.10.10.0 0.0.255.255 area 0

**OSPF Route summarization:**
Router(config)#router ospf 100
Interarea: area 1 range 192.168.5.8 0.0.0.63
External: summary-address: 192.168.64.0 0.0.224.0

**RouterID - So what if you didn't put in a loopback first and need to force a router to become DR?**
Corp(config-router)#router-id 223.255.255.254
Corp(config-router)#do clear ip ospf process
[Just set a routerID which will beat any other RID or loopback set on other routers, then reset OSPF]

# ***Basic IGP Routing in IPv6***




***Adding IPv6 (including local default routing)***

*Add IPv6 to the Central1, DFW, and HOU routers by using a simple subnet scheme of 11, 12, 13, 14, and 15*

```
Central1#config t
Central1(config)#ipv6 unicast-routing
Central1(config)#int f0/0
Central1(config-if)#ipv6 address 2001:db8:3c4d:11::/64 eui-64
Central1(config-if)#int s0/0
Central1(config-if)#ipv6 address 2001:db8:3c4d:12::/64 eui-64
Central1(config-if)#int s0/1
Central1(config-if)#ipv6 address 2001:db8:3c4d:13::/64 eui-64

DFW(config)#ipv6 unicast-routing
DFW(config)#int s0/0/0
DFW(config-if)#ipv6 address 2001:db8:3c4d:12::/64 eui-64
DFW(config-if)#int fa0/0
DFW(config-if)#ipv6 address 2001:db8:3c4d:14::/64 eui-64

HOU(config)#ipv6 unicast-routing
HOU(config)#int s0/0/1
HOU(config-if)#ipv6 address 2001:db8:3c4d:13::/64 eui-64
HOU(config-if)#int f0/0
HOU(config-if)#ipv6 address 2001:db8:3c4d:15::/64 eui-64
```

***Static IPv6 Routing***

First static route line uses the next-hop address, and the exit interface on the second entry

*(On the DFW router, use show ipv6 int brief, and then copy the interface address used for the next hop)*

```
Central1(config)#ipv6 route 2001:db8:3c4d:14::/64  2001:DB8:3C4D:12:21A:2FFF:FEE7:4398 150
Central1(config)#ipv6 route 2001:DB8:3C4D:15::/64 s0/1 150
Central1(config)#do sho ipv6 route static
        S   2001:DB8:3C4D:14::/64 [150/0]
            via 2001:DB8:3C4D:12:21A:2FFF:FEE7:4398
```

*For DFW and HOU routers put a single entry in each router to get to remote subnet 11 (Central1):*

```
DFW(config)#ipv6 route 2001:db8:3c4d:11::/64 s0/0/0 150
HOU(config)#ipv6 route ::/0 s0/0/1  -- a default route
```

***IPv6 RIPng (a different pair of routers for this one)***

```
Austin(config)#ipv6 unicast-routing
Austin(config)#interface fastethernet 0/0
Austin(config-if)#ipv6 enable
Austin(config-if)#ipv6 address 2001:db8:c18:2::/64 eui-64
Austin(config-if)#ipv6 rip RIPNG1 enable
Austin(config-if)#interface fastethernet 0/1
Austin(config-if)#ipv6 enable
Austin(config-if)#ipv6 address 2001:db8:c18:1::/64 eui-64
Austin(config-if)#ipv6 rip RIPNG1 enable
Austin(config-if)#no shutdown

Houston(config)#ipv6 unicast-routing
Houston(config)#interface fastethernet 0/0
Houston(config-if)#ipv6 enable
Houston(config-if)#ipv6 address 2001:db8:c18:2::/64 eui-64
Houston(config-if)#ipv6 rip RIPNG1 enable
Houston(config-if)#interface fastethernet 0/1
Houston(config-if)#ipv6 enable
Houston(config-if)#ipv6 address 2001:db8:c18:3::/64 eui-64
Houston(config-if)#ipv6 rip RIPNG1 enable
```

***Switch to IPv6 OSPF Routing***

```
Central1(config)#int f0/0
Central1(config-if)#ipv6 ospf 1 area 0
Central1(config-if)#int s0/0
Central1(config-if)#ipv6 ospf 1 area 0
Central1(config-if)#int s0/1
Central1(config-if)#ipv6 ospf 1 area 0

DFW(config)#int f0/0
DFW(config-if)#ipv6 ospf 1 area 0
DFW(config-if)#int s0/0/0
DFW(config-if)#ipv6 ospf 1 area 0
[Same for others]

SanAnt(config)#int f0/0
SanAnt(config-if)#ipv6 address autoconfig default
SanAnt(config-if)#ipv6 ospf 1 area 0
```

*The same IPv4 OSPF RID is still there- change the RID under the OSPF process ID in the global config:*

```
Central1(config)#ipv6 router ospf 1
Central1(config-rtr)#router-id 1.1.1.1
Central1(config-rtr)#do clear ip ospf process
```

***Switching from IPv6 OSPF to IPv6 EIGRP***

```
        (you have to turn off OSPF on the interfaces, then add)
Central1(config)#int f0/0
Central1(config-if)#no ipv6 ospf 1 area 0
Central1(config-if)# ipv6 eigrp 1
Central1(config-if)#exit
        [Do for all interfaces]
DFW(config)#int f0/0
DFW(config-if)#no ipv6 ospf 1 area 0
DFW(config-if)# ipv6 eigrp 1
DFW(config-if)#exit
DFW(config-if)#int s0/0/0
 [Same for others]
```

# ***Dynamic IGP Routing Protocols - Quick Overview***

**RIP for IPv4**
Central1(config)#router rip
Central1(config-router)#network 10.0.0.0
Central1(config-router)#network 172.16.0.0
Central1(config-router)#version 2
Central1(config-router)#no auto-summary
*Add the default as a static route, and enter RIP config to set default-info originate to propagate it.*
Central1(config)#ip route 0.0.0.0 0.0.0.0 Fa0/0
Central1(config)#router rip
Central1(config-router)#default-information originate

**RIPng for IPv6 - add to interfaces**
Austin(config)#ipv6 unicast-routing
Austin(config)#interface fastethernet 0/0
Austin(config-if)#ipv6 enable
Austin(config-if)#ipv6 address 2001:db8:c18:2::/64 eui-64
Austin(config-if)#ipv6 rip RIPNG1 enable
Austin(config-if)#interface fastethernet 0/1
Austin(config-if)#ipv6 enable
Austin(config-if)#ipv6 address 2001:db8:c18:1::/64 eui-64
Austin(config-if)#ipv6 rip RIPNG1 enable
Austin(config-if)#no shutdown

**OSPF for IPv4**
int loopback 0
ip address 172.31.1.1 255.255.255.255
router ospf 300
Test(config-router)#network 192.168.10.64 0.0.0.15 area 0
Test(config-router)#network 192.168.10.80 0.0.0.15 area 0

**OSPFv3 for IPv6 - add to interfaces**
Corp(config)#ipv6 router ospf 1
Corp(config)#ipv6 router-id 1.1.1.1
Corp(config)#int f0/0
Corp(config-if)#ipv6 ospf 1 area 0

**EIGRP for IPv4**
Corp#config t
Corp(config)#router eigrp 20
Corp(config-router)#network 10.10.11.0 0.0.0.255
Corp(config-router)#network 172.16.10.0 0.0.0.3
Corp(config-router)#network 172.16.10.4 0.0.0.3
Corp(config-router)#no auto-summary
---
SF(config)#router eigrp 20
SF(config-router)#network 172.16.0.0
SF(config-router)#network 10.0.0.0
SF(config-router)#no auto-summary

**EIGRPv6 for IPv6 - add to interfaces**
Corp(config)#ipv6 unicast-routing
Corp(config)#ipv6 router eigrp 10
Corp(config-rtr)#no shut
Corp(config-rtr)#router-id 1.1.1.1
Corp(config-rtr)#int s0/0/0
Corp(config-if)#ipv6 eigrp 10
Corp(config-if)#int s0/0/1
Corp(config-if)#ipv6 eigrp 10

## ***Bigger Picture: Routing Table Manager; Timers***

**Administrative Distance- Lowest score wins**
*How the device's Routing Table Manager (RTM) chooses from multiple active routing protocols available to get to the same subnet advertised on one of them. Got a router with a few turned on? Lower AD is chosen.*

***To understand this, a directly connected device has an AD of 0 (as it should)***

| | | | |
|---|---|---|---|
| Static route | 1 | IS-IS | 115 |
| EIGRP summary | 5 | RIP | 120 |
| external BGP | 20 | external EIGRP | 170 |
| Internal EIGRP | 90 | internal BGP | 200 |
| OSPF | 110 | Unknown* | 255 |

***Timers - How to choose a good protocol in your designs:***
These are the defaults. Refer to documentation to how to change them, like "ip hello-interval eigrp" or "ip hold-time eigrp"

| Protocol | Type | Hello/Keepalive | Hold/Dead | Notes |
|---|---|---|---|---|
| ***Interior Routing Protocols*** | | | | |
| RIP | DV | Update 30 sec | 90 sec | Timeout 180 sec; Flush 120 sec |
| OSPF | LS | BC+PtP 10 sec | BC+PtP 40 sec (4x hello) | For NBMA 30 sec hello/ 120 sec dead (4x) |
| EIGRP | AdDV | BC+PtP 5 sec | BC+PtP 15 sec (3x hello) | For NBMA 60 sec hello/ 180 sec dead (3x) |
| ***Exterior Routing Protocols*** | | | | |
| IS-IS | LS | 10 sec<br>Juniper 9 sec? | 30 sec (3x hello)<br>Juniper 27 sec? | Also has L1 and L2 timers. PtP vs NBMA. Differ in Juniper. See documentation. |
| *(IS-IS has several timers and issues that are outside the scope of this table)* | | | | |
| BGP | PathV | 60 sec keepalive | 180 sec (3x keepalive) | |
| *(BGP also has iBGP and eBGP with timers outside the scope of this table)* | | | | |

***OSPF Path Cost in Routing - Don't get confused with Spanning Tree Cost in Switching!***
Choosing the best route - cost=reference_bandwidth /interface_bandwidth

| Interface | Default Bandwidth | Formula (Kbps) | OSPF Cost |
|---|---|---|---|
| Serial | 1544 Kbps | 100,000/1544 | 64 |
| Ethernet | 10,000 Kbps | 100,000/10,000 | 10 |
| Fast Ethernet | 100,000 Kbps | 100,000/100,000 | 1 |

The default ref bandwidth assumes the fastest link in the network runs at 100 Mbps.
Use **auto-cost reference-bandwidth 10000 to** accommodate links up to 10 Gbps in speed.

Here are the rules for how a router sets its OSPF interface costs:
- Set the cost explicitly, with **ip ospf cost x interface**, to a value between 1-65,535
- To affect all routes on the device, use **distance 80**. This also affects the AD of non-OSPF routes.
- Change interface bandwidth with the **bandwidth** *speed* command, with speed in Kbps
- Change the reference bandwidth, using subcommand **auto-cost reference-bandwidth** *ref-bw,* in Mbps

**STP Path Cost:**

| | | |
|---|---|---|
| 10 Mbps | 100 | 2000000 |
| 100 Mbps | 19 | 200000 |
| 1 Gbps | 4 | 20000 |
| 10 Gbps | 2 | 2000 |
| 100 Gbps | | 200 |

**Distance vector routing:**
Split Horizon is just not advertising the route back from where it was learned from. Poison reverse labels "where it came from" an unreachable link with "infinite metric" (16 for RIP)
***Split-horizon routing with poison reverse*** is a variant of split-horizon route advertising in which a router actively advertises routes as **unreachable** over the interface over which they were learned by setting the route metric to infinite (16 for RIP). The effect of such an announcement is to immediately remove most looping routes before they can propagate through the network.

The main disadvantage of poison reverse is that it can significantly increase the size of routing announcements in certain fairly common network topologies, but it allows for the improvement of the overall efficiency of the network in case of faults. Split horizon states that if a neighboring router sends a route to a router, the receiving router will not propagate this route back to the advertising router on the same interface.

With route poisoning, when a router detects that one of its connected routes has failed, the router will poison the route by assigning an infinite metric to it and advertising it to neighbors. When a router advertises a poisoned route to its neighbors, its neighbors break the rule of split horizon and send back to the originator the same poisoned route, called a poison reverse. In order to give the router enough time to propagate the poisoned route and to ensure that no routing loops occur while propagation occurs, the routers implement a hold-down mechanism.

# *BGP4 Overview*

- Built for reliability, scalability, and control - not speed.
- Is a **path-vector protocol**. Routing between AS's is called *interdomain routing*
- Uses TCP port 179. BGP peers exchange incremental, triggered route updates and periodic keepalives.
- The administrative distance for eBGP routes is 20, for iBGP routes is 200.
- Routers run only one instance/process of BGP at a time; are called *BGP speakers*; neighbors are *BGP peers*
- Networks not in the local router's routing table will not be sent out in updates
- Routes learned by the BGP process are propagated by default but are often filtered by a routing policy.
- When an update about a network leaves an AS, it's ASN is prepended to the list of ASs that have handled that update. Once it receives an update, it examines the AS list. If it finds its own ASN in that list, the update is discarded (loop prevention/built in split-horizon)
- no auto-summary and no synchronization are defaults in BGP

**Because BGP uses TCP, it has reliability, error recovery, and flow control.**
Before a BGP speaker can peer with neighbor, neighbor must be statically defined; TCP session-capable, IP reachable. You also get the buffering and other TCP benefits (resent segments if timer reaches 0, acknowledgement may be delayed up to a 1 second to determine if any data should be sent, etc.) The underlying TCP session can be shown with 'show tcp brief'.

**Autonomous Systems and ASNs**
AS is a group of networks under a common administration. IANA ASN range from 0 to 65,535, with 64,512 - 65,534 ASNs to be private (private means can connect to only one other ASN or multiple ASNs that can't cause loop.

| **ASN range** | 0 | 1 - 64,495 | 64,496 - 64,511 | 64,512 - 65,534 | 65,535 |
|---|---|---|---|---|---|
| **Purpose** | Reserved | Public ASN | Documentation | Private ASN | Reserved |

Public ASNs are assigned and registered by RIPE NCC; private ASNs can be removed in eBGP using the 'neighbor ebgp-neighbor-address remove-private-as' command.

**Path Attributes: AS_PATH and routing**
- Path attributes (PAs) are factors that allow BGP to select a route over another. By default, no BGP PAs have been set, and BGP uses AS_PATH PA when choosing the best route among many routes.
- When a router advertises a route with AS_PATH, it will list ASNs the path will go through, to help find the shortest path and prevent routing loops.
- Looping is prevented by ignoring route updates that contain the current AS's ASN.

**Internal and External BGP**
BGP peering used: iBGP if same AS, eBGP if different AS. If the ASN configured in a device's **router bgp** and **neighbor** statements match, BGP initiates an internal session (iBGP); otherwise an external session (eBGP).
When advertising to an iBGP peer, no ASN is added- not needed. To an eBGP peer, it's ASN is added.
There are 2 types of routing:
- Hot-potato routing: traffic exits the AS via the closest exit point.
- Cold-potato routing: traffic exits the AS via the path closest to the destination.

**BGP Databases**
BGP uses three databases:
- BGP DB, or RIB (Routing Information Base): Networks known by BGP, with their paths PAs **- show ip bgp**
- Routing table: Paths to each network used by the router, and the next hop for each **- show ip route**
- Neighbor database: All configured BGP neighbors. **show ip bgp summary**

**BGP States and Message Types**
The BGP process has different states, managed by a Finite State Machine (FSM). The messages don't have specific names like LSA's do in OSPF, just typical hello, update, etc.
1. Router tries to establish TCP connection with IP address configured in 'neighbor' command at port 179
2. After 3-way handshake, first BGP message sent (Open), with parameters to establish neighborship.
3. If a match, neighborship formed (Established), update messages sent (list of PAs and prefixes)

| State | Meaning |
|---|---|
| Idle | Administratively down or waiting- needs TCP session initiated by peer. Listening. |
| Connect | TCP handshake. If successful, router sends Open message and changes to OpenSent. If not, router resets the ConnectRetry timer and goes to Active state.If timer reaches 0 while the router is in Connect state, timer is reset and another attempt is made, remaining in Connect. |
| Active | TCP connection ok, but no BGP messages sent to peer yet. Can also mean Connect wasn't successful so it's trying again. If ConnectRetry timer expires, it kicks back to Connect state.<br>If peering stops here, can also mean TCP session attempts were made from an unexpected IP address, which are rejected. If that's the case, it stays in Active and resets ConnectRetry. (attack vector?) |
| OpenSent/ | TCP connection exists, an Open message has been sent to peer. Transitions to OpenReceive to wait for |

| | |
|---|---|
| | initial keepalive from peer to move into OpenConfirm state. |
| OpenReceive | If TCP disconnect received, router terminates session, resets ConnectRetry timer, and goes back to Active. |
| OpenConfirm | Open messages finished, initial keepalive received. Sends Keepalives to peer to see configs match, and listens. On receipt of keepalive, moves to Established, else moves to Idle |
| Established | Achieved after receiving more Keepalives from peer. All neighbor parameters match, the neighbor relationship works, and the peers can now exchange Update messages. |

To check state, use **sh ip bgp summary**. The State/PfxRcd column shows a numeric value for a state of Established- if not in an Established state, the value in this heading lists the text name of the current BGP state

All BGP messages share the same header, composed of:
- 16 byte marker field: set to all 1s to detect a loss of synchronization.
- 2 byte length field: indicate total length of BGP message, range from 19 to 4096
- There is a PA field in Update (AS Path Attribute)
- 1 byte type field: type code, indicate different BGP messages - the number in () below0

| Message (type#) | Purpose |
|---|---|
| Open (1) | First message after TCP connection has established to get neighborship started.<br>BGP version, ASN, hold timer, BGP identifier, optional authentication |
| Keepalive (4) | Maintain peering. Contains only header. Keepalives exchanged every 60 sec by default. By default, hold timer is 3x keepalive interval. Updates can also reset keepalive interval |
| Update (2) | Used to exchange PAs and the associated prefix/length (NLRI) that uses those attributes. Each contains a single set of PAs and all related NLRI. Also contain withdrawn routes. |
| Notification (3) | Signals error with code, subcode, and data. Often signals to resets connection |

All prefixes, except filtered by **neighbor <ipaddr> route-map <name> in**, are in the routing table for calculation.

**eBGP neighborship**
To configure BGP, you need at least 2 commands:
config)#router bgp <asn>
config-router)#neighbor <ipaddr> remote-as <remote-asn>

For BGP neighbor relationship to form,
- TCP connection between them must first be established
- Both routers need complimentary **neighbor <ipaddr> remote-as <asn>** with matching ASN and the IP address of the interface for which TCP packets will exit (if the interface is not explicitly defined)
- BGP RID must not be the same
- MD5 authentication must pass (if configured)

BGP RID is established by
1) **bgp router-id <rid>** - OR -
2) highest IP address of any up/up loopback interfaces at BGP initialization
3) highest IP address of any up/up normal interface at BGP initialization

To configure authentication, use **neighbor <ipaddr> password <password>** on both routers. The <ipaddr> refers to the IP address of the other router, while password must be identical for neighborship to succeed.

To verify neighborship **show ip bgp neighbors**, and **show tcp brief** to display the TCP connection
Use **neighbor <ipaddr> shutdown** to shut down that connection, move to Idle (keeps neighbor config).

The 'neighbor' command can be configured with the **peer-group** parameter, so one set of commands can be applied to all neighbors in the peer-group (either external or internal, not both)
Simplify configuration and reduce updates.
neighbor <name> peer-group
neighbor <ipaddr> peer-group <name>
(neighbor must have 'neighbor remote-as' set).

**In most IGPs, the network command starts the routing process on an interface.**
**In BGP, the command tells the router to originate an advertisement for that network.**
- network does not have to be connected - it just has to be in the routing table.
- can even be a network in a different AS (not usually recommended, but next-hop-self sort of does it)

BGP assumes you are using the default classful subnet mask.
- need to advertise a subnet? Use the optional keyword **mask** and specify the subnet mask to use
- routing table must contain an exact match (prefix and subnet mask)

**Network declarations need to be SPECIFIC**

If you misconfigure a **network** command, like **network 192.168.1.1 mask 255.255.255.0**, BGP will look for exactly 192.168.1.1/24 in the routing table. With no match, BGP won't announce it to neighbors.

If you issue the command **network 192.168.0.0 mask 255.255.0.0** to advertise a CIDR block, BGP will look for ONLY 192.168.0.0/16 in the routing table. It may find 192.168.1.0/24 or 192.168.1.1/32; however, it may never find 192.168.0.0/16. In this case, you can configure a static route towards a null interface so BGP can find an exact match in the routing table: **ip route 192.168.0.0 255.255.0.0 null0**

**BGP and Loopback Addresses**

- Recall that a loopback interface provides a more robust configuration - it never goes down, so it adds more stability than physical the ones, and BGP will still operate if the link to the closest interface fails.

If the physically connected port is down:

- The router can still send the packet without being interrupted because loopback is always up.
- If another interface tries to reach the neighbor it will be blocked when source address doesn't match.

When multiple links exist between 2 BGP routers, and you would like to establish BGP neighborship between them, there are 2 options:

- configure a connection for each physical interface; consumes bandwidth and memory.
- Configure connections using virtual (loopback) interfaces, which requires less bandwidth and memory, and ensure the interface is always up/up.

**eBGP** assumes that neighbors are directly connected and peering with the IP of that specific interface. If not, you must use **neighbor *ip-address* ebgp-multihop *number-of-hops*** command. If you are peering with loopback interface IP addresses, you are going to have to use it.

**iBGP** assumes that internal neighbors might not be directly connected, so this command is not needed. With loopback IP addresses in iBGP, you must change the source to match the loopback with **neighbor** *ip-address* **update-source** *interface*

Router(config-router)**#neighbor 172.16.1.2 update-source loopback0**

- Without neighbor **update-source**, iBGP will use the closest IP interface to the peer. In the case of a point-to- point eBGP session, this command is not needed because there is only one path for BGP to use.

Using loopback and update-source for connections whether iBGP or eBGP is considered a best practice. It ensures an alternate route to the specific router is available if the physical interface goes down

# *BGP Path Attributes - Manipulating Path Selection*

Routes learned via BGP have properties referred to as *BGP attributes*, and an understanding of how they influence route selection is required for the design of robust networks.

| PA | Description | Route Direction |
|---|---|---|
| NEXT_HOP | Next-hop IP address used to reach a prefix. | N/A |
| Weight[1] | Range 0 - 2^16 -1, set when receiving updates. Not advertised to any BGP peers. Cisco proprietary | Outbound |
| LOCAL_PREF | Range 0 - 2^32 -1, set and spread throughout an AS to influence the choice of best route for all routers in that AS | Outbound |
| AS_PATH (length) | The number of ASNs in the AS_Path PA. | Outbound, Inbound |
| ORIGIN | Value implying the route was injected into BGP; I (IGP), E (EGP), or ? (incomplete information). | Outbound |
| Multi-Exit Discrim (MED) | Set and advertised by routers in an AS, impacting the BGP decision of routers in the other AS. Smaller is better. | Inbound |

**Four categories of attributes exist as follows:**
- Well-known mandatory: Must be recognized by all BGP routers, present in all BGP updates, and passed on to other BGP routers. For example, **AS path, origin, and next hop.**
- Well-known discretionary: Must be recognized by all BGP routers and passed on to other BGP routers but need not be present in an update, for example, **local preference.**
- Optional transitive: Might or might not be recognized by a BGP router but is passed on to other BGP routers. If not recognized, it is marked as partial, for example, **aggregator, community.**
- Optional nontransitive: Might or might not be recognized by a BGP router and is not passed on to other routers, for example, **Multi-Exit Discriminator (MED), originator ID.**

**BGP Path Selection Criteria**
1. Highest weight (most local).
2. Highest LOCAL_PREF (inside an AS this is global).
3. Choose routes that this router originated (next-hop = 0.0.0.0)
4. Shortest AS_PATH
5. Lowest origin type/code - (i)iBGP is lowest, (e)eBGP is next, and ? is last
6. Lowest MED (if the same AS advertises the possible routes)
7. eBGP wins over an iBGP route.
8. Nearest IGP neighbor (lowest IGP metric)
9. Oldest route (for eBGP to minimize the effects of route flapping)
10. Lowest router ID.
11. Lowest IP address.

BGP only chooses one route as the best route (thus, no load balancing). Step 1, 2, and 4 are typically used to influence outbound routes, while MED for outbound routes. 9, 10 and 11 aren't used except in a tie
Mnemonic: N WLLA OMNI

| Step | Mnemonic Letter | Short Phrase | Which Is Better? |
|---|---|---|---|
| 0 | N | Next hop: reachable? | If no route to reach Next_Hop, router cannot use this route. |
| 1 | W | Weight | Bigger weight. |
| 2 | L | LOCAL_PREF | Larger preference. |
| 3 | L | Locally injected routes | Locally injected ‘network’ is better than iBGP/eBGP learned. |
| 4 | A | AS_PATH length | Smaller paths win |
| 5 | O | ORIGIN | Prefer I over E. Prefer E over ? |
| 6 | M | MED | Smaller Multi-Exit Discriminator (MED) |
| 7 | N | Neighbor Type | Prefer eBGP over iBGP. |
| 8 | I | IGP metric to Next_Hop | Smaller metric wins. If no IGP is used, consider tied. |
| 9 | A | Age equals seniority | Oldest eBGP route |
| 10 | O | think not OSPF | Route with lowest BGP RID |
| 11 | S | think not OSPF | Route with lowest neighbor IP address |

# *BGP - Basic Configuration*

| | |
|---|---|
| **router bgp 100**. | Starts BGP routing process 100 |
| **neighbor 192.31.7.1 remote-as 200** | Add a peer for sessions. The ASN tells if eBGP or iBGP |
| **network 192.135.250.0** | What locally learned networks to advertise. |
| **network 128.107.0.0 mask 255.255.255.0** | Specify an individual subnet. |
| (**no) neighbor 24.1.1.2 shutdown**" | Disable/-enable an active session |
| **timers bgp 90 240** | Timers. Keepalive (default 60sec) Holdtime (default 180sec) |
| **neighbor 172.16.1.2 update-source loopback0** | Use this specific interface |
| (**no**) **synchronizatio**n | iBGP route must be added to table by other IGP before used |

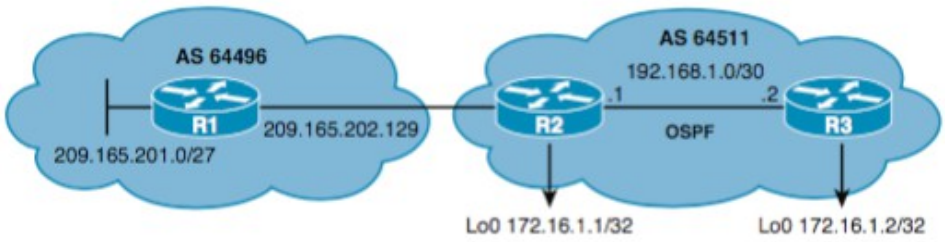
Sychronization is off by default in later IOS. Using iBGP meshes is preferred. Redistribution from BGP into an IGP when using BGP for MPLS is reasonable and commonly done. See SWITCH book pg 617 on avoiding loops

**iBGP Next-Hop Behavior**

- The eBGP next-hop attribute is the IP address that is used to reach the advertising router; an edge router belonging to the autonomous system "next door", in most cases, the IP address of the connection between peers.
- For iBGP, the eBGP next-hop address is not changed, carried into the local autonomous system as-is.

The next-hop-self command is a way to get around this (illustrated below)

- R2 tells R3 that it instead will be the next-hop address when trying to reach networks outside its autonomous system.

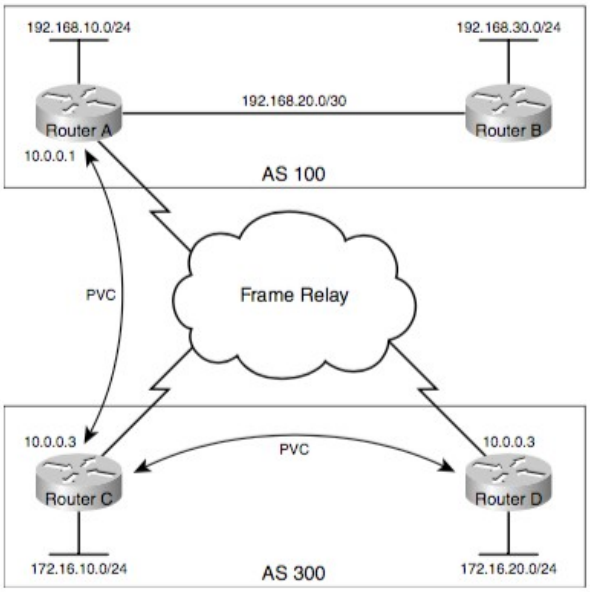



***Setup on R2:***

| | |
|---|---|
| **router bgp 64511** | Starts the BGP routing process. |
| **neighbor 209.165.202.129 remote-as 64496** | Identifies R1 as an eBGP neighbor |
| **neighbor 172.16.1.2 remote-as 64511** | Identifies R3 as an iBGP neighbor |
| **neighbor 172.16.1.2 update-source loopback0** | Loopback0 IP is source for all BGP TCP packets to R3 |
| **neighbor 172.16.1.2 next-hop-self** | Advertises itself as next hop for networks from other AS |

*R3 will then use R2 as the next hop instead of using the eBGP next-hop of 209.165.202.129.*

**Here is another example in a slightly different setup where it is almost mandatory:**



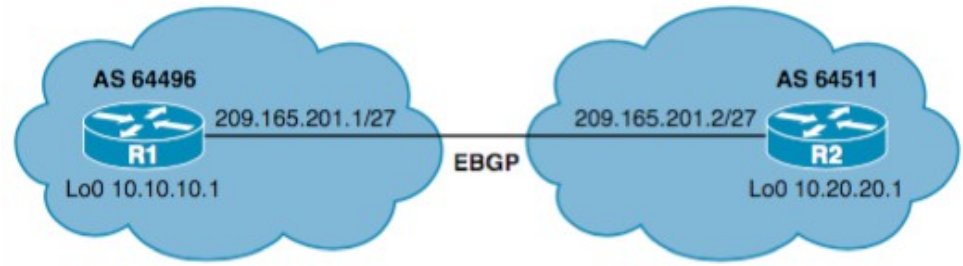

RouterC(config)**#router bgp 300**
RouterC(config-router)**#neighbor 10.0.0.1 remote-as 100**
RouterC(config-router)**#neighbor 10.0.0.1 next-hop-self**
- Forces all updates to 10.0.0.1 to advertise this router as next hop

- Router C advertises 172.16.20.0 to Router A with next hop of 10.0.0.3 as if the common media were Ethernet.
- Routing will fail - Router A has no direct PVC to Router D and cannot reach the next hop
- To remedy this situation, neighbor next-hop-self causes Router C to advertise 172.16.20.0 with next-hop set to it's 10.0.0.3.
- This proves useful in non-meshed networks (such as Frame Relay or X.25) where BGP neighbors might not have direct access to all other neighbors on the same IP subnet.

RouterC(config-router)**#neighbor 10.0.0.1 next-hop-unchanged**
Enables an eBGP multihop peer to propagate the next hop unchanged.

Do not use with route reflector setups
This command should not be configured on a route reflector, and the neighbor next-hop-self command should not be used to modify the next-hop attribute for a route reflector when this feature is enabled for a route reflector client.

**eBGP Multihop**
By default, eBGP neighbors exchange packets with a TTL set to 1. Even though eBGP neighbors are usually directly connected (over a WAN connection) to establish sessions, sometimes one of the directly connected routers is unable to run BGP. The command **ebgp-multihop** allows for a logical connection to be made between peer routers, even if not directly connected - up to 255 hops away and still be able to create an eBGP session. The **ebgp-multihop** is only used for eBGP sessions, and must be configured on each peer (each end). If you attempt to establish eBGP session between loopbacks, BGP packets will be dropped due to an expired TTL.




R1(config)**# ip route 10.20.20.1 255.255.255.255 209.165.201.2**
- Define a static route to Loopback0 on R2.
R1(config)**# router bgp 64496**
R1(config-router)**# neighbor 10.20.20.1 remote-as 64511**
- Identifies a peer router at 10.20.20.1

R1(config-router)# **neighbor 10.20.20.1 update-source loopback0**
R1(config-router)# **neighbor 10.20.20.1 ebgp-multihop 2**
- Allows for two routers (not directly connected) to establish an eBGP session with TTL of 2.

R2(config)# **ip route 10.10.10.1 255.255.255.255 209.165.201.1**
- Define a static route to Loopback0 on R1.

R2(config)# **router bgp 64511**
R2(config-router)# **neighbor 10.10.10.1 remote-as 64496**
R2(config-router)# **neighbor 10.10.10.1 update-souce loopback0**
R2(config-router)# **neighbor 10.10.10.1 ebgp-multihop 2**
- Allows for two routers (not directly connected) to establish an eBGP session with TTL of 2.

**Loopbacks on eBGP need ebgp-multihop**
If redundant links exist between two eBGP neighbors and loopback addresses are used, you must configure **ebgp-multihop** because of the default TTL of 1. Otherwise, the router decrements the TTL before giving the packet to the loopback interface, meaning that the normal IP forwarding logic discards the packet. Configuring the value to 2 solves the problem.

**Default Routes, default-originate to give default to specific neighbors**
For a default route to send to all neighbors/peers you just use **network 0.0.0.0** (be sure it's in the routing table)
For the default route 0.0.0.0 to only be advertised to a specific neighbor, use **neighbor 192.168.100.1 default-originate**

**BGP Peer Groups**
To ease the burden of configuring a large number of neighbors with identical or similar parameters (for example, route maps, filter lists, or prefix lists), the concept of peer groups was introduced. Simply configure a peer group with all the BGP parameters that are to be applied to many BGP peers. BGP neighbors are bound to the peer group, and the network administrator applies the peer group configuration on each of the BGP sessions. The result below, all four iBGP neighbors have the same basic BGP configuration assigned to them.

| | |
|---|---|
| **router bgp 65500** | Starts the BGP routing process |
| **neighbor INTERNAL peer-group** | Creates a BGP peer group called INTERNAL |
| **neighbor INTERNAL remote-as 65500** | Assigns a first parameter to the peer group |
| **neighbor INTERNAL next-hop-self** | Assigns a second parameter to the group |
| **neighbor INTERNAL update-source loopback0** | Assigns a third parameter to the peer group |
| **neighbor INTERNAL route-reflector-client** | Assigns a fourth parameter to the peer group |
| **neighbor 192.168.1.2 peer-group INTERNAL** | Assigns the peer group to neighbor R2 |
| **neighbor 192.168.1.3 peer-group INTERNAL** | ... to neighbor R3 |
| **neighbor 192.168.1.4 peer-group INTERNAL** | ... to neighbor R4 |
| **neighbor 192.168.1.5 peer-group INTERNAL** | ... to neighbor R5 |

A peer group can be, among others, configured to do the following:
- Use IP of a specific interface as source address when opening the TCP session or use next-hop-self feature
- Use, or not use, things like the eBGP multihop function, or MD5 authentication on the BGP sessions.
- Filter out any incoming or outgoing routes using a prefix list, a filter list, and a route map.
- Assign a particular weight value to the routes that are received.

**MP-BGP**
Original BGP was designed for only IPv4. Multiprotocol BGP can run over as IPv6, multicast IPv4, and MPLS - can exchange routes for IPv4, IPv6, or both. The extensions enable NEXT_HOP to carry IPv6 addresses and NLRI (Network Layer Reachability Information) to an IPv6 prefix, thanks to TCP. The IPv4 and IPv6 routes use separate TCP connections.

R1(config)#**ipv6 unicast-routing**
R1(config)#**router bgp 65500**
R1(config-router)#**neighbor 2001:0DB8:12::2 remote-as 65501**
R1(config-router)#**neighbor 192.168.1.2 remote-as 65501**
R1(config-router)#**address-family ipv4 unicast**
- Enters IPv4 address family configuration mode for unicast prefixes (default IPv4)

R1(config-router-af)#**neighbor 192.168.1.2 activate**
- Enables the exchange of IPv4 BGP info with R2. IPv4 neighbors auto-activate, so keyword is optional.

R1(config-router-af)#**network 10.1.1.1 mask 255.255.255.255**
R1(config-router)#**address-family ipv6 unicast**
- Enters IPv6 address family configuration mode for unicast address prefixes.

R1(config-router-af)#**neighbor 2001:0DB8:12::2 activate**
- Enables the exchange of IPv6 BGP information with R2.

R1(config-router-af)#**network 2001:0DB8:1::1/64**

Router2 would have a reciprocal set of the same commands to work with R1.
Recall that the router ID is set on IPv4 addresses (**bgp router-id** *IPv4_address)*, IPv6 can still use it
Troubleshoot with **show bgp ipv6 unicast [ optional: summary | neighbors ]** and **show ipv6 route bgp**

**Route Aggregation - Setting the Atomic Aggregate attribute**

- A BGP router can transmit overlapping routes (nonidentical routes pointing to the same destination)
- When making a best path decision, a router always chooses the more specific path.

R1(config-router)**#aggregate-address 172.16.0.0 255.255.0.0**

- Creates aggregate entry in the BGP table to cover specific BGP routes in that range.
- More specific routes will still be sent unless **summary-only** is used.

R1(config-router)**#aggregate-address 172.16.0.0 255.255.0.0 summary-only**

- Creates aggregate route AND suppresses advertisement of more-specific routes.
- Specific AS_PATH info on those subnets are lost.

R1(config-router)**#aggregate-address 172.16.0.0 255.255.0.0 as-set**

- Creates an aggregate entry but the path advertised will be an AS_SET (list of AS_PATHs)

Below is a use of aggregate route as a precaution: to send an aggregate address, we only need one of the more specific routes configured. But by configuring all of them, and *not* using **summary-only**, they will all be sent, and the aggregate will be sent in case one of the networks goes down.

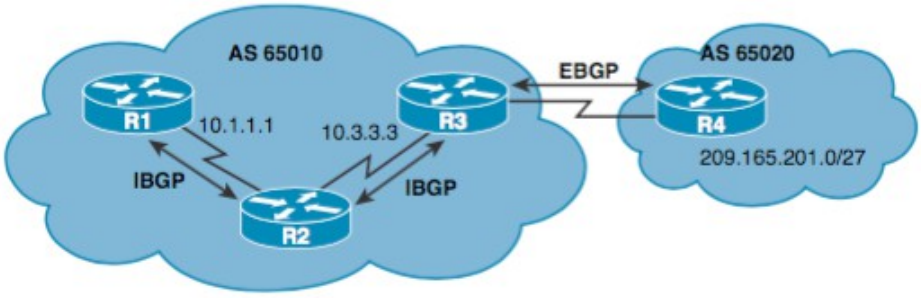
Lubbock(config)#router bgp 1
Lubbock(config-router)#neighbor 10.1.1.2 remote-as 2
Austin(config)#router bgp 2
Austin(config-router)#neighbor 10.1.1.1 remote-as 1
Austin(config-router)#network 172.16.0.0 mask 255.255.255.0
Austin(config-router)#network 172.16.1.0 mask 255.255.255.0
Austin(config-router)#network 172.16.2.0 mask 255.255.255.0
Austin(config-router)#aggregate- address 172.16.0.0 255.255.252.0

Thus, both Lubbock and Austin will have all the specific routes *and* the aggregate address in its BGP table

```
Lubbock#show ip bgp 172.16.0.0 255.255.252.0
BGP routing table entry for 172.16.0.0/22, version 18
Paths: (1 available, best #1)
<text omitted>
 Origin IGP, localpref 100, valid, external, atomic-aggregate,
  best
```

**Route Reflectors**

By default, a router that receives an eBGP route advertises it to its eBGP and iBGP peers.
If it receives it through iBGP, it won't advertise it to its iBGP peers, as loop-prevention
One way for all iBGP routers to receive a route after it is originated into the AS is to have a full mesh of iBGP peers, which can get complex with a large number of peers. Route reflectors eliminate resorting to that.




The objective is to allow R2 to advertise to R1 the 209.165.201.0/27 network learned from R3. Without these commands, R1 would never learn the 209.165.201.0/27 network without a full-mesh iBGP topology.

| | |
|---|---|
| R2(config)**#router bgp 65010** | Enters BGP routing configuration mode |
| R2(config-router)**#neighbor 10.1.1.1 route-reflector-client** | Designates a BGP route reflector and the specified neighbor as a client |
| R2(config-router)**#neighbor 10.3.3.3 route-reflector-client** | Designates a BGP route reflector and the specified neighbor as a client |

**Verifying and Troubleshooting BGP Connections**

| | |
|---|---|
| **show ip bgp** | Displays entries in the BGP routing table - AS_PATH with the last ASN; Internal is 'i'. '*' is valid, and '>' is best route |
| **show ip bgp** *prefix [subnet-mask]* | List possible routes, per prefix [e.g., for default 0.0.0.0 0.0.0.0] |
| **show ip bgp** <ipsubnet/cidr> **longer-prefixes** | Show which route is older by placing its entry later |
| **show ip bgp neighbors** | Info about BGP/ TCP connections to neighbors, neighbor info |
| **show ip bgp neighbors** <address> **[routes \| received-routes \| advertised-routes]** | Monitor routes received/ learned from a specific neighbor. |
| **show ip bgp rib-failure** | List networks not put into the Routing Information Base (RIB) and the reason not used |
| **show ip bgp summary** | Status of all BGP connections, memory use of BGP databases, activity stats and list of BGP neighbors |
| **show ip route bgp** | Displays the BGP entries from the routing table |

**show tcp brief** to display the TCP connection
Use **neighbor <ipaddr> shutdown** to shut down that connection, move to Idle (keeps neighbor config)
**show ip bgp [ peer-group | regexp | community-list ]**
**show access-lists** and **show ip access-lists**
**debug ip bgp, debug ip bgp events**, **debug ip bgp ipv4 unicast**, **debug ip bgp updates**

**clear ip bgp *** - Forces BGP to clear its table and the * resets all BGP sessions. Put IP for a specific neighbor
Using the form **clear ip bgp *** is both processor and memory intensive - use only in smaller environments.
However, you may need to use this form when the following changes occur: Additions or changes to the BGP-related access lists, weights, distribution lists, BGP timer's specifications, administrative distance, or route maps

**clear ip bgp 10.1.1.2 soft out** - The **soft** keyword forces the remote router to resend all BGP info **without** resetting the connection. Routes from this neighbor are not lost. Works when changing outbound policy, does not help if you are changing an inbound policy. The keyword **out** means for all outbound updates (inbound is **in**).

Hard reset can be harsh - the underlying TCP connection is zapped so neighborship drops, removes all BGP table entries from that neighbor. It's a long recovery that interrupts routing, causes flapping with disassociated peers, and the full set of routing updates generates tons of traffic. Soft resets don't bring down the neighborship or TCP connection, and just resends adjusted outgoing updates, which then adjust the BGP table. It is supported with route refresh capability included in all modern IOS versions

**neighbor 10.1.1.2 soft-reconfiguration inbound** Causes the router to store all updates from this neighbor. This is memory intensive, but is a safeguard before running **clear ip bgp** that a copy of the routing info is indeed saved first (arguably paranoid since there is the **clear ip bgp x.x.x.x soft** option)

**Routing Information Base failures (IP routing table omissions)**
When the BGP best-path algorithm has chosen a best route for a prefix, it gets handed to the Routing Table Manager (RTM) that weighs all routes by administrative distance before adding to the routing table [aka Routing Information Base (RIB)] Generally, a prefix learned with BGP should not conflict with a connected or IGP-learned route. Sometimes it arises when implementing MPLS VPNs with BGP/IGP redistribution. The **show ip bgp rib-failures** command lists routes which BGP has chosen as best, but the RTM function has not placed into the IP routing table.

As an example, consider two routers R1 and R2 finding their route to 185.0.0.0/8 - R1 learns about it from eBGP (AD of 20), but inside the AS, R2 sees it from both OSPF and iBGP. R2's BGP determined a best route, but the RTM instead chose the lower-AD OSPF route (AD 110) rather than the higher-AD iBGP route (AD 200). Running show ip bgp rib-failure shows one line for each best BGP route that the RTM does not place into the IP routing table. In this case, this command on 2 lists the route for 185.0.0.0/8.

```
E2# show ip bgp rib-failure
Network            Next Hop                  RIB-failure   RIB-NH Matches
185.0.0.0/8        10.100.1.1          Higher admin distance            n/a
```

## *Contents of show ip bgp - A Deeper Look*

The command **show ip bgp** shows the BGP topology database, which includes all the networks BGP knows about, the next hop, some of the attributes, and the AS path for each route.

```
route-server>show ip bgp
BGP table version is 22285573, local router ID is 12.0.1.28
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
       r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

  Network     Next Hop         Metric LocPrf Weight Path
* 3.0.0.0     12.123.137.124                 0     7018 2914 9304 80 i
*>            12.123.1.236                   0     7018 2914 9304 80 i
* 3.51.92.0/23 12.123.137.124                0     7018 ?
*             12.122.125.4     2366          0     7018 ?
*>            12.123.1.236                   0     7018 ?
* 8.6.6.0/24  12.123.137.124                 0     7018 701 14744 14744 14276 i
*             12.123.145.124                 0     7018 701 14744 14744 14276 i
*>            12.123.1.236                   0     7018 701 14744 14744 14276 i
```

Networks are listed in numerical order.
The first three columns list each route's status. They are one-character fields, squeezed together

The asterisk (*) in the first column means that the route has a valid next hop.
Other options:

- s - suppressed: BGP knows about the network but is not advertising it, often if part of a summarized route.
- d - dampened: BGP stopped advertising a network that flaps too often until it is stable for a period of time.
- h - history: BGP knows about this network but does not currently have a valid route to it.
- r - RIB failure: advertised route but not put in routing table. Another protocol may have route with a better AD.
- S - stale: Used with nonstop forwarding - route needs to be refreshed when the peer is reestablished.

The second column has a greater-than sign (>) beside the route that was selected as the best path to that network. In the example, the second route was selected for network 3.0.0.0.

The third column is blank in the example, which means that the router learned all the routes from an external neighbor. A route learned from an IBGP neighbor would have an "I" in the third column. (a - aggregate, i - internal)

The fourth column lists the networks. Those without a subnet mask, such as network 3.0.0.0, use their classful mask. As seen in the example, when the router learns about the same network from multiple sources, it lists only the network once.

The fifth column lists the next-hop address for each route. This might or might not be a directly connected router. A next-hop of 0.0.0.0 means that the local router originated the route (and/or, the router has non-BGP routes to the network).

Inter-autonomous system metric - If a Med value was received with the route, it is listed in the Metric column. Notice that the advertisement for network 3.51.92.0/23 from the router at 12.122.125.4 has a large Med value of 2366. Because the default Local Preference is used for each of the routes shown, no local preference value is displayed. The default Weight value of 0 is listed, however.

Toward the end is the AS path for each network. Reading this field from left to right, the first AS number shown is the adjacent AS this router learned the route from. After that, the AS paths that this route traversed are shown in order. The last AS number listed is the originating AS. In the example, our router received an advertisement about network 3.0.0.0 from its neighbor AS 7018, which heard about it from AS 2914, which heard about it from AS 9304. And AS 9304 learned the route from AS 80, which originated it. A blank AS path means that the route was originated in the local AS.

In the AS Path column, note that network 8.6.6.0 shows AS 14744 twice in its AS path list. Most likely AS 14744 has prepended an extra copy of its AS number to make the path through it less attractive than the path through other autonomous systems. In this case it did not work because the only paths to 8.6.6.0 this router knows about all go through AS 14744.

The last column shows how BGP originally learned about the route.

- Networks 3.0.0.0 and 8.6.6.0 show an "i" for their origin codes. This means that the originating router had a network statement for that route.
- Network 3.51.92.0 shows a "?" as its origin. This means that the route was redistributed into BGP; BGP considers it an "incomplete" route.
- You will likely never see the third possibility, an "e," because that means BGP learned the route from the Exterior Gateway Protocol (EGP), which is no longer in use.

If you specify an address this is how it will display:

router**#show ip bgp 66.66.66.66**

```
BGP routing table entry for 66.66.66.66/32
Paths: (2 available, best #1)
  66.66.66.66/32
  3 i comm 65535:65281
  172.16.23.3 from 172.16.23.3 (peer 3.3.3.3)
    Origin IGP, local pref 100, weight 0, valid, best
     BGP Path Selection Criterion: Lowest BGP Neighbor Router-ID
    IGP Metric: 0 IGP Pref: 0 IGP Protocol: DIRECT
    IGP Next Hop: 0.0.0.0 Route Age: 0:03:22
  66.66.66.66/32 (Second best)
  77 i comm 65535:65281
  172.16.12.1 from 172.16.12.1 (peer 172.16.12.1)
    Origin IGP, local pref 100, weight 0, valid
    IGP Metric: 0 IGP Pref: 0 IGP Protocol: DIRECT
    IGP Next Hop: 0.0.0.0 Route Age: 0:02:21
```

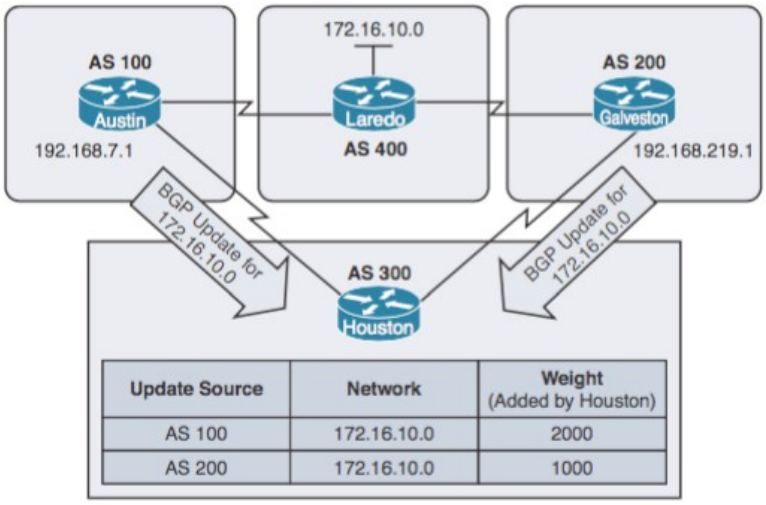
| | |
|---|---|
| Originator | Specifies the router ID of the originator of the route in the local AS. |
| Cluster list | A sequence of cluster ID values for the reflection path that the route has passed. |
| Path | Autonomous system path to the destination network. One entry in this field per autonomous system in the path. |
| Origin codes | Identifies the origin of the entry. Valid values previously mentioned (i, e, ?) |
| Origin | Identifies the origin of the entry as: IGP, EGP, Not clear, or Best. |
| LocPrf | Local preference value. See the set local-preference route map configuration command. Default value is 100. |
| Weight | Weight of the route as defined by set weight and router bgp route map |
| BGP path selection criteria | Displays tie-breaking criterion for best path selection. |
| IGP Metric | Specifies the IS-IS metric or OSPF cost value. |
| IGP Protocol | IGP Protocol: IS-IS or OSPF. |
| IGP Next-Hop | IP address of the next system used when forwarding packets to the destination network. 0.0.0.0 in this field indicates router has non-BGP routes to the network. |
| Route Age | Specifies the time in *hours*:*minutes*:*seconds* that a route has been valid. |
| Second best | Displays second best path information. |

# *Overview of Specific BGP Attributes*

**Weight Attribute**

The weight attribute is a proprietary Cisco attribute
- configured locally on a router and is not propagated to any other routers.
- applies when one router is used with multiple exit points out of an autonomous system
- (local preference attribute is used when two or more routers provide multiple exit points)

Routes with a higher weight are preferred when there are multiple routes to the same destination.
[value ranges from 0 to 65,535, 0 for learned routes, and 32,768 for locally injected routes]
By default, the weight attribute is 32,768 for paths that the router originates, and 0 for other paths.




| Update Source | Network | Weight (Added by Houston) |
|---|---|---|
| AS 100 | 172.16.10.0 | 2000 |
| AS 200 | 172.16.10.0 | 1000 |

Houston(config)**#router bgp 300**
Houston(config-router)**#neighbor 192.168.7.1 remote-as 100**
Houston(config-router)**#neighbor 192.168.7.1 weight 2000**
Houston(config-router)**#neighbor 192.168.219.1 remote-as 200**
Houston(config-router)**#neighbor 192.168.219.1 weight 1000**

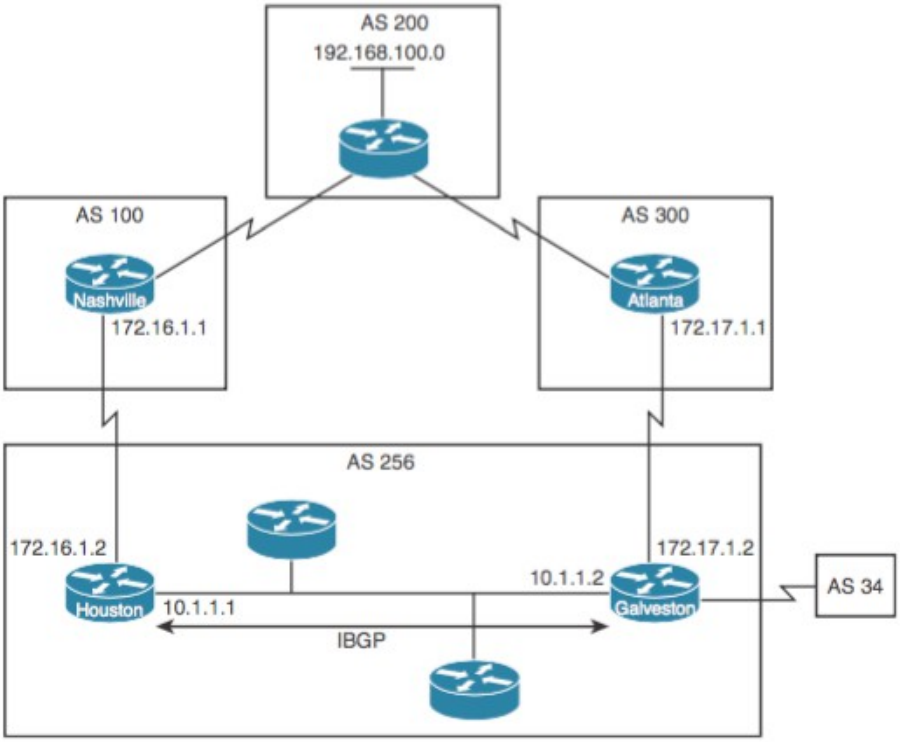
Has Houston forward traffic to the 172.16.10.0 network through AS 100, because the routes entering AS 300 from AS 100 have a higher weight attribute set compared to that same routes advertised from AS 200.
When a router receives a BGP Update, that router can set the Weight either selectively, per route, using a route map, or for all routes learned from a single neighbor

**Local Preference Attribute**

- use to force BGP routers to prefer one exit point over another for leaving the AS
- value can be between 0 and 429,496,729. Higher is preferred, default is 100
- It is local to the AS; it is exchanged between iBGP peers - ***not*** *advertised to eBGP peers*.
- Recall also that BGP does not allow a router to advertise iBGP-learned routes to iBGP peers

Houston(config)#router bgp 256
Houston(config-router)#neighbor 172.16.1.1 remote-as 100
Houston(config-router)#neighbor 10.1.1.2 remote-as 256
Houston(config-router)#bgp default local-preference 150
Galveston(config)#router bgp 256
Galveston(config-router)#neighbor 172.17.1.1 remote-as 300
Galveston(config-router)#neighbor 10.1.1.1 remote-as 256
Galveston(config-router)#bgp default local-preference 200

Based on these two configurations, traffic destined for a remote network that can be reached through autonomous system 256 will be routed through Galveston.




**Using AS_PATH Access Lists with Route Maps to Manipulate the Local Preference Attribute**
Route maps provide more flexibility than the **bgp default local-preference** router configuration command.
Galveston(config)**#router bgp 256**
Galveston(config-router)**#neighbor 172.17.1.1 remote-as 300**
Galveston(config-router)**#neighbor 172.17.1.1 route-map SETLOCAL in**
- reference route map SETLOCAL.

Galveston(config-router)**#neighbor 10.1.1.1 remote-as 256**
Galveston(config)**#ip as-path access-list 7 permit ^300$**
- BGP ACL 7 says permit updates whose AS_PATH starts and ends with 300 (^ and $)

Galveston(config)**#route-map SETLOCAL permit 10**
- creates route map SETWEIGHT, sequence number of 10

Galveston(config-route-map)**#match as-path 7**
- condition when policy routing is allowed- match BGP ACL 7.

Galveston(config-route-map)**#set local-preference 200**
- assigns local pref of 200 to any update coming from autonomous system 300

Galveston(config-route- map)**#route-map SETLOCAL permit 20**
- close route map SETLOCAL with an allow to accept all other routes.

In this example, using the **route-map** command, only updates received from AS 300, as specified in the **ip as_path access-list** command, will have a local preference set to 200.
Through the **neighbor route-map** command; **in** option is required for updates from an eBGP peer

**AS_PATH Attribute Prepending**

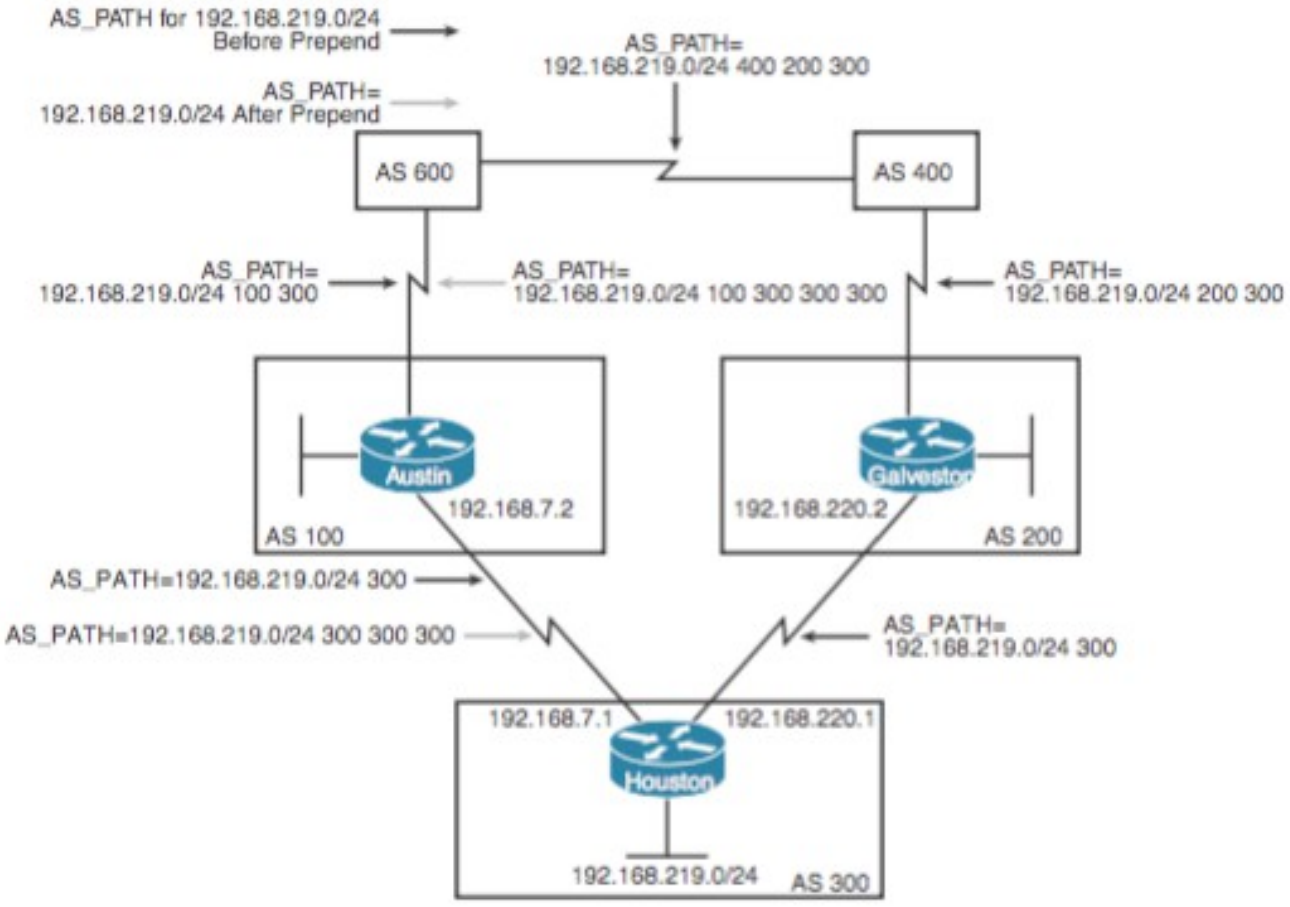
AS paths can be manipulated by prepending, or adding, extra ASNs to the AS_PATH. By this attribute's criteria, the shortest AS_PATH attribute is preferred; making it longer advertises it as a less viable route.

Houston(config)**#router bgp 300**
Houston(config-router)**#network 192.168.219.0**
Houston(config-router)**#neighbor 192.168.220.2 remote-as 200**
Houston(config-router)**#neighbor 192.168.7.2 remote-as 100**
Houston(config-router)**#neighbor 192.168.7.2 route-map SETPATH out**
- all routes going out to 192.168.7.2 are filtered with SETPATH route map

Houston(config)**#route-map SETPATH permit 10**
- create route map SETPATH; sequence number of 10

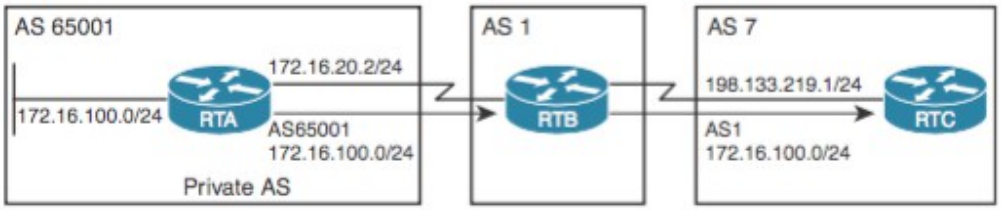
Houston(config-route-map)**#set as-path prepend 300 300**
- add 300 twice to the AS_PATH before sending to192.168.7.2

In this scenario, you want to use the configuration of Houston to influence the choice of paths in AS 600. Currently, the routers in AS 600 have reachability information to the 192.168.219.0/24 network via two routes: via AS 100 with an AS_PATH attribute of (100, 300), and via AS 400 with an AS_PATH attribute of (400, 200, 300). Assuming that the values of all other attributes are the same, the routers in AS 600 will pick the shortest AS_PATH attribute: the route through AS 100. Thus, prepend, or add, extra AS numbers to the AS_PATH attribute for routes that Houston advertises to AS 100 to have AS 600 select AS 400 as the preferred path of reaching the 192.168.219.0/24 network.




The result of this configuration is that the AS_PATH attribute of updates for network 192.168.219.0 that AS 600 receives via AS 100 will be (100, 300, 300, 300), which is longer than the value of the AS_PATH attribute of updates for network 192.168.219.0 that AS 600 receives via AS 400 (400, 200, 300).
AS 600 will choose AS 400 (400, 200, 300) as the better path.

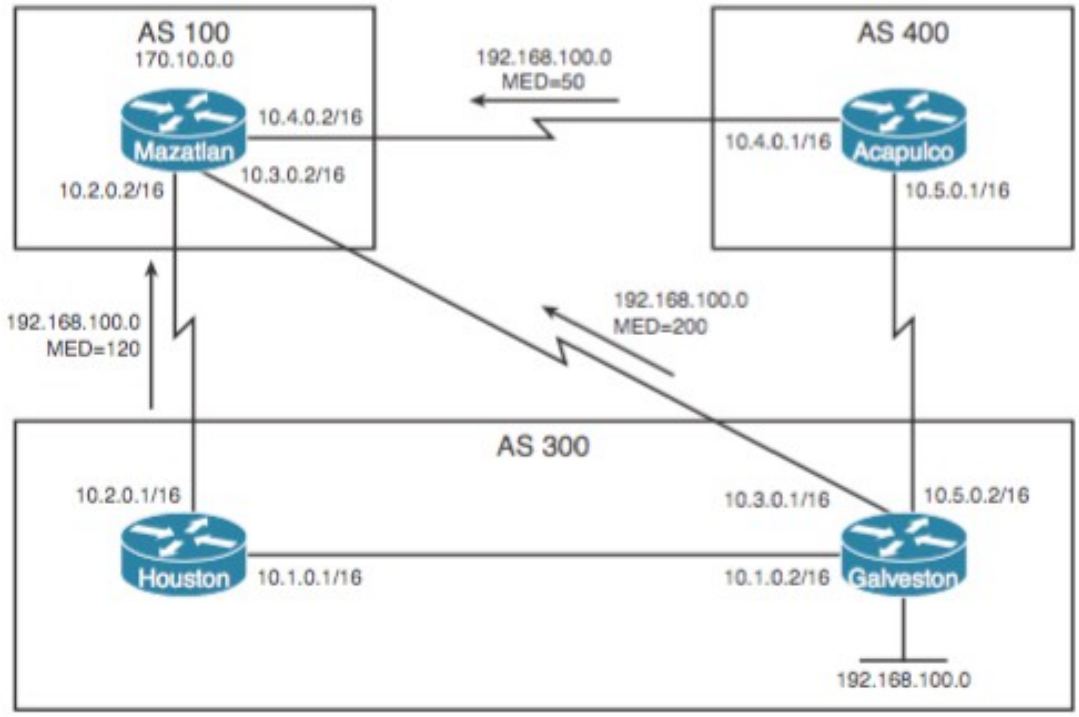
**AS_PATH: Removing Private Autonomous Systems**
Private ASNs (64,512 to 65,535) cannot be passed on to the Internet because they are not unique. Use **remove-private-as**, to strip private ASNs out of the AS_PATH list before advertised




RTB(config)**#router bgp 1**
RTB(config-router)**#neighbor 172.16.20.2 remote-as 65001**
RTB(config-router)**#neighbor 198.133.219.1 remote-as 7**
RTB(config-router)**#neighbor 198.133.219.1 remove-private-as**

**<u>MED Attribute</u> (Multi-Exit Discriminator)**
Also called ***the BGP metric***, indicates the preferred path is into an AS between two eBGP neighbors.
- AS to AS specific value describing that hop only- is not later handed to the next peer.
**- A lower value is preferred over a higher value.**
- The default value is 0. Change the default using the **default-metric**
- By default, only for paths from neighbors in the same AS. Override with **bgp always-compare-med**

Below, the routes for AS 300 to choose from are from Houston and Galveston. This is where MED is used by default (what it is intended for), so it won't be looked at in the case of AS 400 (Acapulco) unless it is specifically told to




Example: influence Mazatlan router to choose Houston as the entry point for AS 300 to reach 192.168.100.0.

Mazatlan(config)**#router bgp 100**
Mazatlan(config-router)**#neighbor 10.2.0.1 remote-as 300**
Mazatlan(config-router)**#neighbor 10.3.0.1 remote-as 300**
Mazatlan(config-router)**#neighbor 10.4.0.1 remote-as 400**
Acapulco(config)**#router bgp 400**
Acapulco(config-router)**#neighbor 10.4.0.2 remote-as 100**
Acapulco(config-router)**#neighbor 10.4.0.2 route-map SETMEDOUT out**
- reference route map SETMEDOUT.
Acapulco(config-router)**#neighbor 10.5.0.2 remote-as 300**
Acapulco(config)**#route-map SETMEDOUT permit 10**
- create route map SETMEDOUT sequence number of 10
Acapulco(config-route-map)**#set metric 50**
- set MED to 50

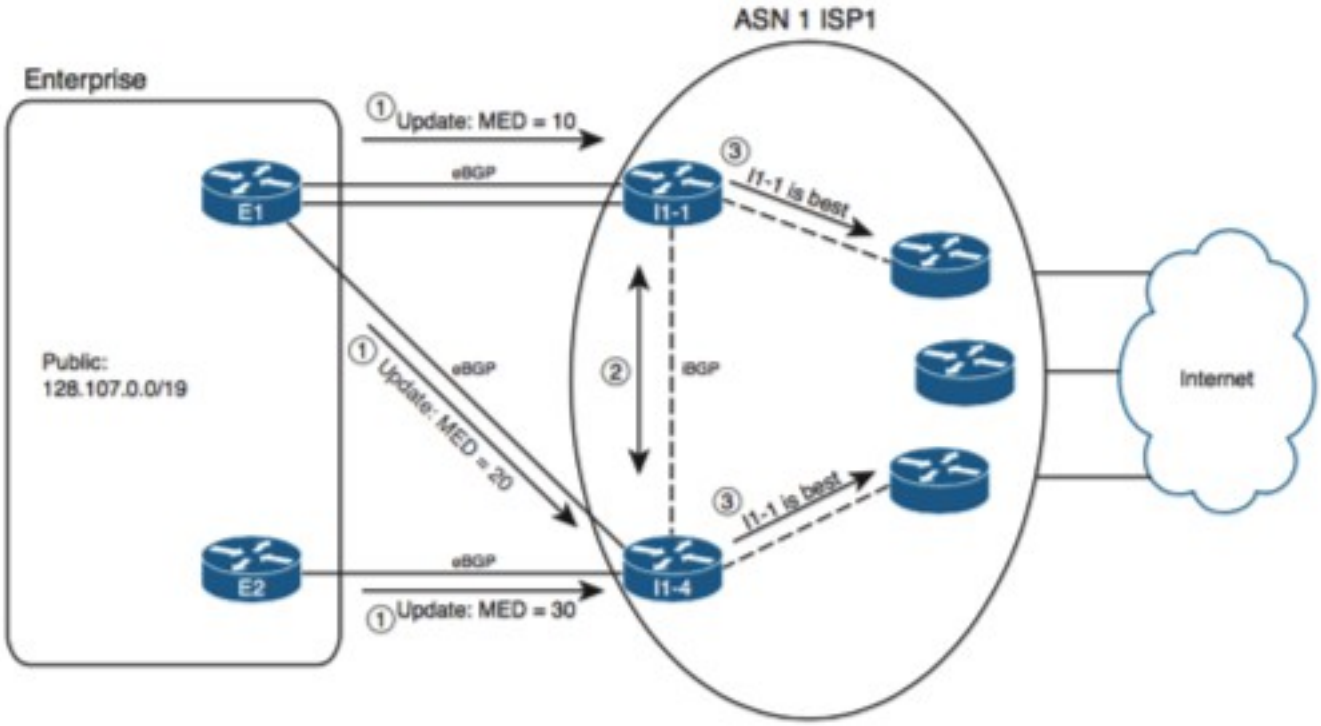
*Houston and Galveston have same setup with relative neighbor statements. Below are MEDs they set:*
Houston(config-router)**#neighbor 10.2.0.2 route-map SETMEDOUT out**
Houston(config-router)**#neighbor 10.1.0.2 remote-as 300**
Houston(config-route-map)**#set metric 120**
Galveston(config-router)**#neighbor 10.3.0.2 remote-as 100**
Galveston(config-router)**#neighbor 10.3.0.2 route-map SETMEDOUT out**
Galveston(config-route-map)**#set metric 200**

Mazatlan can only compare the MED attribute coming from Houston (120) to the MED attribute coming from Galveston (200) even though the update coming from Acapulco has the lowest MED value. Mazatlan will choose Houston as the best path for reaching network 192.168.100.0.

To force Mazatlan to include updates for network 192.168.100.0 from Acapulco in the comparison, use the **bgp always-compare-med** router configuration command on Mazatlan:
Mazatlan(config)**#router bgp 100**
Mazatlan(config-router)**#neighbor 10.2.0.1 remote-as 300**
Mazatlan(config-router)**#neighbor 10.3.0.1 remote-as 300**
Mazatlan(config-router)**#neighbor 10.4.0.1 remote-as 400**
Mazatlan(config-router)**#bgp always-compare-med**
Assuming that all other attributes are the same, Mazatlan will choose Acapulco as the best next hop for reaching network 192.168.100.0.

The most recent IETF decision about BGP MED assigns a value of infinity to a missing MED, making that route the least preferred. The Cisco IOS default is the opposite - treating routes without the MED attribute as having a MED of 0, making the route that is lacking the MED variable the **most** preferred.
- To configure the router to conform to the IETF standard, use **bgp bestpath missing-as-worst**.

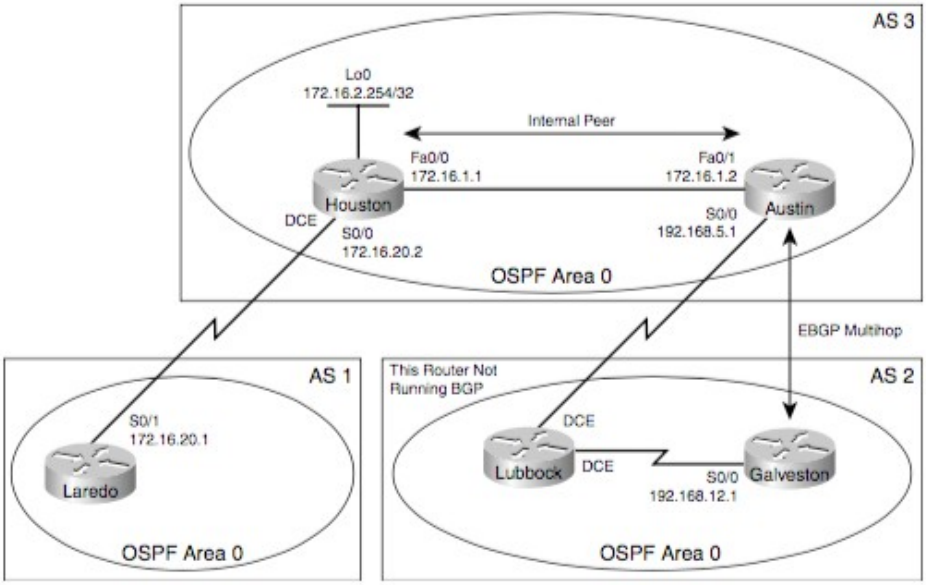
**MED with a dual-homed single ISP**
The enterprise can announce a MED that tells the ISP which path into the enterprise is best. As a result, the ISP can discriminate between the multiple exit points from that ISP to the enterprise. Those inbound routes into the enterprise from the ISP typically consist of either one, or a few, public IP address ranges. Say an enterprise engineer prefers the top BGP neighborship as the best path to use for inbound routes (MED 10), the middle link next (MED 20), and the bottom connection last (MED 30)




Step 1. E1 and E2 advertise 128.107.0.0/19, setting MED with an outbound route map, to various settings: MED 10 sent by E1 to I1-1, MED 20 sent by E1 to I1-4, and MED 30 sent by E2 to I1-4.
Step 2. I1-1 and I1-4 have an iBGP connection, so they learn each other's routes and agree as to which route wins based on MED.
Step 3. I1-1 and I1-4 also tell the other routers inside ISP1, causing all inbound traffic to funnel toward Router I1-1.
(Remember MED is advertised by one AS into another, propagated inside the AS, but not sent to another AS.

Note that Routers I1-1 and I1-4 in this example could have chosen a better route based on all the earlier best-path steps. However, a brief analysis of the steps tells us that unless someone makes an effort to override the effects of MED, these routers' best-path algorithms will use MED. Assuming that the enterprise and ISP agree to rely on MED, the earlier best-path steps should not matter.

## *BGP Configuration Example*




**Houston Router**

| | |
|---|---|
| Router(config)#hostname Houston | Sets the router name to Houston. |
| Houston(config)#interface loopback 0 | Moves to loopback interface mode. |
| Houston(config-if)#ip address 172.16.2.254 255.255.255.255 | Assigns an IP address and netmask. |
| Houston(config-if)#interface fastethernet 0/0 | Moves to interface configuration mode. |
| Houston(config-if)#ip address 172.16.1.1 255.255.255.0 | Assigns an IP address and netmask. |
| Houston(config-if)#no shutdown | Enables the interface. |
| Houston(config-if)#interface serial 0/0/0 | Moves to interface configuration mode. |
| Houston(config-if)#ip address 172.16.20.2 255.255.255.0 | Assigns an IP address and netmask. |
| Houston(config-if)#clock rate 56000 | Assigns the clock rate. |
| Houston(config-if)#no shutdown | Activates the interface. |
| Houston(config-if)#exit | Returns to global configuration mode. |
| Houston(config)#router ospf 1 | Starts the OSPF routing process. |
| Houston(config-router)#network 172.16.0.0 0.0.255.255 area 0 | Any interface with 172.16.*x*.*x* to be placed into OSPF area 0. |
| Houston(config-router)#exit | Returns to global configuration mode. |
| Houston(config)#router bgp 3 | Starts the BGP routing process. |
| Houston(config-router)#no synchronization | Turns off route synchronization. |
| Houston(config-router)#neighbor 172.16.1.2 remote-as 3 | Identifies a peer router at 172.16.1.2. |
| Houston(config-router)#neighbor 172.16.1.2 update-source loopback 0 | Use any interface for TCP connections, as long as Loopback0 is configured. |
| Houston(config-router)#neighbor 172.16.20.1 remote-as 1 | Identifies a peer router at 172.16.20.1. |
| Houston(config-router)#no auto- summary | Disables auto-summarization. |
| Houston(config-router)#exit | Returns to global configuration mode. |
| Houston(config)#exit | Returns to privileged mode. |
| Houston#copy running-config startup- config | Saves the configuration to NVRAM. |

**Laredo Router**

| | |
|---|---|
| Router(config)#hostname Laredo | Sets the router name to Laredo. |
| Laredo(config)#interface serial 0/0/1 | Moves to interface configuration mode. |
| Laredo(config-if)#ip address 172.16.20.1 255.255.255.0 | Assigns an IP address and netmask. |
| Laredo(config-if)#no shutdown | Activates the interface. |
| Laredo(config-if)#exit | Returns to global configuration mode. |
| Laredo(config)#router bgp 1 | Starts the BGP routing process. |
| Laredo(config-router)#no synchronization | Turns off route synchronization. |
| Laredo(config-router)#neighbor 172.16.20.2 remote-as 3 | Identifies a peer router at 172.16.20.2. |

| | |
|---|---|
| Laredo(config-router)#no auto-summary | Disables auto-summarization. |
| Laredo(config-router)#exit | Returns to global configuration mode. |
| Laredo(config)#exit | Returns to privileged mode. |
| Laredo#copy running-config startup- config | Saves the configuration to NVRAM. |

**Galveston Router**

| | |
|---|---|
| Router(config)#hostname Galveston | Sets router name to Galveston. |
| Galveston(config)#interface serial 0/0/0 | Moves to interface configuration mode. |
| Galveston(config-if)#ip address 192.168.12.1 255.255.255.0 | Assigns an IP address and netmask. |
| Galveston(config-if)#no shutdown | Activates the interface. |
| Galveston(config-if)#exit | Returns to global configuration mode. |
| Galveston(config)#router ospf 1 | Starts the OSPF routing process. |
| Galveston(config-router)#network 192.168.12.0 0.0.0.255 area 0 | Any interface with 192.168.12.x to be placed into OSPF Area 0. |
| Galveston(config-router)#exit | Returns to global configuration mode. |
| Galveston(config)#router bgp 2 | Starts the BGP routing process. |
| Galveston(config-router)#neighbor 192.168.5.1 remote-as 3 | Identifies a peer router at 192.168.5.1. |
| Galveston(config-router)#neighbor 192.168.5.1 ebgp-multihop 2 | Two routers not directly connected need an eBGP session. |
| Galveston(config-router)#no auto-summary | Disables auto-summarization. |
| Galveston(config-router)#exit | Returns to global configuration mode. |
| Galveston(config)#exit | Returns to privileged mode. |
| Galveston#copy running-config startup- config | Saves the configuration to NVRAM. |

**Austin Router**

| | |
|---|---|
| Router(config)#hostname Austin | Sets the router name to Austin. |
| Austin(config)#interface serial 0/0/0 | Moves to interface configuration mode. |
| Austin(config-if)#ip address 192.168.5.1 255.255.255.0 | Assigns an IP address and netmask. |
| Austin(config-if)#no shutdown | Activates the interface. |
| Austin(config-if)#interface fastethernet 0/1 | Moves to interface configuration mode. |
| Austin (config-if)#ip address 172.16.1.2 255.255.255.0 | Assigns an IP address and netmask. |
| Austin(config-if)#no shutdown | Activates the interface. |
| Austin(config-if)#exit | Returns to global configuration mode. |
| Austin(config)#router ospf 1 | Starts the OSPF routing process. |
| Austin(config-router)#network 172.16.0.0 0.0.255.255 area 0 | Any interface with 172.16.*x*.*x* to be placed into OSPF area 0. |
| Austin(config-router)#network 192.168.5.0 0.0.0.255 area 0 | Any interface with 192.168.5.*x* to be placed into OSPF area 0. |
| Austin(config-router)#exit | Returns to global configuration mode. |
| Austin(config)#router bgp 3 | Starts the BGP routing process. |
| Austin(config-router)#no synchronization | Turns off route synchronization. |
| Austin(config-router)#neighbor 172.16.2.254 remote-as 3 | Identifies a peer router at 172.16.2.254. |
| Austin(config-router)#neighbor 192.168.12.1 remote-as 2 | Identifies a peer router at 192.168.12.1. |
| Austin(config-router)#neighbor 192.168.12.1 ebgp-multihop 2 | Two routers not directly connected need eBGP session. |
| Austin(config-router)#no auto-summary | Turns off auto-summarization. |
| Austin(config-router)#exit | Returns to global configuration mode. |
| Austin(config)#exit | Returns to privileged mode. |
| Austin#copy running-config startup- config | Saves the configuration to NVRAM. |

# *Syntax for BGP Filtering Methods*

**Traditional access lists not great for granular BGP filters**
The old-school IOS ACLs we know might be seem like a good way to match traffic by prefix/subnet, but it can get to be a mess. Cisco IOS 12.0 came up with a better way with **prefix-list**, but it helps to know why this was done.

Here is an easy scenario: declare BGP neighbors, make a simple access list to match one subnet, and apply it with **distribute-list** to the updates going **out**, so we can say "don't send routing updates from that **access-list 1** to this neighbor". This works fine.

R1(config)# **router bgp 3**
R1(config-router)# **neighbor 172.16.1.2 remote-as 3**
R1(config-router)# **neighbor 172.16.20.1 remote-as 1**
R1(config-router)# **neighbor 172.16.20.1 distribute-list 1 out**
R1(config-router)# **exit**
R1(config)# **access-list 1 deny 192.168.10.0 0.0.0.255**
R1(config)# **access-list 1 permit any**

Problems come up when advertising the aggregate address of 172.16.0.0/16 but not the individual subnet- you need an extended ACL. The way those work with BGP route filters, the ACL will first match the network address and *then* match the subnet mask of the prefix. To do this, both network and netmask are paired with their own wildcard bitmask and it looks like this confusing, ugly mess:

**access-list 101 permit ip 172.16.0.0 0.0.255.255 255.255.0.0 0.0.0.0**

That is why IOS 12.0 gave us the **prefix-list** option.

**Prefix-List Syntax**
This example limits updates sent to the peer to only be routing info for this specific matching subnet:
R1(config)# **ip prefix-list UPDATE172 permit 172.16.0.0/16**
R1(config)# **router bgp 100**
R1(config-router)# **neighbor 192.168.1.1 remote-as 200**
R1(config-router)# **neighbor 192.168.1.1 prefix-list UPDATE172 out**
Applies UPDATE172 to routing updates sent to 192.168.1.1. Permits update of 172.16.0.0/16, but has implicit deny at end of the list for others like 172.16.0.0/17 or 172.16.20/24

**ip prefix-list** list-name **[seq** seq-value**] deny | permit** network/cidr **[ge** ge-value**] [le** le-value**]**

| Parameter | Description |
|---|---|
| *list-name* | The name of the prefix list. |
| **seq** | (Optional) Applies a sequence number to the entry being created or deleted. |
| *seq-value* | (Optional) Specifies the sequence number. |
| **deny** | Denies access to matching conditions. |
| **permit** | Permits access for matching conditions. |
| *network/cidr* | (Mandatory) The network number and length (in bits) of the netmask. |
| **ge** | (Optional) Applies *ge-value* to the range specified. |
| *ge-value* | (Optional) Specifies the beginning or "from" value of a range |
| **le** | (Optional) Applies *le-value* to the range specified. |
| *le-value* | (Optional) Specifies the end or "to" value of a range |

- **-** There is an **implicit deny** statement at the end of each prefix list.
- The range of sequence numbers that can be entered is from 1 to 4,294,967,294.
- A router tests for prefix list matches from the lowest sequence number to the highest.
- By numbering your **prefix-list** statements, you can add new entries at any point in the list.
- If no seq # is given, default is applied: 5 is applied to the first, next unnumbered entries are incremented by 5.

**ip prefix-list NY_ROUTES permit 192.0.0.0/8 le 24**
Permit routes with a netmask of up to 24 bits in 192.0.0.0/8 - No seq number given- gets the default 5
**ip prefix-list NY_ROUTES deny 192.0.0.0/8 ge 25**
Deny routes with netmask of 25 bits or greater in 192.0.0.0/8 - No seq num given- 10 is applied (+5)
**ip prefix-list RENO permit 10.0.0.0/8 ge 16 le 24**
Permit routes 10.0.0.0/8 with a netmask between 16 to 24 bits - No seq number given- gets the default 5
**ip prefix-list HOUSTON seq 3 deny 0.0.0.0/0**
Assigns a sequence number of 3 to this statement.
**no ip prefix-list TORONTO seq 10**
Removes sequence number 10 from the TORONTO list.

**AS_PATH Access Lists**

The AS_PATH attribute can be searched with regular expressions to match routing updates, and slap with an access-list label to filter or modify. Most of these should look familiar. They can also be tested with **show ip bgp**

| | |
|---|---|
| ^ | Matches the beginning of the input string. |
| $ | Matches the end of the input string. |
| ^$ | Matches an empty string in the AS_PATH field (the local device's AS is blank, so this matches that) |
| _ | Matches a space, comma, left brace, right brace, beginning or end of an input string |
| . | Matches any single character. |
| * | Matches 0 or more single- or multiple-character patterns. |

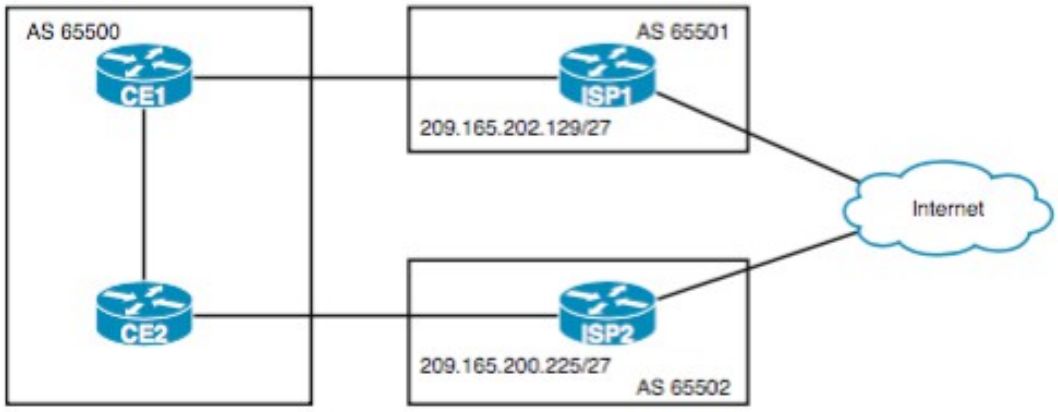
To find all subnets reachable via autonomous system 65002 (AS_PATH begins with 65002):

R1**#show ip bgp regexp ^65002_**

| Network | Next Hop | Metric | LocPrf | Weight | Path |
|---|---|---|---|---|---|
| *>i172.16.0.0 | 192.168.28.1 | 100 | 0 | | 65002 65003 i |
| * i172.24.0.0 | 192.168.28.1 | 100 | 0 | | 65002 65003 65004 65005 i |

| | |
|---|---|
| show ip bgp regexp _65004$ | ... originating from autonomous system 65004 (AS_PATH ends with 65004): |
| show ip bgp regexp ^65002_ | ... reachable via autonomous system 65002 (AS_PATH begins with 65002): |
| show ip bgp regexp _65005_ | ... transiting through autonomous system 65005 (AS_PATH contains 65005): |
| show ip bgp regexp 2150 | Will match 2150, 12150 or 21507. This is where "_" helps |
| show ip bgp ^$ | Originates from THIS autonomous system (AS_PATH is blank) |

This simple example demonstrates using prefix lists and AS_PATH access-lists together. Notice how the AS_PATH list is referred to with the **filter-list** keyword. The first job is to allow CE1 and CE2 to only learn ISP routes with a mask greater than / 15 (ge 16) and less than /25 (le 24). The second job is to ensure that AS 65000 does not become a transit autonomous system for ISP1 to reach ISP2 (and vice versa).




**CE1 Configuration**

| | |
|---|---|
| **ip prefix-list ISP1 permit 0.0.0.0 ge 16 le 24** | Prefix list which only permits routes with a mask between 16 and 24 |
| **ip as-path access- list 1 permit ^$** | Creates AS_PATH ACL matching routes that only originate from within autonomous system 65500 |
| **router bgp 65000** | |
| **neighbor 209.165.202.129 prefix-list ISP1 in** | Assigns ISP1 prefix list to neighbor 209.165.202.129 (ISP1) for all routes learned from that neighbor |
| **neighbor 209.165.202.129 filter-list 1 out** | Assigns AS_PATH ACL to neighbor 209.165.202.129 (ISP1) for all routes sent to that neighbor |

**CE2 Configuration**

| | |
|---|---|
| **ip prefix-list ISP2 permit 0.0.0.0 ge 16 le 24** | Creates a prefix list that only permits routes with a mask between 16 and 24 |
| **ip as-path access- list 1 permit ^$** | Creates an AS_PATH access list matching routes that only originate from within AS 65500 |
| **router bgp 65000** | |
| **neighbor 209.165.200.225 prefix-list ISP2 in** | Assigns ISP2 prefix list to neighbor 209.165.200.225 (ISP2) for all routes learnt from that neighbor |
| **neighbor 209.165.200.225 filter-list 1 out** | Assigns AS_PATH ACL to neighbor 209.165.200.225 (ISP2) for all routes sent to that neighbor |

**Route Maps**
This example uses a prefix-list to catch updates matching 172.16.10.0/24 and label with "AS400_ROUTES". Then a route-map is created so we can basically make a conditional statement that says, if it sees those, make a change to the weight attribute. If it doesn't match and falls through to match the second rule, set a different weight. This same mechanism can be used to make other changes, or specify what updates to use or drop.

Finally, at the end make the BGP process, add the neighbors and directs to check the route-map for the rules. Adding the keyword "in" refers to routing updates from 192.168.7.1, and the keyword "out" is used for the routing updates we send out 192.168.7.1

Houston(config)**#ip prefix-list AS400_ROUTES permit 172.16.10.0/24**
Creates a prefix list that matches the 172.16.10.0/24 network belonging to AS 400.
Houston(config)**#route-map SETWEIGHT permit 10**
Creates a route map called SETWEIGHT. A sequence number of 10 is assigned.
Houston(config-route-map)**#match ip address prefix-list AS400_ ROUTES**
Specifies the condition under which policy routing is allowed, matching the AS400_ROUTES prefix list.
Houston(config-route-map)**#set weight 200**
Assigns a weight of 200 to any route update that meets the condition of prefix list AS400_ROUTES.
Houston(config-route-map)**#route- map SETWEIGHT permit 20**
Creates a second statement for the route map. A sequence number of 20 is assigned.
Houston(config-route-map)**#set weight 100**
Assign weight of 100 to all route updates/networks learned from outside (didn't match) AS400_ROUTES
Houston(config-route-map)**#exit**
Returns to global configuration mode.
Houston(config)**#router bgp 300**
Starts the BGP routing process.
Houston(config-router)**#neighbor 192.168.7.1 route-map SETWEIGHT in**
Uses the route map SETWEIGHT to filter all routes learned from neighbor 192.168.7.1.

Route-map can employ regular and extended ACLs, prefix-lists, and AS_PATH access-lists.

| BGP subcommand | Commands to create | What can be matched |
|---|---|---|
| neighbor distribute-list (standard ACL) | access-list, ip access-list | Prefix, with WC mask |
| neighbor distribute-list (extended ACL) | access-list, ip access-list | Prefix and prefix length, with WC mask for each |
| neighbor prefix-list | ip prefix-list | Exact or "first N" bits of prefix, plus range of prefix lengths |
| neighbor filter-list | ip as-path access-list | AS_Path contents; all NLRI whose AS_Paths are matched considered to be a match |
| neighbor route-map | route-map | Prefix, prefix length, AS_ Path, and/or any other PA matchable within a BGP route map |

**BGP Communities**
BGP communities are used to group networking devices that share common properties, regardless of network, autonomous system, or any physical boundaries. In large networks applying a common routing policy through prefix lists or access lists requires individual peer statements on each networking device. Using the BGP community attribute BGP neighbors, with common routing policies, can implement inbound or outbound route filters based on the community tag rather than consult large lists of individual permit or deny statements. Consider a community a group of prefixes that should be treated the same way and an alternative to matching numerous prefixes using an access-list or prefix-list"

Communities are created with **set community** as below. The aa:nn identifier is usually the AS followed by an arbitrary identifier (it would look something like 100:250) several can be entered on the same line. The options in brackets are for built in communities with specific rules built in.

**set community aa : nn [ additive | local-as | no-advertise | no-export ]**
additive - Adds to existing community (one or more keywords)
local-as (aka "well-known community) advertised to only peers in local AS or within a sub-AS of a confederation.
no-advertise - these are not advertised to any peer (internal or external).
no-export routes are like local-as but more strictly not advertised to external peers.

When declaring neighbors, you can share communities with it with **neighbor x.x.x.x send-community**. Here are some snippets demonstrating the use of communities:

```
ip bgp-community new-format
access-list 101 permit ip host 6.6.6.0 host 255.255.255.0
route-map Peer-R1 permit 10
 match ip address 101
 set community 100:300 109:02 33:40

ip community-list 1 permit 100:300
route-map Peer-R3 permit 10
 match community 1
 set local-preference 130
```

ip community-list -- Creates a community list for BGP and control access to it.
match community -- Matches a BGP community.
set comm-list delete -- Removes communities from the community attribute of an inbound or outbound update.
show ip bgp community -- Displays routes that belong to specified BGP communities.

**BGP Confederations**
A BGP confederation divides our AS into sub-ASes to reduce the number of required IBGP peerings. Within a sub-AS we still require full-mesh IBGP but between these sub-ASes we use something that looks like EBGP but behaves like IBGP (called confederation BGP)

```
R2(config)#router bgp 24
R2(config-router)#bgp confederation identifier 2
R2(config-router)#bgp confederation peers 35
R2(config-router)#neighbor 4.4.4.4 remote-as 24
R2(config-router)#neighbor 4.4.4.4 update-source loopback 0
R2(config-router)#neighbor 3.3.3.3 remote-as 35
R2(config-router)#neighbor 3.3.3.3 update-source loopback 0
R2(config-router)#neighbor 3.3.3.3 ebgp-multihop 2
```

When you start the BGP process you have to use the AS number of the sub-AS. Secondly, you have to use the **bgp confederation identifier** command to tell BGP what the main AS number is.
We also have to configure all other sub-AS numbers with the **bgp confederation peers** command, in this case that's only AS 35. R4 is in the same sub-as so you can configure this neighbor just like any other IBGP neighbor. R3 is a bit different though…since it's in another sub-AS we have to use the same rules as EBGP, that means configuring multihop if you are using loopbacks.

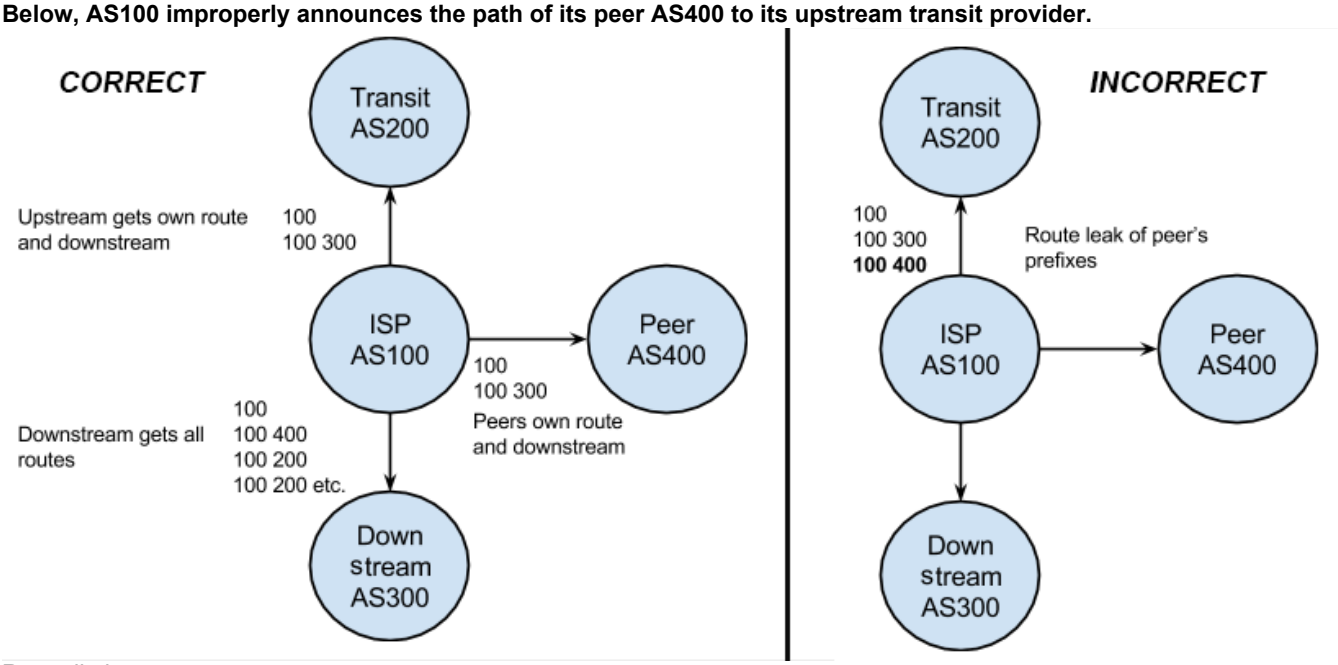
When you look up sh ip bgp x.x.x.x the route is tagged with either **confed-internal** (which means that it came from an IBGP router within the same sub-AS) or **confed-external.** With confed-external the sub-AS number simply appears in parenthese above the given route info. BGP confederations use an attribute called AS_CONFED_SET. This "confederation set" prepends the list with the sub-AS's. Generally speaking, when routes are sent to another AS, all the sub-AS numbers are removed

```
R5#show ip bgp 11.11.11.11
BGP routing table entry for 11.11.11.11/32, version 6
Paths: (2 available, best #2, table Default-IP-Routing-Table)
  Advertised to update-groups:
     2
  (24) 1
    192.168.12.1 (metric 3) from 4.4.4.4 (4.4.4.4)
      Origin IGP, metric 0, localpref 100, valid, confed-external
  (24) 1
    192.168.12.1 (metric 3) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 0, localpref 100, valid, confed-internal, best
```

# *BGP Route Leaks and BGP Hijacking*

Route leaks and hijacks are where illegitimate prefixes/address blocks are wrongly propagated
- illegitimate advertisement of prefixes, blocks of IP addresses
- propagate across networks and lead to incorrect or suboptimal routing.
- Can happen from an AS originating a prefix that it does not actually own
- Also happens when AS announces it can deliver traffic through a route that should not exist
- Are particularly prone to propagation when
    - a more specific prefix is advertised (BGP prefers the most specific block of addresses)
    - a path is advertised that is shorter than the currently available paths (BGP prefers shortest AS Path).
- Usually happen when BGP advertisements are not properly filtered using the no-export community.

**Below, AS100 improperly announces the path of its peer AS400 to its upstream transit provider.**




Remediation
- Maintain communication and work with administrators of neighboring AS'es to collaborate on fixes
- BGP-advertise routes more preferable than the leaked route (a more specific prefix length and/or shortest path)
- For example, if the problematic route advertises a /17, counter with advertising an /18
- Advertising a shorter path is not as effective
- Last resort: consider changing your prefixes entirely by changing DNS records. Only feasible if alternate locations are available (i.e., other data center). The TTL on DNS records also becomes an issue.

Preventative measures
- publish Route Origin Authorizations (ROAs) in the various regional Internet registries (RIRs)
- verifies that a given origin AS is authorized to announce its prefixes, (incl. max prefix lengths)
- networks using Resource Certification (RPKI) can validate the origin AS and verify routes legitimacy

Best practices: Proper route filtering uses a set of robust filtering rules, likely including:
- Filter out Bogon prefixes and routes with Bogon ASNs anywhere in the AS path. Reserved or unallocated IP spaces should never be advertised.
- Filter out routes with more than two Tier 1 (“transit-free”) networks in the AS path. When more are present, at least one of the networks is providing transit to another.
- If you don’t sell transit to large networks (like Tier 1 networks), filter out routes from them in the AS path. Use with a whitelist of prefixes that each of your customers may announce to you.
- Use peer locking- ask peers for all possible upstream networks, and only allow those as intermediate networks (See Snijders NANOG67 - https://www.nanog.org/sites/default/files/Snijders_Everyday_Practical_Bgp.pdf
- Use BGP Maximum-Prefix to set the maximum number of prefixes that can be announced from your peers. This helps during an episode of rapid propagation of problematic AS'es.

2008- Pakistan's ISPs hijacked YouTube’s routes with a more specific prefix pointing to a null interface, The covered prefix was mistakenly leaked to the rest of the Internet, blocking YouTube for everyone else too. The practice of null routing commonly appears as a part of censorship efforts, including the Great Firewall of China (https://blog.thousandeyes.com/deconstructing-great-firewall-china/).

http://dyn.com/blog/mitm-internet-hijacking/
Bigger real-world examples: https://blog.thousandeyes.com/finding-and-diagnosing-bgp-route-leaks/

# ***Setting up GRE with IPSEC for a Typical VPN***

I made this from a transcription I made of Doug Suida's video at:
https://www.youtube.com/watch?v=2PtK8HgkRvM

A followup that uses the same technique to set up DMVPN (Dynamic Multipoint VPN) is here:
https://www.youtube.com/watch?v=WEzo1UvMpg0 - My summarization is on the next pages.

On network mapping it looks like these aren't directly connected but "sh cdp nei only shows f/0/0 - doesn't show as a directly connected neighbor. Neighborship won't happen until you add to OSPF network 10.10.1.0 0.0.0.3 area 0

**Outbound connection:**
R1 ==> 162.27.193.128/30 ==>========INTERNET========<== 45.12.153.200/30 <== R2
**Tunnel inside**
10.10.1.1 ==>Tunnel0 << ------> ====== 10.10.1.0/30 ====== < ------- >> Tunnel0 <== 10.10.1.2

**Setting up the LANs, uplinks, and tunnel**

Local: 192.168.1.0/24
```
R1> int Loopback0
R1> ip address 192.168.1.1
R1> int f0/1
R1> ip address 192.168.1.2
R1> router ospf 123
R1> network 192.168.1.0 0.0.0.255 area 0
```
Uplink
```
R1> int f0/0
R1> ip address 162.27.193.130
R1> exit
```
Tunnel
```
R1> int tunnel0
R1> tunnel source f0/0
R1> tunnel destination 45.12.153.202
R1> ip address 10.10.1.1 255.255.255.252
```

Local: 10.1.1.0/24
```
R2> int Loopback0
R2> ip address 10.1.1.1
R2> int f0/1
R2> ip address 10.1.1.2
R2> router ospf 123
R2> network 10.1.1.0 0.0.0.255 area 0
```
Uplink
```
R2> int f0/0
R2> ip address 45.12.153.202
```
Tunnel
```
R2> int tunnel0
R2> tunnel source f0/0
R2> tunnel destination 162.27.193.130
R2> ip address 10.10.1.2 255.255.255.252
R2>
```

These are also added on both routers but not needed
> tunnel path-mtu-discovery
> ip ospf mtu-ignore

The first one gets the tunnel to do it's own MTU discovery since it is over the internet
The second one is to do the same with OSPF to make sure it stays neat
You can put in a keepalive on the tunnel but don't need it since it's a logical interface
If you are skiddish about OSPF go ahead.
On OSPF: no tunnel = no multicast and no neighborship

So now, after testing and making sure it's all working, we can add standard VPN mechanisms with IPSEC.
You might otherwise have many policies and transform sets but here just one.

1. Define traffic to be encrypted

```
R1> ip access-list extended MY-IPSEC-TRAFFIC
R1> remark VPN Traffic
R1> permit gre host 162.27.193.130 host 45.12.153.202
```

You'll do the same on R2 with the address reversed

2. Set up Phase I, with a preshared key rather than a certificate

```
R1> crypto isakmp policy 1
R1> authentication pre-share
####### that is where you'd put cert-based rsa-encr, rsa-sig
R1> encryption aes 128
####### other options 3des des aes and 128, 192, 256
R1> hash sha
####### you can also choose to put md5
R1> group 2
####### choose DH group 1, 2 or 5
R1> lifetime 86400
####### 60-86400 seconds (86400 sec = 1 day, is the default)
R1> crypto isakmp key 0 mypassword address 45.12.153.202
####### If the password you are giving here is encrypted or not:  0 is plaintext, 6 is encrypt
```

3. Set up Phase II - make the IPSEC transform set

```
R1> crypto ipsec transform-set TRANSFM-TUN esp-aes 128 esp-sha-hmac
####### There are tons of transforms to use.  The above are the defaults.
R1> mode tunnel
####### Or transport
```

4. Create crypto-map

```
R1>  crypto map CRYPTOMAPPY 1 ipsec-isakmp
R1(config-cryptomap)> description DESC to R2
R1(config-cryptomap)> match address MY-IPSEC-TRAFFIC
####### Name of the ACL made in step 1
```

5. Apply to interfaces

```
R1> set peer 45.12.153.202
R1> set transform-set TRANSFM-TUN

R1> int f0/0
R1> crypto map CRYPTOMAPPY
R1> int tunnel 0
R1> crypto map CRYPTOMAPPY
```

Test

```
R1> show crypto ipsec sa
```

Show stats of # crypted/ decrypted - tells us if it really being encrypted
If you do a sh run config you won't see a lot of the stuff that were used here which were default settings

Valid Encryption Methods
esp-des
esp-3des (default)
esp-aes (128-bit encryption)
esp-aes-192
esp-aes-256
esp-null

Valid Authentication Methods
esp-md5-hmac
esp-sha-hmac (default)

## ***DMVPN - Dynamic Multipoint VPN with mGRE (multipoint GRE), IPSec and NHRP***

5 sites (routers), full mesh
R1 has individual tunnels to R2, R3, R4, and R5, and each of those routers have similar tunnels to eachother.
10 tunnels? DMVPN can automate the process. Sort of a hub and spoke configuration mashed in with a frame relay-style network with one hub site (not full mesh FR). With DMVPN, it turns out you don't have to configure all the tunnels after doing those coming out of R1. Those tunnels among the "spokes" routers are built dynamically as-needed by the DMVPN configuration.

If some host off of R4 needs something off of the network hosted by R2, it can dynamically create the tunnel so it can be used, wait for it to time out, and then tear it down again. Say R4 queries the hub router (R1), which provides the next hop information it needs to build the tunnel; it does so, and when they finish tear the tunnel down. In this scenario you don't need anything but interior routing protocols
On routing, lets say R3 adds a network and sends out an advertising update. The update only goes to the hub router- not the others, and it's the hub router that sends the update to the others. Sortof like a NBMA setup, and have to turn off split horizon, add Next Hop Resolution Protocol (NHRP), etc. Also- participant routers are never neighbors with any other router than the "hub" router, even when a tunnel is created. R1 is depended on for information needed to build the tunnels and propagate the routing updates, and no other participating router can communicate with another without this "hub" router as it's resource.

Here we set up R1, then just R2 and R3 - it will become apparent after one spoke is done, that it's a replication.
After setting up the "hub router" and the first spoke router, the rest of the spoke routers can use the same stuff with only the tunnel and gateway IPs needing to be set, and EIGRP tuned on.

Overview: EIGRP 10 with:
R1 - f0/0 - 54.45.12.1 to default GW .2/30 - internal network (loopback): 192.168.17.0/24 DMVPN IP: 192.168.1.1
R2 - f0/0 - 54.45.12.5 to 6/30 - internal network (loopback): 192.168.200.0/24 DMVPN IP: 192.168.1.12
R3 - f0/0 - 54.45.12.9 to 10/30 - internal network (loopback): 10.1.1.0/24 DMVPN IP: 192.168.1.3
**Set up all of this before starting the exercise. Ping-test all the links and you are ready:**

Differences from previous: we create a transform, no cryptomap with ACL. Create ipsec profile instead
1. Set ISAKMP policy and define pre-shared key
R1> crypto isakmp policy 10
R1> authentication pre-share
R1> encryption aes 192
R1> hash md5
R1> group 2
R1> crypto isakmp key 0 ISAKEY address 0.0.0.0 0.0.0.0 <---said isakey was name - no password?
####### Here we don't have a specific IP address like the other example. Put ANY with matching key (with 0's)

2. Phase II - make the IPSEC transform set
R1> crypto ipsec transform-set DMVPN-TRANSFM esp-aes 192 esp-md5-hmac

3. Make a profile (instead of the cryptomap for single tunnel)
R1> ipsec profile DMVPN-PROFILE
R1(ipsec profile)> set security-association lifetime seconds 120
####### 120 sec is the minimum. Is lifetime before tunnel gets torn down!
R1(ipsec profile)> set transform-set DMVPN-TRANSFM
####### DONE WITH IPSEC

Tunnel
R1> int tunnel0
R1> ip address 192.168.1.1 255.255.255.0
R1> no ip redirects
####### MTU for encryption on top of packets
R1> ip mtu 1440

####### Next Hop Resolution Protocol (NHRP) stuff
R1> ip nhrp authentication ISAKEY
R1> ip nhrp network-id 1

####### Allow multicast protocols to use dynamic tunnels, shut off split horizon and next-hop-self
R1> ip nhrp map multicast dynamic
R1> no ip split-horizon eigrp 10
R1> no next-hop-self
R1> tunnel source fastEthernet 0/0
R1> tunnel mode gre multipoint
R1> tunnel key 0
R1> tunnel protect profile DMVPN-PROFILE
R!> router eigrp 10
R1> network 192.168.1.0

***So it should be understood at this point:***
**R2 and R3 both use tunnel0 to get to R1; tunnel0 goes from R2 to R3. It is the SAME tunnel for ALL connected routers and that is what multipoint GRE provides to us. Very much like FR- here one tunnel for the whole mess instead of 5, or 10 tunnels.**

R2 - needs the same IPSEC config we just did. On R1, do a sh running config and copy ipsec material. Just copy and paste it into the R2 command line after config t is entered. Done.
Like the IPSEC config, the tunnel config is almost the same... don't copy and paste this quite yet- this is the configuration that is going to be copied, pasted into all the other spoke routers.

R2> int tunnel0
R2> ip address 192.168.1.2 255.255.255.0
R2> no ip redirects
R2> ip mtu 1440
R2> ip nhrp authentication ISAKEY
R2> ip nhrp map multicast dynamic
**######## Set the hub for next hop info and map with it's gateway address:**
**R2> ip nhrp nhs 192.168.1.1**
**R2> ip nhrp map 192.168.1.1 54.45.12.1**
**R2> ip nhrp map multicast 54.45.12.1**
R2> ip nhrp network-id 1
R2> tunnel source fastEthernet 0/0
R2> tunnel mode gre multipoint
R2> tunnel key 0
R2> tunnel protection profile DMVPN-PROFILE
R2> router eigrp 10
R2> network 192.168.1.0

**Other spokes will be the same as R2 except for the router's ip address:**
####### config t and copy IPSEC info from sh running confg on R2, paste.
R3> int tunnel0
R3> ip address 192.168.1.3 255.255.255.0
####### Copy and paste the rest of the tunnel from first running config here, then make EIGRP declaration:
R3> router eigrp 10
R3> network 192.168.1.0
That's all. Do this for as many spokes as you want and it "just works"

sh crypto isakmp sa -- this is really just to show phase 1 info - does show tunnel is up or not
sh crypto ipsec sa -- this is for actual tunnel stats - shows traffic data and encryption %'s

So, what happens when the "hub" goes down? Is there a way to have failover/redundancy? It turns out, when naming NHS's you're not limited to 1 IP addy- you just need map commands for all added. This addresses the SPOF problem.

# *TCP Ports - Interactions and Scanning*

| | |
|---|---|
| SYN | Synchronize. Initiates a connection between hosts. |
| ACK | Acknowledge. Established connection between hosts. |
| PSH | Push. System is forwarding buffered data. |
| URG | Urgent. Data in packets must be processed quickly. |
| FIN | Finish. No more transmissions. |
| RST | Reset. Resets the connection. |

**TCP Connect and SYN stealth (aka half-open) scans**
TCP connect (-sT) is the most reliable scan type but also the noisiest and most detectable since it attempts the standard 3-way TCP handshake to a port. If that port is open and unfiltered, it will think there is a connection attempt it is waiting for. When a SYN stealth scan (-sS) receives an ACK response it will shoot back a RST to slam the connection shut. It is also gives more clear results than inverse scans. To Nmap, SYN/ACK means port is open, RST means port is closed and no response (even on retrying) or ICMP back means it is filtered en route.

Packet fragmentation can evade IDS packet filtering and detection. A stealth SYN can be used in a way that splits the TCP header over several IP packets to mess with firewalls. Some firewalls may have rule sets that block IP fragmentation queues, but might not due to the adverse effect on network performance
Example: nmap -sS -T4(time delay) -A -f -v 192.168.0.2

**Inverse TCP Flag scanning** sends packets with various TCP flags (or no flags) enabled.
- *When the port is open (or firewall-filtered), no response comes from the host.*
- *When the port is closed, a RST/ACK is received from the target host.*

It's often said these are called inverted since responses are sent back only by closed ports. Inverse TCP scans are named for the flags they use, NMap has -sF (FIN scan, with only the FIN flag set), -sN (Null scan with no flags set) and -sX for an Xmas scan setting all three FIN, URG, PUSH flags. Nmap doesn't have a URG and PSH similar individual shortcut (like --flag URG), Hping has -U and -P, but these don't yield different results than other inverse scans (NUL, FIN and Xmas all you really need). A maimon scan sends packets with both the FIN and ACK flags set. It was considered more useful but not in modern times.

NULL, FIN and Xmas scans: if we get a RST, we know the port is closed. If it's not closed, the remote host shouldn't respond so it's either open or filtered

Caveats: According to RFC 793, a RST/ACK packet must be sent for connection reset, when the port is closed on host side. RFC 793 is completely ignored in Windows, so you won't get a RST/ACK response when trying to interact with the closed port. So, is only effective when target is not a Windows OS. If an "ICMP unreachable" is returned Nmap considers the port firewall-filtered. These might get through some firewalls (FIN most likely), and sometimes no response doesn't mean a port is open, since instead the packet was dropped by the firewall.
Finally, for ALL types of scans, just because a method can pass through a firewall does NOT necessarily mean it is slipping through undetected!

**ACK scans** (are not considered inverse)
Sent with just the ACK flag set, these never determines ports are closed or open (or even open/filtered). It is used to map out firewall rulesets, determining whether they are stateful or not and which ports are filtered, which usually means running other scans and cross-referencing the results. The simplest usage shown below, determines getting ***a RST back means "unfiltered"*** (RFC 793 again), and either an ICMP unreachable or no result at all is "filtered" (firewalls that block usually make no response or send back an ICMP destination unreachable)

```
# nmap -sA -T4 scanme.nmap.org
        Not shown: 994 filtered ports
        22/tcp  unfiltered ssh
        25/tcp  unfiltered smtp …
```

Forbidding anything but outbound connections is fine, but blocking ACK packets without state info doesn't tell which side started the connection. To block unsolicited ACK packets and allow packets belonging to legitimate connections, firewalls must stateful. Either way, the stateless approach is still quite common (netfilter/iptables, basic zone-based, etc). Cross-reference a SYN scan below shows us both getting responses; SYN scan says 98/100 ports are filtered, ACK says all are filtered (all getting RST back). This is a stateless use of "iptables -A INPUT -m multiport -p tcp --destination-port 22,80 -j ACCEPT" on the firewall.

**# nmap -sS -p1-100 -T4 para**
Not shown: 98 filtered ports
22/tcp open ssh
80/tcp closed http
Nmap done: 1 IP address (1 host up) scanned in 3.81 seconds

**# nmap -sA -p1-100 -T4 para**
All 100 scanned ports on para (192.168.10.191) are: unfiltered

Nmap done: 1 IP address (1 host up) scanned in 0.70 seconds
The ACK scan shows some packets are probably reaching the destination host- firewall forgery is always possible. Other scan types, such as FIN scan, may even be able to determine which ports are open and thus infer the purpose of the hosts. While you may not be able to establish TCP connections to those ports, they can be useful for determining which IP addresses are in use, OS detection tests, certain IP ID shenanigans, and as a channel for tunneling commands to rootkits installed on those machines. Such hosts may be useful as zombies for an IP ID idle scan.

Two ways of interpreting other info from ACK scans can be more revealing: checking the time-to-live (TTL) field and the WINDOW field of received packets

It doesn't tell us if a port is open or closed, but it does try to tell us if the
firewall is stateful (keeps tracks of connections) or not (probably just denies
incoming SYN packets).
If the firewall is non-stateful and just drops SYN packets, an ACK will get in
because it looks like a reply to something from the other side.
If an open OR closed port receives an unexpected ACK, it should send a RST back.
So if we get a RST back, then it means the firewall is non-stateful (or there's
just not one in place). If we don't get a response, or some ICMP unreachable is
sent, it's most likely filtered.

TTL-Base Analysis:
TTL value can be used as a marker of how many systems the packet has hopped through
Below, the value on the RST packets returned by port 22 is 50, whereas the other ports return a value of 80. This suggests that port 22 is open on the target host because the TTL value returned is smaller than the TTL boundary value of 64*.
1: host 192.168.0.12 port 21: F:RST TTL:80 WIN:0
2: host 192.168.0.12 port 22: F:RST **TTL:50** WIN:0
The *firewalk* assessment tool is similar http://www.packetfactory.net/projects/firewalk/.

Window-based Analysis:
If window field has non-zero value then port is open, no response, presumed filtered
1: host 192.168.0.20 port 22: F:RST -> ttl: 64 **win: 512**
2: host 192.168.0.20 port 23: F:RST -> ttl: 64 win: 0

The advantage of using ACK flag probe scanning is detection is difficult (for both IDS and host-based systems). The disadvantage is it relies on TCP/IP stack implementation bugs, which are prominent in BSD-derived systems but not in many other modern platforms.

hping3 -A 72.14.207.99 -p 80 -c 1

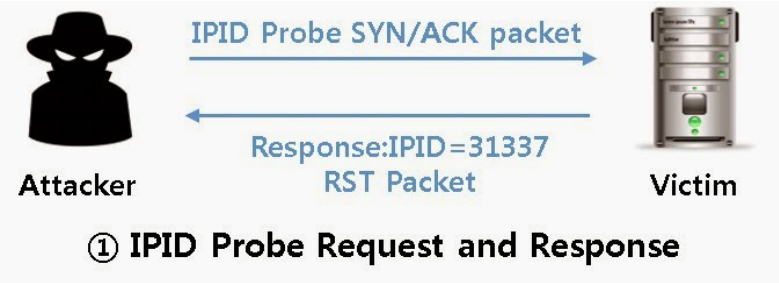
nmap -sW

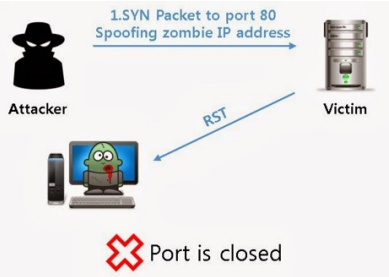
Some systems use a positive Window size for open ports, and zero for closed. Fewer systems do the exact opposite. If you scan and get tons of open ports and just a few closed ones, chances are it's the opposite. And some systems don't do either, so you can't always trust it.

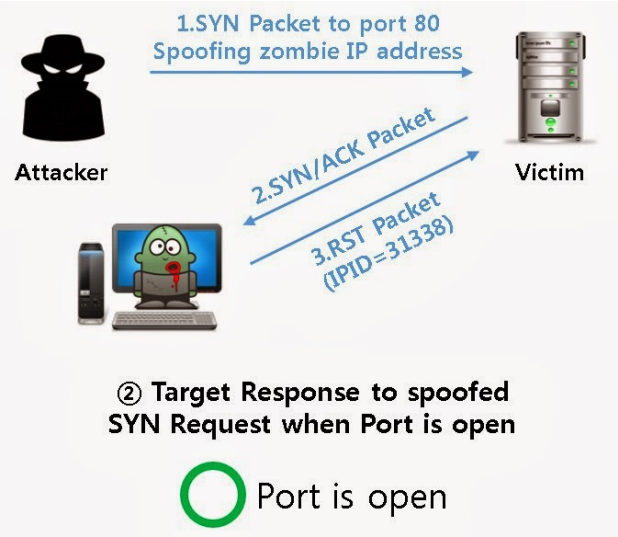
**IDLE scan/ IP ID header scan - IP ID (IP fragment ID number)**
Sends a spoofed source address to a computer to find out what services are available for complete blind scanning of a remote host. This is accomplished by spoofing another computer's IP address. No packet is send from your own IP address; instead, another host is used to scan the remote host and determine the open ports. This is done by expecting the sequence number of the zombie host and if the remote host checks the IP of the scanning party, the IP of the zombie machine will show up. You don't need access to the zombie machine to do any of this.

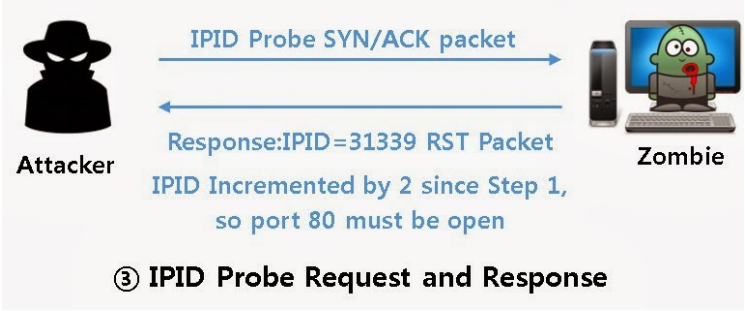
As noted, to determine if port is open, SYN scan full or stealth, get back an RST if closed, SYNACK if open.
Consider:
- An unsolicited SYNACK is responded to with a RST; An unsolicited RST will be ignored
- Each IP packet has a fragment ID (IPID), incremented for each sent. Check to get # of packets sent since probe

Step 1) Send an unsolicited SYNACK to the target to get a RST containing IP ID (say it's 31337)
Step 2) Spoof the IP address of your zombie machine, and send a SYN packet to the target (port 80)
- If the port is open, the target will send a SYNACK to the zombie and it will shoot a RST back to the target. (IP ID predicted is 31338)
- If the port is closed, the target will send a RST to the zombie host, and it won't respond
Step 3) Probe the target again with another SYNACK. The IP ID should have incremented by 2 from the last RST obtained in step 1 if the tested port on the target is open.




nmap -Pn -p- -sI www.host123.com www.host345.com
Idlescan using zombie www.host123.com (192.130.18.124:80); Class: Incremental
Nmap scan report for 198.182.30.30.110
(the 40321 ports scanned but not shown below are in the state: closed)

| Port | State | Service |
|---|---|---|
| 21/tcp | open | ftp |
| 25/tcp | open | smtp |
| 80/tcp | open | http |

Nmap done: 1 IP address (1 host up) scanned in 1931.23 seconds

**ICMP Echo scanning/List scan**
ICMP echo scanning is used to discover live machines by pinging all the machines in the target network. ICMP probes sent to the broadcast or network address are relayed to all the host addresses in the subnet. The live systems will send ICMP echo reply message to the source of ICMP echo probe.
nmap -P 192.168.0.0/24 --OR-- nmap -sn 192.168.0.2 --OR-- nmap -sL -v 192.168.2.5

**UDP Scanning -sUV**
Sends UDP packets to each target port, often empty but some commonly known port services need a specific payload, so Nmap tries add it in (the -V option checks Nmap's database). Some listening applications will still discard empty packets as invalid, firewalls also drop packets without responding, and UDP might just drop packets or time out. So, a UDP response means it is open, no response can be either open or filtered (can't tell!). Nmap running with application version detection (-sUV) helps there a bit.
If you get a UDP response back, it's open
If the port is closed, it will likely shoot back an ICMP port unreachable (type 3/code 3)
If it is another ICMP unreachable error (type 3, code 1, 2, 9, 10, or 13) it's likely filtered.

Some OSs limit the frequency of ICMP Port Unreachable messages, Nmap adjusts accordingly. Microsoft-based OSs do not usually implement any type of ICMP rate limiting. Not usually useful for most types of attack, but it can reveal information about some exploitable services (DHCP, SNMP, NFS, etc). Malware often uses UDP
nmap -sU -v 192.168.0.2

***Some flags available in Nmap (see https://nmap.org/book/man-briefoptions.html )***

| | | | | | |
|---|---|---|---|---|---|
| -sT | TCP connect scan | -sR | RPC scan | -oN | Normal output |
| -sS | SYN scan | -sL | List/DNS scan | -oX | XML output |
| -sF | FIN scan | -sI | Idle scan | -oG | Greppable output |
| -sX | XMAS tree scan | -Po | Don't ping | -oA | All output |
| -sN | Null scan | -PT | TCP ping | -T Paranoid | Serial scan; 300 sec between scans |
| -sP | Ping scan | -PS | SYN ping | -T Sneaky | Serial scan; 15 sec between scans |
| -sU | UDP scan | -PI | ICMP ping | -T Polite | Serial scan; .4 sec between scans |
| -sO | Protocol scan | -PB | TCP and ICMP ping | -T Normal | Parallel scan |
| -sA | ACK scan | -PB | ICMP timestamp | -T Aggressive | Parallel, 300 sec timeout, 1.25 sec/probe |
| -sW | Windows scan | -PM | ICMP netmask | -T Insane | Parallel, 75 sec timeout, 0.3 sec/probe |

IP PROTOCOL SCAN
Looks for supported IP protocols rather than open ports; sends raw IP packets with different values in the protocol field of the header. Instead of looking for “ICMP Port Unreachable”, it looks for ICMP Protocol Unreachables to tell if it's unsupported (somewhat “closed”). If we get a response back in the same protocol, it's supported (somewhat “open”). If we get some different ICMP unreachable, it's probably filtered. If we don't get anything back, it's either open (and didn't reply) or filtered.
# nmap -sO 192.168.10.1
# hping3 -c 1 --rawip --ipproto 0 192.168.10.1
# hping3 -c 1 --icmp 192.168.10.1
# hping3 -c 1 --rawip --ipproto 2 192.168.10.1

FTP bounce scanning
Hosts running outdated FTP services can relay numerous TCP attacks, including port scanning. There is a flaw in the way many FTP servers handle connections using the PORT command (see RFC 959 or technical description) that allows data to be sent to user-specified hosts and ports. In their default configurations, the FTP services running on the following older Unix-based platforms are affected
The FTP bounce attack can have a far more devastating effect if a writable directory exists because a series of commands or other data can be entered into a file and then relayed via the PORT command to a specified port of a target host. For example, some- one can upload a spam email message to a vulnerable FTP server and then send this email message to the SMTP port of a target mail server. Figure 4-9 shows the parties involved in FTP bounce scanning.

1. The attacker connects to the FTP control port (TCP port 21) of the vulnerable FTP server that she is going to bounce her attack through and enters passive mode, forcing the FTP server to send data to a specific port of a specific host:
QUOTE PASV
    227 Entering Passive Mode (64,12,168,246,56,185).
2. A PORT command is issued, with an argument passed to the FTP service telling it
to attempt a connection to a specific TCP port on the target server; for example, TCP port 23 of 144.51.17.230:
    PORT 144,51,17,230,0,23
    200 PORT command successful.
3. After issuing the PORT command, a LIST command is sent. The FTP server then
attempts to create a connection with the target host defined in the PORT command issued previously:
LIST
    150 Opening ASCII mode data connection for file list
    226 Transfer complete.
If a 226 response is seen, then the port on the target host is open. If, however, a 425 response is seen, the connection has been refused:
LIST
    425 Can't build data connection: Connection refused
Nmap supports FTP bounce port scanning with the –P0 and –b flags used in the following manner:
  nmap –P0 –b username:password@ftp-server:port <target host>
The –P0 flag must be used to suppress pinging of the target host, as it may not be accessible from your location (e.g., if you are bouncing through a multihomed FTP server). Also, you may not want your source IP address to appear in logs at the target site.

---

Proxy bounce scanning
Attackers bounce TCP attacks through open proxy servers. Depending on the level of poor configuration, the server will sometimes allow a full-blown TCP port scan to be relayed. Using proxy servers to perform bounce port scanning in this fashion is often time-consuming, so many attackers prefer to abuse open proxy servers more efficiently by bouncing actual attacks through to target networks.
ppscan.c, a publicly available Unix-based tool to bounce port scans, can be found in source form at:
http://examples.oreilly.com/networksa/tools/ppscan.c http://www.phreak.org/archives/exploits/unix/network-scanners/ppscan.c

---

(NMAP) When scanning hardened environments, you should use the -Pn flag to force scanning of each address within scope. A slower timing policy (such as -T2) is also useful, as an aggressive policy will trigger SYN flood protection by firewalls

SYN probes - four response variants: a packet with SYN/ACK flags indicating an open port, RST/ACK denoting closed, no response or an ICMP type 3 message implying a filter

This runs three Nmap scans to identify accessible hosts across TCP, SCTP, and UDP. Optionally, load a list of targets into Nmap from a file using the -iL flag.
nmap –T4 –Pn –v –n –sS –F –oG /tmp/tcp.gnmap 192.168.0.0/24
nmap –T4 –Pn –v –n –sY –F –oG /tmp/sctp.gnmap 192.168.0.0/24
nmap –T4 –Pn –v –n –sU –p53,111,123,137,161,500 -oG /tmp/udp.gnmap 192.168.0.0/24
These scans generate output in /tmp with gnmap file extensions. UDP results may contain false positives. If the UDP dataset looks noisy (i.e. all the hosts are reporting to have open ports), then simply disregard it. Once you're happy with the contents of these files, use grep and awk to generate a refined list of targets, as follows:
grep open /tmp/*.gnmap | awk '{print $2}' | sort | uniq > /tmp/targets.txt
This list should then be fed into four subsequent scans: A fast TCP scan of common services
nmap -T4 -Pn -v --open -sS -A -oA tcp_fast -iL /tmp/targets.txt
A TCP scan of all ports:
nmap -T4 -Pn -v --open -sS -A –p0-65535 -oA tcp_full -iL /tmp/targets.txt
An SCTP scan of all ports:
nmap -T4 -Pn -v --open -sY –p0-65535 -oA sctp -iL /tmp/targets.txt
A UDP scan of common services:
nmap -T3 -Pn -v --open –sU -oA udp -iL /tmp/targets.txt

The -oA flag will generate multiple output files for each scan type, including a gnmap file that you can easily parse, and a full text file (i.e. tcp_fast.nmap) that is human- readable. These scanning modes do not perform service fingerprinting or deep analysis of the exposed network services

Same procedure for IPv6 - first sweeping IPv6 address space for hosts running common network services, and then perform full scanning of that subset). When TCP sweeping large IPv6 networks, I recommend reducing the port list to increase speed, from -F (100 common ports) to -p22,25,53,80,111,139,443.
Upon preparing a list of targets (e.g. /tmp/targets.txt) from host discovery and sweeping, run the same four scans as before, using the -6 flag to perform the scanning over IPv6:
nmap -6 -T4 -Pn -v --open -sS -A -oA ipv6_tcp_fast -iL /tmp/targets.txt
nmap -6 -T4 -Pn -v --open -sS -A –p0-65535 -oA ipv6_tcp_full -iL /tmp/targets.txt
nmap -6 -T4 -Pn -v --open -sY –p0-65535 -oA ipv6_sctp -iL /tmp/targets.txt
nmap -6 -T3 -Pn -v --open -sU -oA ipv6_udp -iL /tmp/targets.txt

---

**SCTP**
Stream Control Transmission Protocol (SCTP) is a transport protocol that sits alongside TCP and UDP. Intended to provide transport of telephony data over IP, the protocol duplicates many of the reliability features of SS7, and underpins a larger family of protocols known as SIGTRAN. SCTP is supported by operating systems including IBM AIX, Oracle Solaris, HP-UX, Linux, Cisco IOS, and VxWorks.

**SCTP chunk types**

| *ID* | *Value* | *Description* |
|---|---|---|
| 0 | DATA | Payload data |
| 1 | INIT | Initiation |
| 2 | INIT ACK | Initiation acknowledgement |
| 3 | SACK | Selective acknowledgement |
| 4 | HEARTBEAT | Heartbeat request |
| 5 | HEARTBEAT ACK | Heartbeat acknowledgement |
| 6 | ABORT | Abort |
| 7 | SHUTDOWN | Shutdown |
| 8 | SHUTDOWN ACK | Shutdown acknowledgement |
| 9 | ERROR | Operation error |
| 10 | COOKIE ECHO | State cookie |
| 11 | COOKIE ACK | Cookie acknowledgement |
| 12 | ECNE | Explicit congestion notification echo |
| 13 | CWR | Congestion window reduced |
| 14 | SHUTDOWN COMPLETE | Shutdown complete |

---
Tools such as Nmap and SING don't identify these responses from private addresses (behind NAT), as low- level stateful analysis of the traffic flowing into and out of a network is required. A quick and simple example of this behavior can be seen in the ISS BlackICE personal firewall event log in Figure 4-1 as a simple ICMP ping sweep is performed.

---
It is beneficial to run a network sniffer such as Ethereal or tcpdump during testing to pick up on unsolicited ICMP responses, including “ICMP TTL exceeded” (type 11 code 0) messages, indicating a routing loop, and “ICMP administratively prohibited” (type 3 code 13) messages, indicating an ACL in use on a router or firewall.
---
OS Fingerprinting Using ICMP
Ofir Arkin’s Xprobe2 utility performs OS fingerprinting primarily by analyzing responses to ICMP probes.
---
Another SYN port scanner worth mentioning is Scanrand, a component of the Paketto Keiretsu suite. Paketto Keiretsu contains a number of useful networking utilities that are available at http://www.doxpara.com/read.php/code/paketto.html. For Windows, Foundstone’s SuperScan is an excellent port scanning utility with good functionality, including banner grabbing. SuperScan is available from http:// examples.oreilly.com/networksa/tools/superscan4.zip
---
Scanrand is well designed, with distinct SYN probing and background listening components that allow for very fast scanning. Inverse SYN cookies (using SHA1) tag out- going probe packets, so that false positive results become nonexistent, as the listening component only registers responses with the correct SYN cookies.  Scanrand is much faster than bulkier scanners, such as Nmap.

Unicornscan (http://www.unicornscan.org) is another tool that performs fast half- open scanning. It has some unique and very useful features, and it is recommended for advanced users.
(UDP payload scan) against the 10.3.0.1 candidate within my environment, results are as follows:

root@kali:~# unicornscan -mU 10.3.0.1
UDP open domain[ 53] from 10.3.0.1 ttl 128
UDP open netbios-ns[ 137] from 10.3.0.1 ttl 128

UDP scanning results across tools may vary. Nmap provides a comprehensive option with -sV, but testing of a single host using the -F option (scanning 100 ports) takes around 10 minutes to complete.

---
Using malformed TCP flags to probe a target is known as an inverted technique because responses are sent back only by closed ports. RFC 793 states that if a port is closed on a host, an RST/ACK packet should be sent to reset the connection. To take advantage of this feature, attackers send TCP probe packets with various TCP flags set.
---

Vscan is another Windows tool you can use to perform inverse TCP flag scanning. The utility doesn’t require installation of WinPcap network drivers; instead it uses raw sockets within Winsock 2 (present in Windows itself). Vscan is available from http://examples.oreilly.com/networksa/tools/vscan.zip.

---

SING (Send ICMP Nasty Garbage) - http://sourceforge.net/projects/sing
Ability to transmit and receive spoofed packets, send MAC-spoofed packets, and support the transmission of many other message types, including ICMP address mask, timestamp, and information requests, as well as router solicitation and router advertisement messages

ICMPScan -  http://www.bindshell.net/tools/icmpscan
Bulk scanner derived from Nmap that sends type 8, 13, 15, and 17 ICMP; can process inbound responses by placing the network interface into promiscuous mode, thereby identifying internal IP addresses and machines that respond from probes sent to subnet network and broadcast addresses. Because ICMP is a connectionless protocol, it is best practice to resend each probe (using –r 1) and set the timeout to 500 milliseconds (using -t 500). We also set the tool to listen in promiscuous mode for unsolicited responses (using the –c flag).

# SSH Tunnels

**Shortened version of what's at https://robotmoon.com/ssh-tunnels/**

## ***Port forwarding - Forwards a port from one system (local or remote) to another***

**Local port forwarding-** Forwards connections from a port on a local system to a port on a remote host.
To forward traffic on the SSH client to some destination through an SSH server. This lets you access remote services over an encrypted connection as if they were local services.

Example use cases:

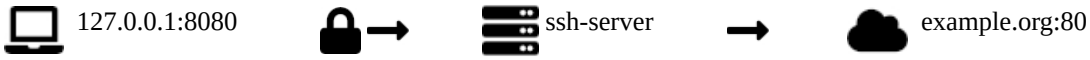
- Accessing a remote service (redis, memcached, etc.) listening on internal IPs
- Locally accessing resources available on a private network
- Transparently proxying a request to a remote service

If you want to use a secure connection to access a remote service that communicates over plaintext. For example, redis and memcached all use plaintext protocols. If you securely access one of these services on a remote server over public networks, you can tunnel a connection from your local system to the remote server instead of having it listen over the public internet.

**SSH -L is FROM-INSIDE-IP:port:TO-OUTSIDE-ACCESS-IP:port SSHServerName**

### ssh -L 127.0.0.1:8080:example.org:80 ssh-server

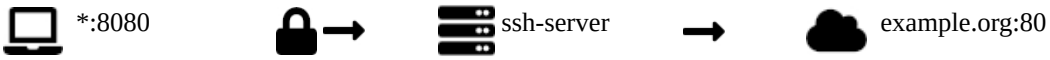
Forward connections to 127.0.0.1:8080 on your local system to port 80 on example.org through ssh-server.




The traffic between your local system and ssh-server is wrapped in an SSH tunnel, but the traffic between ssh-server and example.org is not. From the perspective of example.org the traffic originates from ssh-server.

### ssh -L 8080:example.org:80 ssh-server
### ssh -L *:8080:example.org:80 ssh-server

Forward connections to port 8080 on all interfaces on your local system to example.org:80 through a tunnel to ssh-server.




### ssh -L 192.168.0.1:5432:127.0.0.1:5432 ssh-server

Forward connections to 192.168.0.1:5432 on your local system to 127.0.0.1:5432 on ssh-server. Note that 127.0.0.1 here is localhost from the viewpoint of ssh-server.

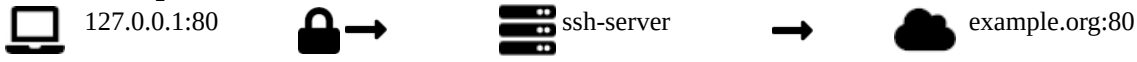

***SSH configurations. Make sure:***

TCP forwarding is enabled on the SSH server. In /etc/ssh/sshd_config set "AllowTcpForwarding yes"
If you're forwarding ports on interfaces other than 127.0.0.1 then you'll need to enable GatewayPorts on your local system, either within ssh_config or as a command-line option
In /etc/ssh/ssh_config set "GatewayPorts yes"

**Forwarding from privileged ports**

If you want to open a privileged port (ports 1-1023) to forward traffic, you'll need to run SSH with superuser privileges on the system that opens the port.

### sudo ssh -L 80:example.com:80 ssh-server



**Remote port forwarding -** Forwards a port on a remote system to another system

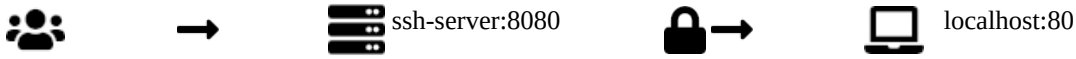
To forward traffic on an SSH server to a destination through either the SSH client or another remote host. This gives users on public networks access to resources on private networks. Example use cases:

Making a local development server available over a public network
Granting IP-restricted access to a remote resource on a private network

**SSH -R is OUTSIDE-ACCESS-IP:port:TO-INSIDE-IP:port SSHServerName**

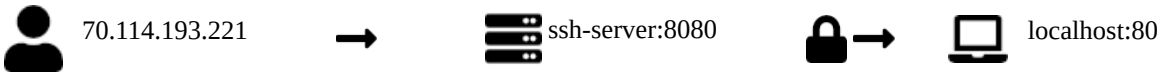
### ssh -R 8080:localhost:80 ssh-server

Forwards traffic to all interfaces on port 8080 on ssh-server to localhost port 80 on your local computer. If one of these interfaces is available to the public internet, traffic connecting to port 8080 will be forwarded to your local system.




### ssh -R 70.114.193.221:8080:localhost:80 ssh-server

Forwards traffic to ssh-server:8080 to localhost:80 on your local system but only allows access to the SSH tunnel entrance on ssh-server from IP address 70.114.193.221. This requires the GatewayPorts clientspecified option in the server's sshd_config.




### ssh -R 8080:example.org:80 ssh-server

Forwards traffic to all interfaces on ssh-server:8080 to localhost:80 on your local system. From your local system, traffic is then forwarded to example.org:80. From the perspective of example.org the traffic is originating from your local system.


**SSH server configurations set in /etc/ssh/sshd_config**

By default, forwarded ports are not accessible to the public internet. You'll need to add this to your sshd_config on your remote server to forward public internet traffic to your local computer.

GatewayPorts yes

If you want only *specific* clients to be allowed access, you can use this instead:

GatewayPorts clientspecified

**Dynamic port forwarding-** To forward traffic from a range of ports to a remote server

Opens a SOCKS proxy on the SSH client that lets you forward TCP traffic through the SSH server to a remote host.

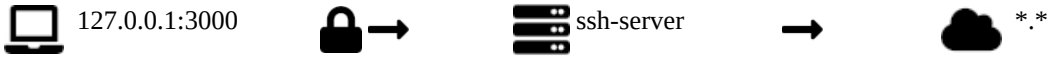
### ssh -D 3000 ssh-server

Opens a SOCKS proxy on port 3000 of all interfaces on your local system. This allows you to forward traffic sent through the proxy to the ssh-server on any port or destination host. By default, SSH will use the SOCKS5 protocol, which forwards TCP and UDP traffic.


### ssh -D 127.0.0.1:3000 ssh-server

Opens a SOCKS proxy on 127.0.0.1:3000 on your local system. When you have a SOCKS proxy running, you can configure your web browser to use the proxy to access resources as if connections were originating from ssh-server. For example, if ssh-server had access to other servers within a private network, you could access those other servers locally as if you were on the network, without a VPN. Test the proxy like this:

curl -x socks5://127.0.0.1:12345 https://example.org




**SSH client configuration - /etc/ssh/ssh_config**

If you want the SOCKS proxy to be available to more interfaces than just localhost, this needs to be set: GatewayPorts yes

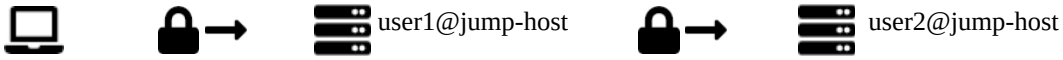
When configuing the SSH client, you can also configure it with a command instead of editing manually:

ssh -o GatewayPorts=yes -D 3000 ssh-server

## **Jump hosts and proxy commands -** Transparently connecting to a remote host through intermediate hosts

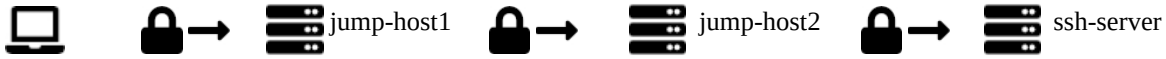
### ssh -J user1@jump-host user2@remote-host
### ssh -o "ProxyJump user1@jump-host" user2@remote-host

Establishes a SSH connection with jump-host and forwards TCP traffic to remote-host, connecting to remote-host. The -J command should work out of the box if jump-host already has SSH access to remote-host. If it does not, you can use agent forwarding to forward the SSH identity of your local computer to remote-host.



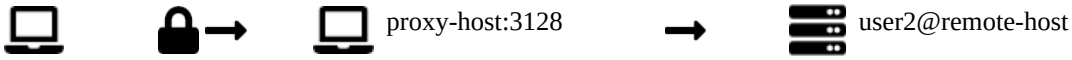

### ssh -J jump-host1,jump-host2 ssh-server

To specify multiple comma-separated jump hosts.




### ssh -o ProxyCommand="nc -X 5 -x localhost:3000 %h %p" user@remote-host

Connecting to a remote server through a SOCKS5 proxy using netcat. From the perspective of the server, the originating IP is from proxy-host. However, the SSH connection itself is end-to-end encrypted so proxy-host only sees an encrypted stream of traffic between the local system and remote-host.




**SSH client configuration**

To enable agent forwarding, you can use ssh-add to add your local SSH identity to your local ssh agent.
The command ssh-add will simply add the currently logged-in user, while adding configuration to /etc/ssh/ssh_config will be for system-wide SSH client configuration settings

## **Reliable SSH Tunnels -** Keeping SSH tunnels open through network failures

The commands listed above work on an ad-hoc basis, but if you want to maintain SSH tunnels through network outages or unreliable connections, you'll have to do some additional setup. Another issue is, by default, the TCP connection used to establish an SSH tunnel may time out after a period of inactivity. To prevent timeouts, you can configure the server and/or the client to send heartbeat messages by adding this into /etc/ssh/sshd_config:

On the server, add:

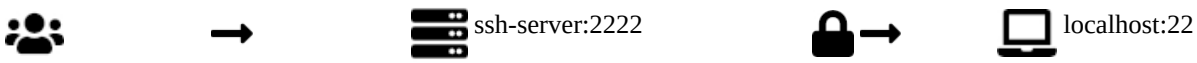
```
ClientAliveInterval 15
ClientAliveCountMax 4
```

On the client add:

```
ServerAliveInterval 15
ServerAliveCountMax 4
```

## Using AutoSSH to restqablish dropped connections

While the above options may prevent a connection from dropping due to inactivity, they will not re-establish dropped connections. To ensure that an SSH tunnel will be re-established, you can use autossh, which builds an SSH tunnel and monitors its health.
AutoSSH accepts the same arguments for port forwarding as SSH.



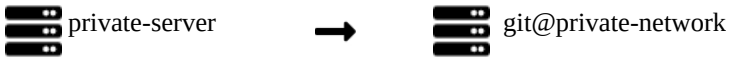

This establishes a reverse tunnel that comes back after network failures. By default, AutoSSH will open extra ports on the SSH client and server for health checks. If traffic appears to no longer pass between the health check ports, AutoSSH will restart the SSH tunnel.

### autossh -R 2222:localhost:22 -M 0 -o "ServerAliveInterval 10" -o "ServerAliveCountMax 3" remote-host

Using the -M 0 flag disables the health check ports and allows the SSH client to handle the health checks. In this example, the SSH client expects the server to send a heartbeat every 10 seconds. If 3 heartbeats fail in a row, the SSH client exits, and AutoSSH will re-establish a new connection.

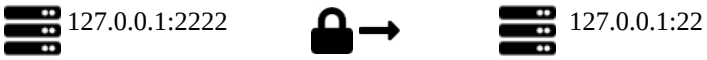
## Transparent access to remote resource on a private network

Let's say there's a git repository on a private network that's only accessible through a private server on the network.
This server is not accessible to the public internet. You have direct access to the server, but don't have VPN access to the private network.




For convenience, you'd like to access this private git repository as if you were connecting to it directly from your local system. If you have SSH access to another server that's accessible from both your local system and the private server, you can accomplish this by establishing an SSH tunnel and using a couple of ProxyCommand directives.

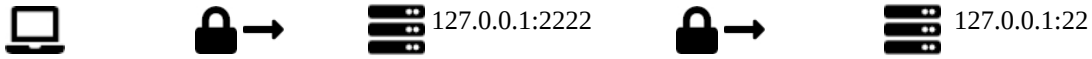
**ssh -L 127.0.0.1:22:127.0.0.1:2222 intermediate-host**
This forwards port 2222 on intermediate-host to port 22 on the private server.




Now, if you SSH to port 2222 from intermediate-host, you're connecting to the SSH server on the private server despite the private server not being accessible by the public internet.

**ssh -p 2222 user@localhost**

Now you can now access the private git repository as if you were on the private network.




If you'd like to make the backdoor even more convenient, you can add some directives to your local ~/.ssh/config:

```
Host git.private.network
    HostName git.private.network
    ForwardAgent yes
    ProxyCommand ssh private nc %h %p

Host private
    HostName localhost
    Port 2222
    User private-user
    ForwardAgent yes
    ProxyCommand ssh tunnel@intermediate-host nc %h %p
```

## SSH command-line flags

These are some useful SSH command-line flags when establishing tunnels. For simplicity, the examples given here leave these out.

| | |
|---|---|
| -f | Forks the ssh process into the background |
| -n | Prevents reading from STDIN |
| -N | Do not run remote commands. Used when only forwarding ports |
| -T | Disables TTY allocation |

Here's an example of a command you would run to create an SSH tunnel in the background that forwards a local port through the ssh server:

**ssh -fnNT -L 127.0.0.1:8080:example.org:80 ssh-server**

---

In some misconfigured systems supporting SSH you may not be able to gain a shell via ssh, but you can port forward and such. There is an article discussing that here:
https://www.pixelstech.net/article/1328532389-SSH-Security-and-You---bin-false-is-%2Anot%2A-security

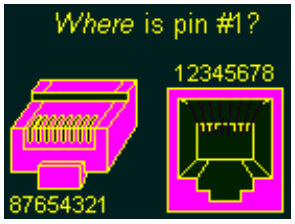
# Socat to Netcat Commands Task Quick Reference

*NOTE: In BSD/ MacOS versions of Netcat, using -l and -p in the same directive together throws an error. Required is "nc -l <port>..." the -l option immediately followed by port number. In that case -lABCD are acceptable as long as any of ABCD dont also require a value follow them, example: "nc -lks <port>" won't be acceptable, but "nd -lk <port> -s <device>" will.*

| **Connections/ Data Transfer:** | **Socat example** | **Netcat example** |
|---|---|---|
| Open TCP connections: | socat tcp-listen:<port>,fork tcp:<host>:<port> | nc -l <port> -c "nc <host> <port>" (protocol=<protocol> supported) |
| Open UDP connections: | socat udp-listen:<port>,fork udp:<host>:<port> | nc -u -l <port> -c "nc -u <host> <port>" (protocol=<protocol> supported) |
| Connect to remote systems: | socat tcp:<host>:<port>,fork (connect to specific port) | nc <host> <port> # connect to specific port) |
| Connect to serial devices: | socat serial:<device>,raw,echo=1 tcp:<host>:<port> | nc -l -p <port> <device> # Serial to TCP |
| Send/ write files: | socat file:<file>,fork tcp:<host>:<port> | nc <host> <port> < <file> |
| Receive/ read files: | socat tcp-listen:<port>,fork write:<file> | nc -l <port> > <file> |
| **Tunneling and Proxying:** | ***Using SSH for tunnels is a better option than nc and socat*** | ***See https://robotmoon.com/ssh-tunnels/*** |
| Create basic tunnels: | socat tcp-listen:<port>,fork tcp:<remote_host>:<remote_port>user=user,password=pass | nc -l <port> -c "nc <remote_host> <remote_port>" |
| Establish nested tunnels: | socat tcp-listen:<port1>,fork tcp:<host1>:<port2>,fork tcp:<remote_host>:<remote_port> | nc -l -p <port1> \| nc <host1> <port2> & nc <remote_host> <remote_port> |
| SSL/TLS tunnels: | socat openssl:<cipher>,fork tcp:<host>:<port> | nc -l <port> \| openssl enc -d des3 # enlist SSL |
| SOCKS tunnel (firewall traversal): | socat tcp:<host>:<port>,proxy:socks:<proxy_server>: <proxy_port>,fork tcp:<remote_host>:<remote_port> | proxychains nc -l <host> <port> \| nc -x <proxy_server>: <proxy_port> <remote_host> <remote_port> |
| PKI authentication: | socat tcp-listen:<port>,fork pty,ptyexec:/bin/bash,user= remote_user, keyfile=/path/to/private_key | ssh -i /path/to/private_key user@host "nc -l -p <port>" |
| SSH tunnel: | socat tcp-listen:<port>,fork exec:ssh -N -f -L <port>:localhost:<remote_port> <user>@<remote_host> | ssh -N -f -L <port>:localhost:<remote_port> <user>@<remote_host> & nc -l -p <port> |
| Proxy TCP connections: | socat tcp-listen:<port1>,fork tcp:<host1>:<port2> | nc -l -p <port1> \| nc <host1> <port2> *(socat is better for this)* |
| Basic IP forwarding: | socat tcp-listen:<port1>,fork TCP:<host2>:<port2> | *nc -l <port1> \| nc <host2> <port2>* *(socat is better for this)* |
| Port forwarding: | socat tcp-listen:<port1>,fork tcp:<host1>:<port2> | nc -l -p <port1> \| nc <host1> <port2> *(socat is better for this)* |
| UDP hole punching: | socat udp-listen:<port>,fork udp:<host>:<port> | nc -u -l <port1> -c "nc -u <host2> <port2> |
| Multiplexing connections: | socat tcp-listen:<port>,fork tcp:<host1>:<port1>,fork tcp:<host2>:<port2> *-In a script you could add error checks, logging, etc.* | *Netcat isn't really designed to do this but you could use it in a script to accomplish it, since it gets out of scope for a 'one-liner)* |
| **Other Networking:** | | |
| Serial communication: | socat serial:<device>,raw,echo=1 tcp:<host>:<port> | nc -l -p <port> <device> # TCP to serial |
| Unix domain sockets: | socat unix-listen:<path>,fork unix-connect:<path> | |
| Execute remote commands: | socat tcp:<host>:<port>,fork exec:<command> | nc -l -p <port> \| bash -i ***SSH is preferable- both socat and netcat are too insecure*** |
| Traffic shaping and bandwidth limiting: | socat tcp:<host>:<port>,fork rate:<bandwidth>,link:<delay> (<bandwidth> is desired rate and <delay> is desired latency) | *Doing this with nc would employ tc, which can do this by itself, so it's redundant)* |
| Load balancing and failover: | socat tcp-listen:<port>,fork tcp:<db1>:<port>,fork tcp:<db2>:<port>,roundrobin (distribute connections evenly) | nc <haproxy_host> <haproxy_port> *(inefficient since you still have to configure haproxy server to do the actual task)* |
| DNS resolution: | socat tcp-listen:<port>,fork tcp:dns:<dns_server>:53 | *This would employ dig to use a custom DNS which makes nc irrelevant/ redundant* |
| Capture network traffic: | socat tcp-listen:<port>,fork tcp:<host>:<port>,fork write:<filename> | nc -l <port> > traffic.log |
| Inspect and modify data: | socat tcp-listen:<port>,fork tcp:<host>:<port>,fork exec:<commands> | nc -l <port> \| <commands> |
| Port scanning: | *Not efficient. resort to nc or - better yet- just use nmap* | nc -z <host><start_port>-<end_port> *(just use nmap??)* |

***Some of these examples might show that you CAN use socat or netcat for these things, but that doesn't mean that's a GOOD way to do things. SSH is a good example, but so are nmap, etc. But, if you just need to mess with something like a directly connected device with no security concerns, nc and socat might be fine for tunnelling or remote commands instead of SSH!***

**Category 5/ 6, UTP (Unshielded Twisted Pair); RJ-45 8P8C (8 position, 8 contact) connector**
Twisting of the 2 wires in each of the 4 pairs minimizes interference (one is insulated by the other).
Each pair also even have a different "twist rate" to minimize interference between them. Each pair is transmit +/-, receive +/-
Shielded twisted pair surrounds each pair in metal shielding and uses a grounding wire.




The only difference between T568A and T568B wiring standards is that pairs 2 and 3 (orange and green) are swapped. Both configurations wire the pins "straight through", i.e., pins 1 through 8 on one end are connected to pins 1 through 8 on the other end.

As you can see below, a crossover cable for both T568A and T568B is ALMOST T568A wiring on one end and T568B on the other- making it easy to remember- HOWEVER- notice that pins 7 and 8 also switch to pins 4 and 5 (below, blue and brown switch places, as well as green and orange).
***So crossover is 1 and 2 to 3 and 6, and 4 and 5 to 7 and 8***

**Straight-Through Ethernet Cable Pin Out for T568B**

| RJ45 Pin # | Wire Color | 10/100Base-TX Signal | 1000Base-T Signal |
|---|---|---|---|
| **1** | White/Orange | Transmit+ | BI_DA+ |
| **2** | Orange | Transmit- | BI_DA- |
| **3** | White/Green | Receive+ | BI_DB+ |
| **4** | Blue | Unused | BI_DC+ |
| **5** | White/Blue | Unused | BI_DC- |
| **6** | Green | Receive- | BI_DB- |
| **7** | White/Brown | Unused | BI_DD+ |
| **8** | Brown | Unused | BI_DD- |

**Straight-Through and PoE Ethernet Cable Pin Out for T568A**

| RJ45 Pin # | Wire Color | 10/100Base-TX Signal | PoE |
|---|---|---|---|
| 1 | White/Green | Transmit+ | Mode A + |
| 2 | Green | Transmit- | Mode A + |
| 3 | White/Orange | Receive+ | Mode A - |
| 4 | Blue | Unused | Mode B + |
| 5 | White/Blue | Unused | Mode B + |
| 6 | Orange | Receive- | Mode A - |
| 7 | White/Brown | Unused | Mode B - |
| 8 | Brown | Unused | Mode B - |

**Crossover Cable Pin Outs for T568B**

| Pin # (END 1) | Wire Color | Pin # (END 2) | Wire Color |
|---|---|---|---|
| 1 | White/Orange | 1 | White/Green |
| 2 | Orange | 2 | Green |
| 3 | White/Green | 3 | White/Orange |
| 4 | Blue | 4 | White/Brown |
| 5 | White/Blue | 5 | Brown |
| 6 | Green | 6 | Orange |
| 7 | White/Brown | 7 | Blue |
| 8 | Brown | 8 | White/Blue |

One other type of wiring is a rollover/ console cable for switch and routers using the EIA-TIA 232 serial COM port. It is just straight-through in reverse order- picture cutting the end off a straight-through cable, turn it over, and put on a new connector. Often USB to serial converter and rollover cables are used.

Crossover cables are necessary when endpoints transmit on the same pin pair, they need to be crossed for compatibility. Use crossover for "like devices" not unlike:
**Transmits on Pins 1,2: workstation NICs, routers, cabled non-USB WAPs**
**Transmits on Pins 3,6: switches and hubs**
Some devices have auto-MDIX to autosense cable type and make the link work. Often not present.

Category 5 cabling is meant for 100Base-TX, but was first used as 10Base-T (wired 2-pair instead of 4) for legacy equipment. Cat 5e is an updated version of Cat5, offers reduced crosstalk, ok for 1000BASE-T
Cat6 cable is *preferred* for 1000BASE-T Ethernet networks. Some Cat6 is made of thicker wires (for example, 22 or 23 gauge instead of 24); more pair twisting gives thicker insulation for reduced crosstalk.

**Copper Ethernet Cabling Types**

| | | | |
|---|---|---|---|
| 10BASE2 (thinnet) | RG-58 coax | 10 Mbps | 185 meters |
| 10BASE5 (thicknet) | RG-8 coax | 10 Mbps | 500 meters |
| 10Base-T | Cat3 (POTS) or Cat5 | 10 Mbps | 100 meters |
| 100Base-T (fast ethernet) | Cat5 | 100 Mbit/s | 100MHz |
| 1000Base-T (gigabit ethernet) | Cat5e | 1Gbit/s | 100MHz |
| 10GBase-T | Cat6 | 1Gbit/s | 250MHz |
| 10GBase-T | Cat6a | 10Gbit/s | 500MHz, shielded |
| 10GBase-T | Cat7 (not TIA/EIA ratified) | 10Gbit/s | 600MHz, shielded |

*Cat 5/6/6a have different speed performance at different lengths and shorter is faster.*
*Cat5 can do 1000Base-T, but not guaranteed. Cat 5e/6 achieve maximum efficiency at 55 meters.*
*When sizing copper ethernet cables remember that cable should not extend more than **100m (~328ft)**.*

**Power over Ethernet (PoE)**
Endspan means PoE is built into switch. Midspan means a Inline Power Injector is used to add power
Mode A (phantom): Power and data use same twisted pairs; Mode B: separate pairs for each power and data.
IEEE 802.3af (802.3at type1) specifies a max 15.4W, 802.3at (type 2) specifies a max of 30W (typically 25W)

***Other Cabling Types and Specifications***

*All RG (Radio Guide) specs use a familiar CATV F-connector on coaxial cable. BASE means "baseband"*
**RG-6** coax is commonly used by local cable companies to connect individual homes to the distribution point. For higher frequency signals over longer distances (~70 meters). RG-6 replaced RG-9 with better shielding.
**RG-59** low-freq, shorter distance, like component video; older shielding, susceptible to UHF interference, impedance of 75 Ohms. RG-6 was preferred for video.
**RG-58** - early 10BASE2 max length 185 meters; impedance 50 Ohms. Better frequency range, shielding
**RG-8**- 10BASE5, 500 meters. RG-58 and -8 predicted to replace RG-6 and RG-59
[You may see RG-500 for heavy outdoor use (burial) - has a FAT core, 75 Ohms, super-long range]

**Fiber Optics:** *LX FX and S's can all do MMF - FX has full and half duplex (below)*

| | | | |
|---|---|---|---|
| 100Base-SX | 100Mbps | 200-550m | MMF (short wavelength laser) |
| 100Base-FX | 100Mbps | 2km full duplex, 400m half-duplex | MMF |
| 1000Base-LX | 1000Mbps | 10km SMF or 550m MMF | (both SMF and MMF) |
| 1000Base-LH | 1000Mbps | 10km | SMF |
| 1000Base-ZX | 1000Mbps | 70km | SMF |
| 10GBase-SR and -SW | 10 Gbps | 300m (short reach) | MMF |
| 10GBase-LW | 10 Gbps | 10km | SMF |
| 10GBase-LR | 10 Gbps | 25km (long reach) | SMF |
| 10GBase-ER and -EW | 10 Gbps | 40km (extended reach) | SMF |
| 100GBASE-ER4 | 100 Gbps | 40km | SMF |
| 100GBASE-SR10 | 100 Gbps | 100-125km | MMF |

Advantages of fiber: difficult to monitor (tap), no RF interference, support long distances
Multimode (MMF) refers that different light frequencies bouncing along a fiber bundle; often uses LEDs (less expensive), so signal degrades at a shorter distance. The core is typically 62.5 microns or larger in diameter
Singlemode (SMF) uses a single fiber with single frequency of light; more expensive- lasers to make the distance and increase signal strength. Typically uses a small light carrying core of 8 to 10 microns in diameter.

**Fiber Connectors:**


The SC connector as called a subscriber, standard, or square connector.
The MTRJ goes by two names, the "Media Termination Recommended Jack" and "Mechanical Transfer Registered Jack". It has two fiber strands (that is, a transmit strand and a receive strand) included in a single connector. Despite the second nickname, it doesn't lock into place or anything.
The LC (Lucent/Local/Little Connector) connects to a terminating device by pushing the connector into the terminating device with a "click" and can be removed by depressing the tab on the connector to pull it out.
A Straight Tip (ST) connector is sometimes referred to as a bayonet connector; most commonly used with multimode; connects by pushing the connector in and then twisting the connector housing to lock it in place.
- Multimode Delay Distortion - on multi-mode fiber-optic cables when an initial transmission can arrive at the receiver after a second transmission
- Mode of Propagation: the path that light takes through a fiber-optic cable.
- A multiplexer combines a number of signals into a single signal for transmission over the medium.

***Bulk Data Carrier Types***

**Bytes to bits in bandwidth conversions**
*"A megabit per second (abbreviated as Mbps, Mbit/s, or mbps) is a unit of data transfer rates equal to 1,000,000 bits per second (this equals 1,000 kilobits per second). Because there are 8 bits in a byte, a transfer speed of 8 megabits per second (8 Mbps) is equivalent to 1,000,000 bytes per second (approximately 976 KiB/s)"*
1 Kbps = 1000/8=125 bytes; 1 Mbps = 1,000,000/8=125,000 bytes
Bytes are made up of eight bits, so one kilobyte equals eight kilobits
1KB per sec = 8 Kbps; 1MB per sec = 8 Mbps

**T-carrier (T1- US/ Canada), and E-carrier (E1- Europe)**
Transmission System Level 1 - For trunking, time-division multiplexing. The T-carrier system (T1 and T3) refers to copper transport corresponding to DS1 and DS3. DS0 supported twenty 2.4 kbit/s channels, ten 4.8 kbit/s channels, five 9.67 kbit/s channels, one 56 kbit/s channel, or one 64 kbit/s clear channel, which is where DSL and ISDN came in. The base DS0 represents a single voice call digitized at 8 kHz sample rate w/ 8-bit pulse-code modulation at 8000 samples/sec which comes out to 64kbit/s

T1/E1/DS1 - 24 channels (DS0's)- 64 kbit/s per channel 1.5Mbit/s total line rate, 8 bits for framing info. E1 - 32 channels 2.048 Mbit/s line rate
T3/E3/DS3 - 28 T1 circuits - 672 T1 channels - 44.736 Mbit/s. E3 = 16 E1 circuits, 512 channels, 33.368 Mbit/s
In T1/E1 more than one frame is sent at once. Two methods to grouping these frames together:
Super Frame (SF) combines 12 standard 193-bit frames into one. Extended Super Frame (ESF) 24 193-bit frames into one.

**Data Capacity - The beer/soda can analogy:**
Base DS0- single voice call digitized at 8 kHz sample rate w/ 8-bit PCM @ 8000 samples/sec = 64kbit/s

| | | | | | |
|---|---|---|---|---|---|
| **DS0** | 56/64Kbps | 1 POTS line | Old modem | | One can |
| **DS1/T1** | 1.544 Mbps | 1.536 Mbps | | | A case of beer (24 cans) |
| **DS3/T3** | 44.736 Mbps | 28 DS1's | 672 DS0's | | A pallet of beer with 28 cases |
| **OC1** | 1 DS3/T3 | ***[End copper*** | ***and begin*** | ***SONET/SDH]*** | A shrink-wrapped pallet |
| **OC3** | 155.52 Mbps | 3 DS3's | 84 DS1's | 2016 DS0's | A truck that can hold 3 pallets |
| **OC12** | 622.08 Mbps | 12 DS3's | 336 DS1's | 8064 DS0's | A railroad train car - 12 pallets |
| **OC48** | 2488.32 Mbps | 4 OC12's | 48 DS3's | 1344 DS1's | 4 train cars x 12 pallets |
| **OC192** | 9953.28 Mbps | 16 OC12's | 192 DS3's | 5376 DS1's | 16 train cars - 192,024 DS0's |
| **OC768** | 39813.12 Mbps | 64 OC12's | 768 DS3's | 21,504 DS1's | Train with 64 railroad cars |

***Rough calculations for downloading a 1GB file (including ISDN and DSL)***

| **Connection** | **Speed** | **(Y:D:H:M:S)** | **Difference** |
|---|---|---|---|
| 56 K | 56,000 bps | 1:15:40:57 | 96% slower |
| 128 K | 128,000 bps | 17:21:40 | 91% slower |
| 256 K | 256,000 bps | 8:40:50 | 83% slower |
| 512 K | 512,000 bps | 4:20:25 | 66% slower |
| 768 K | 768,000 bps | 2:53:37 | 50% slower |
| T1, DS-1 | 1.544 Mbps | 1:26:21 | Baseline |
| T3, DS-3 | 44.736 Mbps | 2:59 | 2,798% faster |
| OC-3 | 155.520 Mbps | 51 | 9,973% faster |
| OC-12 | 622.080 Mbps | 13 | 40,191% faster |
| OC-48 | 2.488 Gbps | 3 | 161,040% faster |
| OC-192 | 10 Gbps | 1 | 647,569% faster |

**SONET - Synchronous Transport Signals (STS)**

Synchronous Optical networking (SONET) - ANSI - Synchronous Transport Signals (STS)

Synchronous Digital Hierarchy (SDH) - International equivalent - Synchronous Transport Modules (STM)

| SONET | SDH | Bandwidth | Overhead |
|---|---|---|---|
| STS-1/ OC-1 | STM-0 | 51.48Mbps | 1.728Mbps |
| STS-3/ OC-3 | STM-1 | 155.52Mbps | 6.912Mbps |
| STS-12/ OC-12 | STM-4 | 622.08Mbps | 20.736Mbps |
| 2.5G SONET/ STS-48/ OC-48 | STM-16 | 2.488Gbps | 82.944Mbps |
| 5G SONET/ STS-96/ OC-96 | STM-32 | 4.876Gbps | |
| 10G SONET/ STS-192/ OC-192 | STM-64 | 9.953Gbps | 442.368 Mbps |
| STS-256/ OC-256 | STM-128 | 13.271Gbps | |
| STS-768/ OC-768 | STM-256 | 39.813Gbps | 1.327104Gbps |

ISDN - Integrated Services Digital Network - Delivered over T1/E1

ISDN has two levels of service:

BRI - Basic Rate Interface (2B+D):

2 64kbit/s bearer/ user (B) channels (throughput)

1 16kbit/s signaling/delta (D) channel (connection maintenance)

PRI - Primary Rate Interface - hooked up directly to the telco central office

- 23B + 1D on a T1 (1.544 Mbps)
- 30B + 1D + sync/alarm channel on an E1 (2.048 Mbps)
- Commonly to deliver Public Switched Telephone Network (PSTN) to digital PBX (23 lines in one)
- Fewer active B channels can be used for a fractional T1.
- Can be used flexibly and reassigned when necessary (such as video conferencing)
- More channels can be used with more T1s, within certain design limits (PRI pairing)
- PRI pairing uses NFAS (non-facility associated signalling) to accommodate itself

*ISDN Terminology:*

An R reference point resides between a non-ISDN device to a terminal adapter.

An S/T reference point resides between a NT1 and a terminal endpoint 1 (TE1).

A TA (terminal adapter) performs conversion between a non-ISDN device and a TE1 device.

A U reference point resides between a NT1 and the wall jack connecting back to an ISDN service provider.

A NT1 (network termination 1) device interconnects a four-wire ISDN circuit and a two-wire ISDN circuit.