

CCNA

ICND-1 Command Reference

Somenath Sinha

Nov 2018

Contents

| | | |
|----------|--|----------|
| I | ICND1 - Switches | 6 |
| 1 | Basic Cisco Catalyst Switch Configuration | 7 |
| 1.1 | Port Addressing | 7 |
| 1.2 | Configuring a Management IP Address | 7 |
| 1.2.1 | View how we're connected to a Switch | 7 |
| 1.2.2 | Privileged mode, auto-completion and showing possible commands . | 8 |
| 1.2.3 | Configuring the Terminal | 8 |
| 1.2.4 | Interface Configuration Mode | 9 |
| 1.2.5 | Adding a Management IP | 9 |
| 1.3 | Configuring a Default Gateway | 9 |
| 1.4 | Setting Console and VTY Passwords | 9 |
| 1.4.1 | Configuring a password for the VTY lines | 10 |
| 1.5 | Show Running Configuration | 10 |
| 1.6 | Checking for connectivity using Ping | 11 |
| 1.7 | Enabling Telnet Access | 11 |
| 1.7.1 | Show line details | 12 |
| 1.7.2 | Logging in via telnet | 13 |
| 1.7.3 | Keyboard Shortcuts for Telnet | 13 |
| 1.7.4 | History Buffer | 13 |
| 1.7.5 | IP Interface Connection Overview | 14 |
| 1.8 | Enabling SSH Access | 14 |
| 1.8.1 | Setting up User-name, password and domain name | 15 |
| 1.8.2 | Connecting via SSH | 15 |
| 1.9 | Viewing Version Information | 16 |
| 1.10 | Viewing Current Configuration | 16 |

| | | |
|----------|---|-----------|
| 1.11 | MAC Address (CAM) Table Examination | 17 |
| 1.11.1 | Adding a Static MAC Address | 18 |
| 1.12 | Setting the Hostname | 18 |
| 1.13 | Setting the Enable Password | 19 |
| 1.14 | Setting the exec-timeout | 20 |
| 1.15 | Encrypting Passwords on Switches | 20 |
| 1.15.1 | User Privileges | 21 |
| 1.15.2 | Storing password hashes for login | 21 |
| 1.15.3 | Removing a user | 21 |
| 1.16 | Creating a Banner | 22 |
| 1.17 | Specifying Port Speed and Duplex | 23 |
| 1.17.1 | MDI-X (Medium Dependent Interface Crossover) | 23 |
| 1.18 | Saving the Configuration | 23 |
| 2 | Virtual LANs (VLANs) | 25 |
| 2.1 | Virtual LANs (VLANs) : Introduction | 25 |
| 2.2 | VLAN Theory | 25 |
| 2.2.1 | Packet Flow between VLANs | 26 |
| 2.3 | VLAN Creation | 27 |
| 2.3.1 | Show existing VLANs | 27 |
| 2.3.2 | Creating a new VLAN | 27 |
| 2.3.3 | Deleting a VLAN | 28 |
| 2.4 | Assigning Ports to a VLAN | 28 |
| 3 | Trunking | 29 |
| 3.1 | Trunking : Introduction | 29 |
| 3.2 | Trunking Theory | 29 |
| 3.2.1 | Marking Frames for VLANs | 30 |
| 3.2.2 | Native VLAN | 30 |
| 3.3 | Trunking Modes | 31 |
| 3.3.1 | Formation of a Trunk | 31 |
| 3.4 | Creating Trunks | 32 |
| 3.4.1 | Finding out the current trunk mode of a switch port | 32 |

| | | |
|----------|--|-----------|
| 3.4.2 | Showing currently existing Trunks | 33 |
| 3.4.3 | Changing Switch port's trunking settings | 33 |
| 3.4.4 | Trunk settings mismatch | 34 |
| 3.4.5 | Hard-coded Trunk mode | 35 |
| 3.5 | VLAN pruning | 36 |
| 3.5.1 | Showing VLANs allowed on a trunk | 36 |
| 3.5.2 | Allowing only certain VLANs over a certain trunk | 37 |
| 4 | Troubleshooting Switch Operation | 38 |
| 4.1 | Isolating the Issue | 38 |
| 4.1.1 | Pinging to test network connectivity | 38 |
| 4.1.2 | Extended Ping and Ping timeout due to ARP | 39 |
| 4.2 | Checking Interface Status | 40 |
| 4.2.1 | Show interfaces status | 41 |
| 4.3 | Checking for Interface Errors | 42 |
| 4.3.1 | Duplex Mismatch | 43 |
| 4.4 | Checking a Port's VLAN Membership | 44 |
| 4.5 | Checking a Trunk's Status | 44 |
| 5 | Basic Switch Security | 45 |
| 5.1 | Physical Security | 45 |
| 5.2 | Switch Port Security | 45 |
| 5.2.1 | MAC Flooding Attack | 45 |
| 5.2.2 | Port Security | 45 |
| 5.3 | Shutting down Unused Ports | 50 |
| 5.3.1 | Shutting down all ports on a switch | 50 |
| 5.4 | Putting Unused Ports in an Unused VLAN | 51 |
| 6 | Voice VLANs | 53 |
| 6.1 | Voice VLANs: Introduction | 53 |
| 6.2 | Voice VLAN Theory | 53 |
| 6.2.1 | Single VLAN Access port | 54 |
| 6.2.2 | Multi-VLAN Access port | 54 |
| 6.2.3 | Trunk port | 54 |

| | | |
|-----------|---|-----------|
| 6.3 | Voice VLAN Configuration | 55 |
| 6.3.1 | Setting up a Single VLAN access port | 55 |
| 6.3.2 | Setting up a Multi-VLAN access port | 55 |
| 6.3.3 | Setting up a Trunk Port for voice | 55 |
| II | ICND1 - Routers | 57 |
| 7 | Basic Router Operation | 58 |
| 7.1 | Packet flow through a Routed Network | 58 |
| 7.1.1 | Layer 3 Packet flow | 58 |
| 7.1.2 | Layer 2 Packet Flow | 59 |
| 7.2 | Sources of Route Information | 59 |
| 7.2.1 | IP Routing Table codes | 60 |
| 7.2.2 | Administrative Distance | 61 |
| 7.3 | IP Routing Table | 61 |
| 7.4 | Packet Forwarding | 62 |
| 7.4.1 | Process Switching | 62 |
| 7.4.2 | Fast Switching | 62 |
| 7.4.3 | Cisco Express Forwarding (CEF) | 63 |
| 8 | Basic Router Configuration and Verification | 64 |
| 8.1 | Router LEDs | 64 |
| 8.1.1 | GigabitEthernet port LED | 64 |
| 8.2 | Assigning an IPv4 Address to an interface | 65 |
| 8.3 | Assigning an IPv6 Address to an interface | 65 |
| 8.4 | Basic Router Configuration | 67 |
| 8.4.1 | Logging Synchronous | 68 |
| 8.4.2 | Running show commands from Configuration mode | 68 |
| 8.4.3 | Interface Description | 69 |
| 9 | Routing Fundamentals | 70 |
| 9.1 | Configuring Static Routes for IPv4 | 70 |
| 9.1.1 | Static Network Route Configuration | 70 |
| 9.1.2 | Configuring a Static Host Route | 72 |

| | | |
|-----------|--|-----------|
| 9.1.3 | Keeping a route permanently in the Routing table | 74 |
| 9.1.4 | Static Default Route Configuration | 74 |
| 9.1.5 | Floating Static Route | 75 |
| 9.2 | Configuring Static Routes for IPv6 | 76 |
| 9.2.1 | IPv6 Static Network Routes | 77 |
| 9.2.2 | IPv6 Static Host Route | 78 |
| 9.2.3 | IPv6 Static Default Route | 79 |
| 9.2.4 | IPv6 Floating Static Route | 79 |
| 9.3 | Overview of Routing Protocols | 79 |
| 9.3.1 | Scalability | 79 |
| 9.3.2 | Vendor Interoperability and Familiarity | 80 |
| 9.3.3 | Convergence time | 80 |
| 9.3.4 | Summarization | 80 |
| 9.3.5 | Interior or Exterior Gateway Protocol | 81 |
| 9.3.6 | Protocol Classification | 81 |
| 9.4 | Administrative Distance | 82 |
| 9.5 | Split Horizon | 82 |
| 9.6 | Metric | 83 |
| 9.7 | Next Hop Address | 83 |
| 9.7.1 | Using ARP to get the MAC address | 84 |
| 9.8 | Passive Interfaces | 84 |
| 9.8.1 | Network Command and Wildcard Mask | 84 |
| 9.8.2 | Need for Passive Interfaces | 85 |
| 10 | Routing Information Protocol (RIP) | 86 |

Part I

ICND1 - Switches

Chapter 1

Basic Cisco Catalyst Switch Configuration

1.1 Port Addressing

The addressing of ports on a Cisco Catalyst Switch follows the format:

```
interface-type Stack #/Card #/Port #
```

Let us consider the **Catalyst 3750** switch. This switch isn't modular and thus can't have multiple cards, but it supports Stack-wise, which is a Cisco technology to combine multiple physical switches in a stack into a single logical switch. Thus, the middle number when referring to a particular interface (i.e., port on a physical switch) will **always be 0**. Thus, the Giga-bit port that's 16th port on the 3rd stack member of switches will be written as: GigabitEthernet 3/0/16, or **g3/0/16** for short.

For a **Cisco 2960** switch, there's no option for Stack-wise, and hence we don't need the first number. Then, the name for the interface becomes GigabitEthernet 0/16 or **g0/16**.

1.2 Configuring a Management IP Address

We can connect to the switch physically via the console port, or through one of the interfaces using Ethernet. In case of the later, we have to set up a management IP address through which we can use telnet or SSH to control and configure the switch.

1.2.1 View how we're connected to a Switch

To view how we're presently connected to a switch, we can use the `show line` command:

```
1  sw1>show line
2      Tty Typ      Tx/Rx      A Modem  Roty Acc0 AccI   Uses   Noise  Overruns  Int
3  *    0 CTY          - -          - -      - -    0      0      0/0      -
4      1 AUX    9600/9600    - -          - -      - -    0      0      0/0      -
5      2 VTY          - -          - -      - -    1      0      0/0      -
6      3 VTY          - -          - -      - -    0      0      0/0      -
7      4 VTY          - -          - -      - -    0      0      0/0      -
```


| | | | | | | | | | | |
|---|-------|---|---|---|---|---|---|---|-----|---|
| 8 | 5 VTY | - | - | - | - | - | 0 | 0 | 0/0 | - |
| 9 | 6 VTY | - | - | - | - | - | 0 | 0 | 0/0 | - |

The **asterisk (*)** at the beginning of the line shows that it's the port currently being used. The **CTY** refers to a console port, which is the port we're using. If we were using telnet or SSH, i.e., log in remotely to the switch/router, then one of the **VTY** (*Virtual Terminal*) would've been used.

1.2.2 Privileged mode, auto-completion and showing possible commands

The > sign at the end of the prompt signals that we're currently in *user mode*. To change the configuration on the equipment we need to be in *Privileged mode*, which is indicated by a # prompt. To enter the privileged mode, we use the command `enable`. To leave the privileged mode, we use `disable`:

```
1 sw1>enable
2 sw1#disable
3 sw1>
```

To go into privileged mode, we need not type the entire command. Cisco **IOS** (*Internetwork Operating System*) can show all possible commands that can follow the current string on the present line on pressing the ? key. For example:

```
1 sw1>e?
2 enable ethernet exit
3
4 sw1>en?
5 enable
```

This also means that we don't need to type out the entire command. We can simply use short-cuts for the commands, which are the minimum number of letters in the word of a command that is unique. Thus, in the case of the `enable`, the minimum characters required are 2 since there are 3 commands starting with 'e' but only 1 starting with 'en', i.e., `enable`.

```
1 sw1>en
2 sw1#
```

1.2.3 Configuring the Terminal

While we may be in privileged mode, we still need to tell the switch that we want to configure from the terminal. For this, we use the command `configure terminal`, or **conf t** for short. This would bring us into the **global config** mode:

```
1 sw1#conf t
2 Enter configuration commands, one per line. End with CNTL/Z.
3 sw1(config)#
```

To exit the global configuration mode, we use the `exit` command:

```
1 sw1(config)#exit
2 sw1#
3 *Nov 11 21:36:07.353: %SYS-5-CONFIG_I: Configured from console by console
4 sw1#
```

1.2.4 Interface Configuration Mode

To set the management IP address we need to go to the interface configuration port. On a Cisco Switch, unlike on a PC/Laptop, every interface gets associated to the IP address which is of the network it's connected to. Layer-2 Switches are generally blind to the IP addresses since they don't make forwarding decisions on the basis of IP addresses, but this IP Address is required to be associated with the switch to uniquely identify the switch on the network for remote access through telnet/SSH.

By default, on the switch, all the ports belong to a specific **VLAN(Virtual LAN)**. Thus, we can set the IP address of the *default VLAN*, **vlan1**, to set up the management IP address of the switch. To enter the interface configuration for vlan1, we use: `interface vlan1`, or simply **int vlan1** :

```
1 sw1(config)#int vlan 1
2 sw1(config-if)#
```

The prompt changed to `(config-if)#` to show that we've now entered the *Interface config mode*.

1.2.5 Adding a Management IP

Just like with a PC, we need to set up both the IP address as well as the Subnet Mask. Let us consider the switch needs to have the IP **192.168.1.11/24**. We can't enter the value in suffix notation. The equivalent subnet mask is `255.255.255.0`. Thus, the command becomes:

```
1 sw1(config-if)#ip add 192.168.1.11 255.255.255.0
2 sw1(config-if)#no shutdown
3 sw1(config-if)#
4 *Nov 11 21:51:57.295: %LINK-3-UPDOWN: Interface Vlan1, changed state to up
5 *Nov 11 21:51:58.298: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan1, changed
  ↔ state to up
6 sw1(config-if)#
```

Typically the interface `vlan1` (i.e, the 1st virtual LAN) is turned on by default, but sometimes it may have been administratively shut-down, which we can fix using `no shutdown` command, or for short **no shut**.

1.3 Configuring a Default Gateway

A default gateway is required for the switch to determine which path to take to exit it's own subnet and reach another network. While the IP address has to be set for an interface, the default gateway has to be set in global config mode:

```
1 sw1(config)#ip default-gateway 192.168.1.1
```

1.4 Setting Console and VTY Passwords

To set the password for the console port login, we need to enter its configuration, i.e., console line configuration, and then set the password. Let's say we want to set it to *cisco*. We can do this by:

```
1 sw1(config-line)#password cisco
2 sw1(config-line)#login
```

The `login` command enables the password check at login. Unless it's set, even if there's a valid password, the system won't ask for credentials while logging in. Further, to disable password check, we don't have to change/remove the password. We simply need to do `no login`. The `login` command doesn't work unless there's a password set:

```
1 sw1(config-line)#login
2 % Login disabled on line 0, until 'password' is set
```

1.4.1 Configuring a password for the VTY lines

We could set the password for each VTY line individually, but if we're setting them (or a subset of them) all to an identical config, we can use:

```
1 sw1(config)#line vty 0 5
2 sw1(config-line)#password cisco
3 sw1(config-line)#login
4 sw1(config-line)#end
5 sw1#
```

This is assuming we want to set the password for lines 0-5 and not 6 of the available VTY lines. Also notice that we jump levels till we end the configuration of the switch by using `end` command. Unlike the `exit` command, the `end` command doesn't just jump up to the parent config level, but ends the configuration all together and takes us out of the configuration mode all together.

1.5 Show Running Configuration

To see the current configuration we created (but not yet saved), i.e., the configuration that the device is currently using, we use the command:

```
1 sw1#sh run
2 Building configuration...
3
4 Current configuration : 2911 bytes
5 !
6 ! Last configuration change at 22:48:16 UTC Sun Nov 11 2018
7 !
8 version 15.2
9 service timestamps debug datetime msec
10 service timestamps log datetime msec
11 no service password-encryption
12 service compress-config
13 !
14 hostname sw1
15 ...
16 !
17 interface Vlan1
18 ip address 192.168.1.11 255.255.255.0
```

```
19  !
20  ip default-gateway 192.168.1.1
21  ip forward-protocol nd
22  !
23  ...
24  !
25  line con 0
26    exec-timeout 0 0
27    password cisco
28    login
29  line aux 0
30  line vty 0 4
31    password cisco
32    login
33  line vty 5
34    password cisco
35    login
36  line vty 6
37    login
38  !
39  !
40  end
41
42  sw1#
```

At the very bottom we have the details for the console and the VTY lines. It's completely normal for IOS to break down the configuration for 0-6 into ranges (0-4,5,6), even though we configured them together at the same time.

1.6 Checking for connectivity using Ping

The ping command sends **ICMP (*Internet Control Message Protocol*)** packets to the destination device and waits for *ICMP replies* from it to determine whether the host is up. It can help detect delays and packet drops, and whether the host is even connected i.e., reachable. We can use it by:

```
1  PC-1> ping 192.168.1.11
2  84 bytes from 192.168.1.11 icmp_seq=1 ttl=255 time=3.973 ms
3  84 bytes from 192.168.1.11 icmp_seq=2 ttl=255 time=3.969 ms
4  84 bytes from 192.168.1.11 icmp_seq=3 ttl=255 time=5.951 ms
5  84 bytes from 192.168.1.11 icmp_seq=4 ttl=255 time=3.472 ms
6  84 bytes from 192.168.1.11 icmp_seq=5 ttl=255 time=10.913 ms
```

Some variation of the ping command is available on all major OS (Operating Systems).

1.7 Enabling Telnet Access

Telnet is considered a non-secure method of accessing network equipment since data is sent in plain text and packet capture utils like wireshark can capture the traffic and get access to all the data. Thus **SSH (*Secure Shell*)** is considered a much better alternative.

1.7.1 Show line details

To see the details of the VTY/CTY line, we use:

```
1 sw1#sh line vty 0
2   Tty Typ      Tx/Rx      A Modem  Roty Acc0 AccI   Uses   Noise  Overruns  Int
3       2 VTY          -    -        -    -    -       1       0      0/0      -
4
5 Line 2, Location: "", Type: ""
6 Length: 24 lines, Width: 80 columns
7 Baud rate (TX/RX) is 9600/9600
8 Status: Ready, No Exit Banner
9 Capabilities: none
10 Modem state: Ready
11 Group codes:      0
12 Special Chars: Escape Hold Stop Start Disconnect Activation
13                ^~x   none  -    -          none
14 Timeouts:         Idle EXEC   Idle Session  Modem Answer  Session  Dispatch
15                00:10:00      never          none      not set
16
17                Idle Session Disconnect Warning
18                never
19                Login-sequence User Response
20                00:00:30
21                Autoselect Initial Wait
22                not set
23
24 Modem type is unknown.
25 Session limit is not set.
26 Time since activation: never
27 Editing is enabled.
28 History is enabled, history size is 20.
29 DNS resolution in show commands is enabled
30 Full user help is disabled
31 Allowed input transports are lat pad telnet rlogin nasi ssh.
32 Allowed output transports are lat pad telnet rlogin nasi ssh.
33 Preferred transport is lat.
34 Shell: enabled
35 Shell trace: off
36 No output characters are padded
37 No special data dispatching characters
```

The two lines of special interest here are:

```
1 Allowed input transports are lat pad telnet rlogin nasi ssh.
2 Allowed output transports are lat pad telnet rlogin nasi ssh.
```

The **Input Transport** refer to the methods via which we're allowed to login to the router whereas the *output transports* are the methods we can use to login to another device from the router. To see all of the transports methods available we use:

```
1 sw1#conf t
2 Enter configuration commands, one per line.  End with CNTL/Z.
3 sw1(config)#line vty 0 6
4
5 sw1(config-line)#transport input ?
6   all      All protocols
7   lat      DEC LAT protocol
8   nasi     NASI protocol
9   none     No protocols
```

```

10 pad      X.3 PAD
11 rlogin   Unix rlogin protocol
12 ssh      TCP/IP SSH protocol
13 telnet   TCP/IP Telnet protocol
14 udptn    UDPTN async via UDP protocol

```

To only allow telnet and SSH, we use:

```

1 sw1(config-line)#transport input telnet ssh
2 sw1(config-line)#end
3 sw1#sh line vty 0
4 ...
5 Allowed input transports are telnet ssh.
6 Allowed output transports are lat pad telnet rlogin nasi ssh.
7 ...

```

1.7.2 Logging in via telnet

To login via telnet from another device, and see which vty we're using, we use the command:

```

1 sw2>telnet 192.168.1.11
2 Trying 192.168.1.11 ... Open
3 ...
4 sw1>sh line
5      Tty Typ      Tx/Rx      A Modem  Roty Acc0 AccI  Uses   Noise  Overruns  Int
6 *    0 CTY          - -          - -    -    0      0      0/0    -
7      1 AUX  9600/9600  - -          - -    -    0      0      0/0    -
8 *    2 VTY          - -          - -    -    2      0      0/0    -
9      3 VTY          - -          - -    -    0      0      0/0    -
10     4 VTY          - -          - -    -    0      0      0/0    -
11 ...
12 sw1>

```

Here we can see we are using VTY 2 for the telnet.

1.7.3 Keyboard Shortcuts for Telnet

| Short cut | Action |
|-----------------|---|
| Ctrl + A | Jump to the start of the line |
| Ctrl + E | Jump to the end of the line |
| Ctrl + W | Delete previous word in the line |
| Ctrl + X | Delete all characters before cursor. |
| Ctrl + K | Delete all characters from cursor till end. |
| ↑ Arrow | Delete all characters before cursor. |
| ↓ Arrow | Delete all characters before cursor. |
| Esc + B | Move cursor back 1 word. |
| Esc + F | Move cursor forward 1 word. |

1.7.4 History Buffer

The history buffer contains the last 20 commands used by default. This can be viewed with:

```

1  sw1#show history
2      sh line
3      conf t
4      en
5      conf t
6      sh run
7      sh ver | i up
8      sh line status
9      sh line
10     sh controllers gigabitEthernet 0/0
11     sh cont g 0/0 tab
12     sh cont g 0/0 tabular
13     sh diag
14     sh line
15     sh line vty 0
16     conf t
17     sh line vty 0
18     sh ip int br
19     sh run
20     sh mem
21     sh history
22  sw1#

```

The size of the history buffer can be changed in the line configuration mode with:

```

1  sw1#conf t
2  Enter configuration commands, one per line.  End with CNTL/Z.
3  sw1(config)#line vty 0 6
4  sw1(config-line)#history size ?
5      <0-256>  Size of history buffer
6
7  sw1(config-line)#history size 50

```

Now the history buffer will store the last 50 commands.

1.7.5 IP Interface Connection Overview

The `show ip interface brief` command shows us the connection status of the interface, the IP address assigned to each interface and more.

```

1  sw1#sh ip int brief
2  Interface                IP-Address      OK? Method Status          Protocol
3  GigabitEthernet0/0       unassigned      YES unset  up              up
4  GigabitEthernet0/1       unassigned      YES unset  up              up
5  GigabitEthernet0/2       unassigned      YES unset  up              up
6  GigabitEthernet0/3       unassigned      YES unset  up              up
7  Vlan1                    192.168.1.11    YES manual up              up

```

1.8 Enabling SSH Access

Secure SHell (SSH), unlike Telnet requires both a user-name and a password for authentication. This user-name-password can live either on an external authentication server, or locally, i.e., in the memory of the router itself. In addition, the router also needs to know which domain it's a part of.

1.8.1 Setting up User-name, password and domain name

The username and password commands take care of creating an username and password, and can be used together on the same line. The domain name is set using `ip domain-name` command. These have to be set in the global config mode:

```
1 sw1(config)#username cisco password cisco
2 sw1(config)#ip domain-name somuvmmnet.com
```

The domain name is used in the creation and self-signing of the digital certificate. A public-private key pair is used to enforce authentication. They can be generated using the `crypto key generate rsa`. Note that the mod size has to be ≥ 768 bits for SSH 2.

```
1 sw1(config)#crypto key generate rsa
2 The name for the keys will be: sw1.somuvmmnet.com
3 Choose the size of the key modulus in the range of 360 to 4096 for your
4   General Purpose Keys. Choosing a key modulus greater than 512 may take
5   a few minutes.
6
7 How many bits in the modulus [512]: 1024
8 % Generating 1024 bit RSA keys, keys will be non-exportable...
9 [OK] (elapsed time was 0 seconds)
10
11 sw1(config)#
12 *Nov 12 01:57:14.624: %SSH-5-ENABLED: SSH 1.99 has been enabled
13 sw1(config)#
```

Now we have to choose to use SSH version 2, using:

```
1 sw1(config)#ip ssh version 2
```

Finally we need to tell the device where to find the username and password combination. In this case, it's going to be the device memory itself. Hence, we use the `login local` command in the *line config mode*.

```
1 sw1(config)#line vty 0 6
2 sw1(config-line)#login local
```

1.8.2 Connecting via SSH

Depending on the OS, just like the `ping` command, the input options may vary for the `ssh` command, but the premise remains the same. The following is the method to do this on an IOS terminal:

```
1 sw2#ssh -l cisco 192.168.1.11
2 ...
3 Password:
4 ...
5 sw1>exit
6
7 [Connection to 192.168.1.11 closed by foreign host]
8 sw2#
```

1.9 Viewing Version Information

The `show version` or `sh ver` command shows us some critical information about the device itself. This includes the version of the IOS that's being used by the hardware - a critical piece of information when reporting issues to Cisco's TAC (Technical Assistance Centre).

```
1 sw2#sh ver
2 Cisco IOS Software, ... Version 15.2(20170321:233949)
3 ...
4 sw2 uptime is 6 hours, 33 minutes
5 ...
6 Cisco IOSv () processor (revision 1.0) with 984321K/62464K bytes of memory.
7 ...
8 1 Virtual Ethernet interface
9 4 Gigabit Ethernet interfaces
10 ...
11
12 sw2#
```

The command also shows the uptime of the switch, i.e., how long it's been running. Further it shows the total amount of RAM, which in this case is $984321 + 62464$ KB of memory, i.e., $= 1046785\text{KB} \approx 1\text{GB}$ of RAM. The number and types of interfaces is also displayed.

1.10 Viewing Current Configuration

To show the entirety of the running configuration, we use the `show running-configuration` command, or `sh run` for short. There's also the option of seeing specific lines in the configuration by keyword searching using output filters.

```
1 sw1#sh run
2 Building configuration...
3
4 Current configuration : 3169 bytes
5 !
6 ! Last configuration change at 02:12:13 UTC Mon Nov 12 2018
7 !
8 version 15.2
9 ...
10 hostname sw1
11 ...
12 !
13 username cisco password 0 cisco
14 ...
15 !
16
17 sw1#sh run | i username
18 username cisco password 0 cisco
19 sw1#
```

The second command above uses the `i|` filter that only shows lines including the matching keyword. There are several more criteria, some of which are:

| Short cut | Action |
|---------------|---|
| <command> i | Only show lines that include the keyword |
| <command> e | Only show lines that exclude the keyword |
| <command> b | Begin showing all lines from the first instance of the keyword |
| <command> s | Only show the section containing the keyword |
| <command> c | Count the number of lines matching the RegExp |

Just like a normal pager in Linux, the output of the show run command is searchable in the format /<keyword> where upon typing the *keyword* after the / character and pressing enter, the screen jumps to the first line containing the keyword.

```

1  sw1#sh run
2  Building configuration...
3  ...
4  username cisco password 0 cisco
5  no aaa new-model
6  !
7  /vty
8  filtering...
9  line vty 0 4
10 password cisco
11 login local
12 history size 50
13 transport input telnet ssh
14 line vty 5
15 password cisco
16 login local
17 history size 50
18 transport input telnet ssh
19 !
20 end
21
22 sw1#

```

1.11 MAC Address (CAM) Table Examination

The primary objective of a Layer 2 Switch is to learn which MAC addresses live off of which port and storing them in a table called Content Addressable Memory (CAM) or MAC Address table.

Then, when it gets a packet destined for a certain MAC Address, it sends it along the corresponding port it looks up from the CAM table. If the address is unknown, it floods all the ports but the port from which the packet was received, in an effort to learn which port is associated with that MAC address and store it in the table, to aid in future lookups.

The contents of the MAC address can be viewed with the show mac address-table command:

```

1  sw1#sh mac address-table
2          Mac Address Table
3  -----
4
5  Vlan    Mac Address      Type      Ports
6  ----    -
7      1    0cee.bab5.5f00   DYNAMIC   Gi0/0

```

```

8      1      0cee.bacd.5000      DYNAMIC      Gi0/1
9      1      0cee.bacd.5001      DYNAMIC      Gi0/0
10 Total Mac Addresses for this criterion: 3

```

The reason there are multiple MAC addresses associated to a certain port is because that MAC address belongs to another switch through which multiple devices can be reachable. Since that switch connects us to those MAC addresses, i.e., is a direct path, the end-user MAC addresses are mapped to the port leading up to the switch.

While our MAC addresses can be obtained beyond switches, they can't be obtained for devices beyond a router. The dynamic entries in the MAC address table can be aged-out/timed-out if no traffic was seen from that port in a certain time (by default 5mins). To see the ageing time, we use:

```

1 sw1#show mac address-table aging-time
2 Global Aging Time: 300
3 Vlan      Aging Time
4 ----      -

```

There can also be statically mapped MAC addresses that aren't learned but hard-coded into the switch.

1.11.1 Adding a Static MAC Address

We can add a static MAC Address mapping using:

```

1 sw1(config)#mac address-table static a820.6332.0087 vlan 1 interface gi 0/3
2 sw1(config)#end
3 sw1#sh mac address-table
4           Mac Address Table
5 -----
6
7 Vlan      Mac Address      Type      Ports
8 ----      -
9      1      0cee.bab5.5f00    DYNAMIC    Gi0/0
10     1      0cee.bacd.5000    DYNAMIC    Gi0/1
11     1      0cee.bacd.5001    DYNAMIC    Gi0/0
12     1      a820.6332.0087    STATIC     Gi0/3
13 Total Mac Addresses for this criterion: 4
14 sw1#

```

This can be handy when we're troubleshooting and trying to ensure that the MAC address of a device has been learned off of the right port.

1.12 Setting the Hostname

The hostname of a network device acts as the name of the device in the network, helping us easily identify the device. If we have multiple switches in a network, as is often the case, the hostname lets us quickly identify and distinguish the devices. We can change the hostname using:

```

1 sw(config)#hostname sw1
2 sw1(config)#end

```

The current hostname is displayed as the first word before the prompt.

1.13 Setting the Enable Password

Since the `Enable` command lets us configure the switch, we should set a password on it. We can do this using the `enable password` command:

```
1 sw1(config)#enable password cisco
2 sw1(config)#end
3 sw1#
4 *Nov 19 05:37:32.191: %SYS-5-CONFIG_I: Configured from console by console
5 sw1#disable
6 sw1>enable
7 Password:
8 sw1#
```

This creates a password that's stored as clear-text in the running config. Anyone who can view the running config will be able to see it:

```
1 sw1#sh run | i cisco
2 enable password cisco
```

To change this, we can store an encrypted password hash, using the `enable secret` command. This has several options in terms of encryption algorithms:

```
1 sw1(config)#enable secret ?
2   0       Specifies an UNENCRYPTED password will follow
3   5       Specifies a MD5 HASHED secret will follow
4   8       Specifies a PBKDF2 HASHED secret will follow
5   9       Specifies a SCRYPT HASHED secret will follow
6   LINE    The UNENCRYPTED (cleartext) 'enable' secret
7   level   Set exec level password
```

For example, the MD5 (Message Digest 5) algorithm when applied on a word will give us a 128-bit hash value, also called a *digest*. For example, the MD5 digest of 'cat' is 'D077F244DEF8A70E5EA758BD8352FCD8'. When someone enters the password, the switch runs the MD5 hashing algorithm on it to get its hash. If the digest matches that already stored in the memory for the password, then the user is granted access. There's no way to get the password directly from the hash, since the hash is like a fingerprint of the password instead of a scrambled up version of the password itself. We can use the `enable secret` command (which uses MD5 as the default algorithm) as:

```
1 sw1(config)#enable secret somu
2 sw1(config)#end
3 sw1#sh run | i enable
4 enable secret 5 $1$/Rxb$NofwEzG6FNec2v9TiU1VA1
5 enable password cisco
```

The 5 before the hash `1/Rxb$NofwEzG6FNec2v9TiU1VA1` tells us that it's a MD5 digest. There are other algorithms available as well:

| Type | Algorithm | Description |
|------|-----------|---|
| 5 | MD5 | Default algorithm; produces 128bit hash; lesser security than the hashes below. |
| 8 | PBKDF2 | Uses a variant of Password Based Key Derivation Function 2 (PBKDF2) , that itself uses SHA-256 (Secure Hashing Algorithm - 256) to generate a 256-bit hash; more secure than MD5. |
| 9 | SCRYPT | Pronounced "S-Crypt"; Like PBKDF2, but requires even more computational resources to crack. |

To change the algorithm type while setting the secret we can use `enable algorithm-type` :

```

1  sw1(config)#enable algorithm-type ?
2      md5          Encode the password using the MD5 algorithm
3      scrypt       Encode the password using the SCRYPT hashing algorithm
4      sha256       Encode the password using the PBKDF2 hashing algorithm
5
6  sw1(config)#enable algorithm-type sha256 secret cisco
7  sw1(config)#end
8  sw1#sh run | i enable
9  enable secret 8 $8$srB.hFoYl8uTbI$.h2H8Ux2Ie4F/qDgysbS43/RWT1aapN4rlNIA3/0YUc

```

1.14 Setting the exec-timeout

The default behaviour of the switch is to log out the user after a certain period of inactivity, so that no one gains access if an admin forgets to logout. This can be defined through the `exec-timeout` command, followed by the number of minutes and seconds before time-out:

```

1  sw1#conf t
2  Enter configuration commands, one per line.  End with CNTL/Z.
3  sw1(config)#line con 0
4  sw1(config-line)#exec-timeout 5 30

```

The above sets the execution time-out period to 5m 30s. To remove it, we can use either `no exec-timeout` or `exec-timeout 0 0`.

1.15 Encrypting Passwords on Switches

If we do a `show run` after password command, the password for logging in will be stored in clear text just like in the case of `enable password` command.

```

1  sw1#sh run | s line con
2  line con 0
3      exec-timeout 0 0
4      password cisco
5      login
6  sw1#

```

To store an encrypted version of the password (i.e., **not** a hash), we can use *password encryption service*. We do this using the `service password-encryption` command:

```
1 sw1(config)#service password-encryption
2 sw1(config)#end
3 sw1#sh run | s line con
4 line con 0
5   exec-timeout 0 0
6   password 7 070C285F4D06
7   login
8 sw1#
```

1.15.1 User Privileges

Cisco's IOS allows us to set granular access to commands using the privilege levels associated with a username. There can typically be 0-15 privilege levels. However, we typically only use two values: **0: Normal user** and **15: Administrator**. We can create a new user with a set privilege level using:

```
1 sw1(config)#username somu privilege 15 password cisco
2 sw1(config)#end
3 sw1#sh run | i somu
4 username somu privilege 15 password 0 cisco
5 sw1#
```

Again, if we want the password to be encrypted, we use:

```
1 sw1(config)#serv password-encryption
2 sw1(config)#username somu privilege 15 password cisco
3 sw1(config)#end
4 sw1#sh run | i somu
5 username somu privilege 15 password 7 05080F1C2243
```

These use level 7 encryption, which is very easy to decrypt. Hence, a better method is to use the `secret` command which uses hashing just like in the case of the `enable secret` command.

1.15.2 Storing password hashes for login

If we want to use an encrypted password for login or change the algorithm type used for login, we need to have a username. The command allows us to change the algorithm type as well as set an encrypted password by:

```
1 sw1(config)#username somu algorithm-type scrypt secret cisco
2 sw1(config)#end
3 sw1#sh run | i somu
4 username somu secret 9 $9$JFpvgWWbMj8jr2$cK3k0VWA1xMpGD5Ky6Vj4URxXGgtX4jkyML86D4vuuQ
```

1.15.3 Removing a user

To remove a user, we simply need to use the `no username` command:

```

1 sw1(config)#no username somu
2 This operation will remove all username related configurations with same name.Do you want
  ↵ to continue? [confirm]
3 sw1(config)#

```

1.16 Creating a Banner

A **banner** is a section of text displayed during a specific action, such as during login. There can be several types of banners:

```

1 sw1(config)#banner ?
2  LINE          c banner-text c, where 'c' is a delimiting character
3  config-save   Set message for saving configuration
4  exec          Set EXEC process creation banner
5  incoming      Set incoming terminal line banner
6  login         Set login banner
7  motd          Set Message of the Day banner
8  prompt-timeout Set Message for login authentication timeout
9  slip-ppp      Set Message for SLIP/PPP

```

The banner is added by using the **banner** command, followed by the type of banner (shown in the list above) and finally, the **delimiter**. This delimiter signals the start and end of the banner to IOS and thus, has to be a character that doesn't appear in the banner. We can set the banner with:

```

1 sw1(config)#banner login `
2 Enter TEXT message. End with the character '`'.
3 +-----+
4 | sw1 :      [ Swtich 1 ] +----->
5 | Authorized Access Only! +----->
6 +-----+
7 `
8 sw1(config)#end
9 sw1#sh run | b banner login
10 banner login ^C
11 +-----+
12 | sw1 :      [ Swtich 1 ] +----->
13 | Authorized Access Only! +----->
14 +-----+
15 ^C
16 !

```

During the login sessions, the banner will look like:

```

1 sw1 con0 is now available
2 Press RETURN to get started.
3
4 +-----+
5 | sw1 :      [ Swtich 1 ] +----->
6 | Authorized Access Only! +----->
7 +-----+
8
9 User Access Verification
10 Password:

```

1.17 Specifying Port Speed and Duplex

The speed and duplex settings are critical to ensure that data is sent and received properly, and are set per interface. Depending on the interface, the speed is used to set the transmit (Tx) and Receive (Rx) rates. The typical speed settings available are:

```
1 sw1(config)#int g 0/1
2 sw1(config-if)#speed ?
3     10      Force 10 Mbps operation
4     100     Force 100 Mbps operation
5     1000    Force 1000 Mbps operation
6     auto    Enable AUTO speed configuration
```

Duplex determines whether data can be sent and received at the same time. **Full-duplex** means that data can be both sent and received at the same time, while **half-duplex** means data can only be sent or received at a time. The half-duplex mode allows **CSMA/CD (Carrier Sense Multiple Access/Collision Detection)** to work, which isn't required in full-duplex.

```
1 sw1(config-if)#duplex ?
2     auto    Enable AUTO duplex configuration
3     full    Force full duplex operation
4     half    Force half-duplex operation
```

Note that both speed and duplex settings can be set to auto-negotiate. This means the two ends of the link automatically determine the best settings. However, people may choose to hard-code the speed and duplex settings in case auto-negotiation fails. Typically, if we administer devices on both sides of the link, then we choose to hard-code the settings. However, if the end user device such as a laptop is connected, then we may use auto since we don't know what settings these devices are configured with.

1.17.1 MDI-X (Medium Dependent Interface Crossover)

If we consider an RJ45 connector, the pins 1,2,3 & 5 are used for Ethernet. Here, two wires each are used for transmit and receive. In case we connected two switches using a straight-through Ethernet cable, the transmit wires on both sides will be connected, instead of the transmit wires on one side connecting to the receive wires on the other, leading to a failure of communication.

If we enable **MDI-X** it allows the switch to auto-detect which wires should be used for transmit and which for receive. To turn it on, we use the `mdix auto` command. This only is available, of course, only if the switch itself supports it.

1.18 Saving the Configuration

The running configuration isn't stored in a non-volatile memory, i.e., the changes don't survive a power-cycle. To make the changes permanent, we need to copy them to the NVRAM, by saving them as the **startup-configuration**. For this we use the command : `copy run star:`

```
1 sw1#copy running-config startup-config
2 Destination filename [startup-config]?
3 Building configuration...
```

```
4 Compressed configuration from 2781 bytes to 1574 bytes[OK]
5 *Nov 19 08:21:24.796: %GRUB-5-CONFIG_WRITING: GRUB configuration is being updated on
↪ disk. Please wait...
6 *Nov 19 08:21:25.631: %GRUB-5-CONFIG_WRITTEN: GRUB configuration was written to disk
↪ successfully.
7 sw1#
```

In older versions of IOS, there was a `write memory` command that did the same thing. We may still be able to use the `write mem` command, (`wr` for short):

```
1 sw1#wr
2 Building configuration...
3 Compressed configuration from 2781 bytes to 1574 bytes[OK]
4 sw1#
5 *Nov 19 08:25:12.887: %GRUB-5-CONFIG_WRITING: GRUB configuration is being updated on
↪ disk. Please wait...
6 *Nov 19 08:25:13.693: %GRUB-5-CONFIG_WRITTEN: GRUB configuration was written to disk
↪ successfully.
7 sw1#
```

Cisco however, may get rid of the `wr` command in the future, and hence it's suggested to use the `copy run star` command.

Chapter 2

Virtual LANs (VLANs)

2.1 Virtual LANs (VLANs) : Introduction

When interconnecting devices on a switch, we may want some of those devices to be on different logical networks or subnets. This is done by assigning them to a different **VLAN** (*Virtual Local Area Network*).

For example, in an office building, there may be a dedicated VLAN for each department, i.e., a VLAN for the accounts team and one for the sales team so that the data for one department doesn't flow through the network of another team. We can assign different switch-ports to different VLANs. When we want traffic to flow between those VLANs, a router has to be used to route traffic from one subnet to another.

2.2 VLAN Theory

Let us consider a case where we have two floors in an office which has 2 departments: Sales and Accounting. Since we don't want traffic to intermix, we need separate switches on each floor on each department. Given that we have 2 floors with 2 department each, the total number of switches is 4. For a bigger office, with more floors and department each, we'd need many more devices.

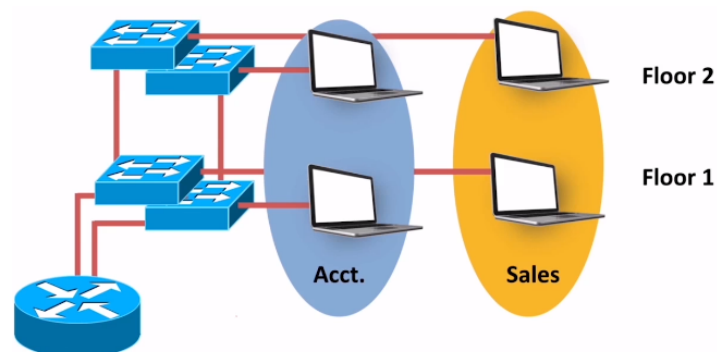


Figure 2.1: Without VLANs

Further, for a new department to be added, a new switch has to be installed on each floor for that department. A solution to this problem is given by the introduction of VLANs. While

the devices are connected to the same physical network, they devices themselves are connected to different logical networks. Thus, instantly the total number of switches required falls dramatically.

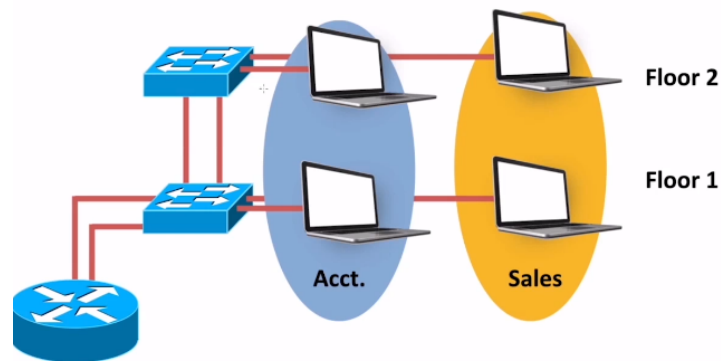


Figure 2.2: With VLANs

In either of the above cases, however, when data has to pass between floors, for example, when routing data between the Accounts and Sales subnets/broadcast domains/VLANs, there has to be one port dedicated on every switch and the router for each VLAN. However, if we use trunking, then these multiple ports can be combined into a single physical port that can carry data for multiple VLANs, while still maintaining separation.

The primary reason we may want different VLANs for different networks is because it allows us to divide the broadcast domains, which means that performance is instantly increased. Further, devices won't be able to perform a packet capture of unknown broadcast/multicast/unicast packets.

2.2.1 Packet Flow between VLANs

Let us consider the situation in diagram 2.3. We have PC₁ connected to VLAN₁₀₀, and PC₂ connected to VLAN₂₀₀, while the router is connected to the switch on a separate port, outside both VLANs, as a trunk (i.e., can carry data for multiple VLANs).

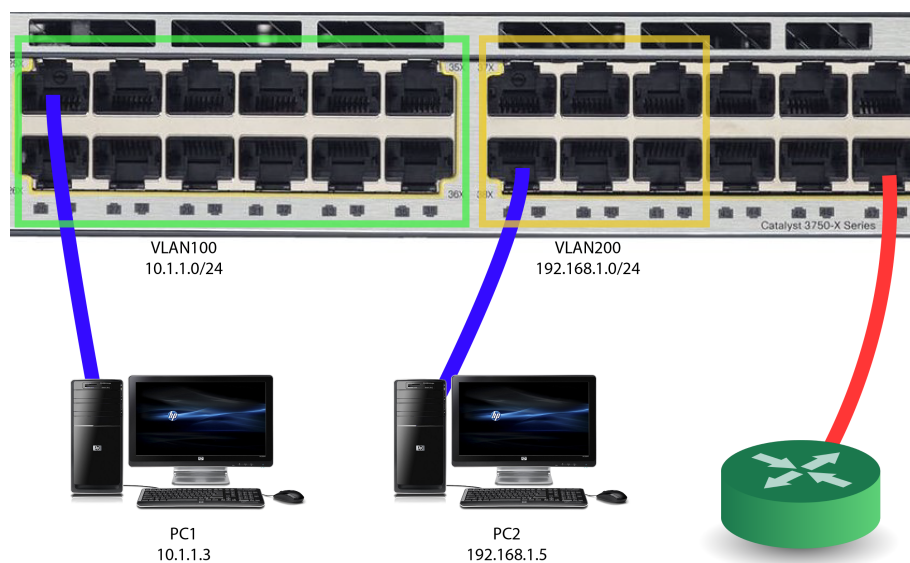


Figure 2.3: Routing between VLANs

When data has to go from VLAN₁₀₀ to VLAN₂₀₀, they can't directly pass since they're on both different subnets in different VLANs. Thus, a router is required to convey traffic from one VLAN to another.

So, data will go from the **ingress port** in VLAN₁₀₀, across the *switching fabric*, i.e., switching back-plane to the router through the trunk port. The router will discover that this packet is destined for the network of VLAN₂₀₀ (192.168.1.0/24), back across the trunk and then the switching fabric to reach the **egress port** in VLAN₂₀₀.

This kind of a configuration where a router is connected to a switch to manage data across VLANs is called *router on a stick* or **router on a trunk** connection. Some of the new Catalyst switches are Multilayer/Layer-3 Switches which don't need a router to perform the routing between VLANs.

2.3 VLAN Creation

2.3.1 Show existing VLANs

To see a brief overview of the existing VLANs, we use the `show vlan brief` command:

```
1 sw1#sh vlan brief
2
3 VLAN Name                                Status    Ports
4 -----
5 1      default                            active    Gi0/0, Gi0/1, Gi0/2, Gi0/3
6 1002 fddi-default                        act/unsup
7 1003 token-ring-default                  act/unsup
8 1004 fddinet-default                     act/unsup
9 1005 trnet-default                       act/unsup
```

The fddi, token ring VLANs can be ignored. The main VLAN is the default, VLAN₁.

2.3.2 Creating a new VLAN

Let us say we want a couple of new VLANs - VLAN₁₀₀ for the accounts department and VLAN₂₀₀ for the sales department in an office. Just like the interface configuration mode, we also have a VLAN configuration mode. The command used to name a VLAN is `name`. We name the two VLANs using:

```
1 sw1(config)#vlan 100
2 sw1(config-vlan)#name ACCT
3 sw1(config-vlan)#exit
4 sw1(config)#vlan 200
5 sw1(config-vlan)#name SALES
6 sw1(config-vlan)#end
7 sw1#sh vlan br
8
9 VLAN Name                                Status    Ports
10 -----
11 1      default                            active    Gi0/0, Gi0/1, Gi0/2, Gi0/3
12 100    ACCT                              active
13 200    SALES                              active
14 ...
```

We see that the VLANs have been created and now we can assign ports to it.

2.3.3 Deleting a VLAN

To delete a VLAN, we use the command `no vlan` followed by the VLAN number:

```
1 sw1(config)#vlan 300
2 sw1(config-vlan)#name TEST
3 sw1(config-vlan)#end
4 sw1#sh vlan br | i TEST
5 300 TEST active
6 sw1#conf t
7 Enter configuration commands, one per line. End with CNTL/Z.
8 sw1(config)#no vlan 300
9 sw1(config)#end
10 sw1#sh vlan br | i TEST
11 sw1#
```

Now, we may assume that the VLANs have been wiped, however, the information about VLANs are stored in a separate section of the memory, called the *flash*. We can view its contents using `show flash`. In it, we have a file called `vlan.dat`. To truly erase all the VLANs, we need to use the `delete flash:vlan.dat` command.

2.4 Assigning Ports to a VLAN

When a port is dedicated to a single VLAN, it's called an **access** port. Contrastingly, there can be **trunk** ports carrying data over multiple VLANs over a single interface. To assign a port to a VLAN, we need to declare it as an access port using the `switchport access` command. Let us consider we want to add the interface `gi0/0` to VLAN 100. We use:

```
1 sw1(config)#int gi0/0
2 sw1(config-if)#switchport access vlan 100
```

To put a range of interfaces all in the same VLAN, we use the *interface range* configuration mode instead of the *interface* config mode. To put all ports between Gi0/1 and Gi0/3 in VLAN 300, we use:

```
1 sw1(config)#interface range gi0/1 - 2
2 sw1(config-if-range)#switchport access vlan 200
3 sw1(config-if-range)#end
4 sw1#sh vlan br
5
6 VLAN Name                Status    Ports
7 -----
8 1    default              active    Gi0/3
9 100  ACCT                 active    Gi0/0
10 200  SALES                 active    Gi0/1, Gi0/2
11 ...
```

Chapter 3

Trunking

3.1 Trunking : Introduction

Since we don't want to mix the traffic between the VLANs, and since they're in different subnets, if we have to send traffic between multiple switches in our network, it might seem like we need separate interfaces to carry the traffic for each VLAN, but that's not the case, since we can use trunk ports. Trunk interfaces allow us to send the data for multiple VLANs over a single connection.

3.2 Trunking Theory

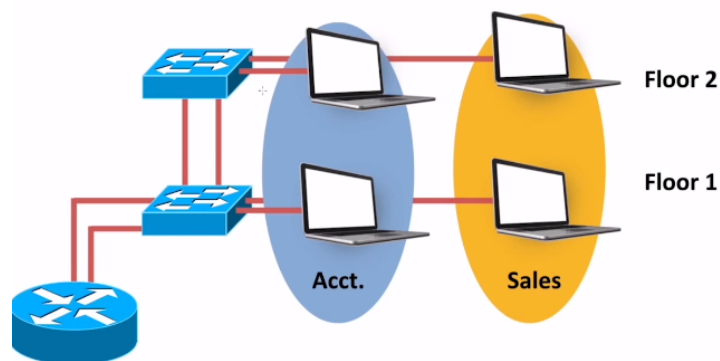


Figure 3.1: Using Access Ports

Using trunks, we can combine the data flowing over both wires between switches, and between the switch and the router.

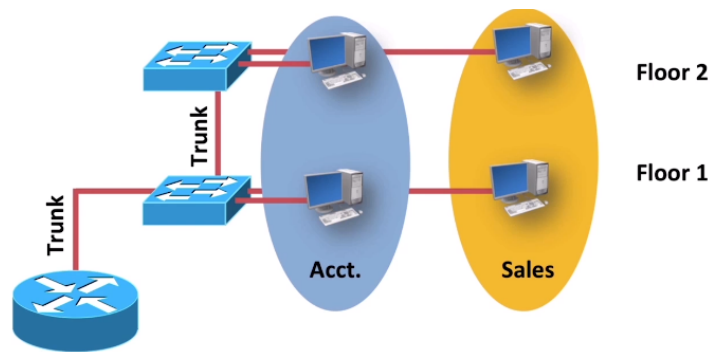


Figure 3.2: Using Trunk Ports

This raises a question of how the switches determine what VLAN the frame belongs to. This is done by 'tagging' each frame (with a number) when it travels through the trunk as belonging to a certain VLAN.

If a frame originating from the Accounting VLAN on the 2nd floor wants to go to the PC on the 1st floor, then the switch on the 2nd floor will tag the frame and when the switch on floor 1 gets it, it'll see that it's destined for the Accounts VLAN and send the frame to the appropriate port.

In case of inter-VLAN communication, the originating switch tags the frame for the accounts VLAN while travelling to the router. The router then receives the frame on one sub-interface (**logical interface**) and sends it out through another sub-interface. When the router forwards the packet to the Sales VLAN, it'll change the tagging to that of Sales VLAN and send it back up the trunk, to the appropriate port for the Sales PC.

3.2.1 Marking Frames for VLANs

For VLAN usage, 4 Bytes of data are added to the frame in accordance to the IEEE 802.1q frame format. Among these 4 bytes, 12 bits identify the particular VLAN a packet belongs to. 3 bits set the priority or **Quality of Service** for that frame. These 4B are:

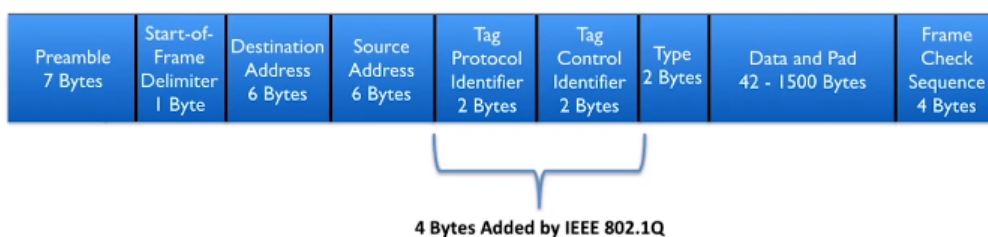


Figure 3.3: 802.1q Frame

3.2.2 Native VLAN

The frames for a Native VLAN aren't tagged. This is the only VLAN that sends untagged data, and the native VLAN can be configured for a switch. Of course, this means that the native VLAN on both switches need to be the same, i.e., they must agree which VLAN will send untagged data.

If one switch has native VLAN set to $VLAN_{100}$ while a switch on the other end of the connection has the native VLAN set to $VLAN_{200}$, then when the first switch will send data

destined for $VLAN_{100}$, the second switch will accidentally forward it to the wrong VLAN, i.e., $VLAN_{200}$.

3.3 Trunking Modes

When two switches are interconnected with a trunk, there are different trunking modes that they may be arranged in. They are:

| Mode | Description |
|--------------------------|---|
| Access | Forces a port to act as an Access port (i.e., unless it's for a voice VLAN, it will bear traffic for a single VLAN) |
| Trunk | Forces a port to act as a Trunk port |
| Dynamic Desirable | Initiates the negotiation of a trunk by sending the required DTP frames |
| Dynamic Auto | Passively waits for the remote switch to start negotiating a trunk formation. |

3.3.1 Formation of a Trunk

A trunk is formed via negotiation of the two switch ports at the end of a link through the **Dynamic Trunking Protocol (DTP)**, which is Cisco's proprietary protocol for this purpose. When a switch port set to **Dynamic Desirable** or **Dynamic Auto** mode receives a DTP frame, it will form a trunk. However, only a **Dynamic Desirable** or **Trunk** switch port can send the necessary DTP frames for trunk negotiation. Thus, we get the following table for trunk formations:

| Switch 1 Mode | Switch 2 Mode | Trunk Formation |
|-------------------|-------------------|-----------------|
| Access | <i>ANY</i> | No |
| Trunk | Dynamic Desirable | Yes |
| Trunk | Dynamic Auto | Yes |
| Trunk | Trunk | Yes |
| Dynamic Desirable | Dynamic Desirable | Yes |
| Dynamic Desirable | Dynamic Auto | Yes |
| Dynamic Auto | Dynamic Auto | No |

In case of Access modes, no matter what kind of frames the port receives, it'll ignore them since it's been hard-coded to be an access port. When there's a switch in trunk mode on one side, and a switch port in dynamic mode on the other side, they will form a trunk since both/one side respectively will send the DTP frames. In case of Trunk/Trunk configuration, both ports are independently told to form a trunk and hence both will send DTP frames and form a trunk.

In the case of Dynamic desirable ports on both sides, a trunk will form since one or both sides will receive the DTP frames from the partner and form a trunk. Similarly when there's dynamic auto on one side and dynamic desirable on the other, the later will send the DTP frames for trunk formation and the former will accept, thus forming a trunk. But when there's switch ports with *dynamic auto* trunk mode on both side, neither will start the negotiation for trunk formation, although they're willing to form a trunk. Due to this, a trunk **won't be formed**.

3.4 Creating Trunks

3.4.1 Finding out the current trunk mode of a switch port

The `show interfaces <interfaceName> switchport` command shows us the VLAN/Trunk details of a switch port, which can be written as `sh int <interfaceNumber> sw` for short:

```
1 sw1#show interfaces g0/0 switchport
2 Name: Gi0/0
3 Switchport: Enabled
4 Administrative Mode: dynamic auto
5 Operational Mode: static access
6 Administrative Trunking Encapsulation: negotiate
7 Operational Trunking Encapsulation: native
8 Negotiation of Trunking: On
9 Access Mode VLAN: 1 (default)
10 Trunking Native Mode VLAN: 1 (default)
11 Administrative Native VLAN tagging: enabled
12 Voice VLAN: none
13 Administrative private-vlan host-association: none
14 Administrative private-vlan mapping: none
15 Administrative private-vlan trunk native VLAN: none
16 Administrative private-vlan trunk Native VLAN tagging: enabled
17 Administrative private-vlan trunk encapsulation: dot1q
18 Administrative private-vlan trunk normal VLANs: none
19 Administrative private-vlan trunk associations: none
20 Administrative private-vlan trunk mappings: none
21 Operational private-vlan: none
22 Trunking VLANs Enabled: ALL
23 Pruning VLANs Enabled: 2-1001
24 Capture Mode Disabled
25 Capture VLANs Allowed: ALL
26
27 Protected: false
28 Appliance trust: none
```

In the above section, line #4 Administrative Mode: dynamic auto tells us that it's set to dynamic auto trunking mode, but the next line, Operational Mode: static access tells us that since the other side hasn't sent the required DTP frames, it's acting as an access port.

Let us consider we have 3 switches, sw1, sw2 and sw3 arranged in the following topology. Each interface starting with gi is a gigabitEthernet port:

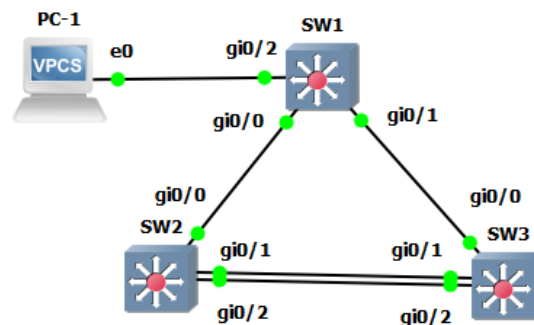


Figure 3.4: Trunk Topology

Let us consider we first want to set up the trunks between the switches *sw1* and *sw2*, and another trunk between the switches *sw1* and *sw3*. The interface we're currently inspecting in figure 3.4 is *gi0/0* on *sw1*, i.e., the one connecting to *gi0/0* on *sw2*.

We also see on line #6 that if a trunk is formed, the trunking encapsulation will be negotiated, due to: Administrative Trunking Encapsulation: negotiate. We might want to hard-code it to **dot1q**.

Finally, if a trunk were to be formed, the native VLAN (the one that carries frames that are untagged) will be set to VLAN1 due to line #10: Trunking Native Mode VLAN: 1 (default). We should also know how to change this behaviour.

3.4.2 Showing currently existing Trunks

The trunks that exist on all the interfaces currently can be seen with `show interfaces trunk`, or `sh int tr` for short. Since there are none currently, the result looks like:

```
1 sw1#show interfaces trunk
2 sw1#
```

When there are existing trunks, it looks like:

```
1
```

3.4.3 Changing Switch port's trunking settings

To change the trunk formation settings of a switch port, we use the `switchport trunk` command in the interface configuration mode. It has several parameters that we can control:

```
1 sw1(config-if)#switchport trunk ?
2   allowed          Set allowed VLAN characteristics when interface is in trunking
3                   mode
4   encapsulation    Set trunking encapsulation when interface is in trunking mode
5   native           Set trunking native characteristics when interface is in
6                   trunking mode
7   pruning          Set pruning VLAN characteristics when interface is in trunking
8                   mode
```

Changing Trunk Encapsulation

The trunk encapsulation determines the format in which the trunk negotiation is performed. This has the possible values of:

```
1 sw1(config-if)#switchport trunk encapsulation ?
2   dot1q           Interface uses only 802.1q trunking encapsulation when trunking
3   isl             Interface uses only ISL trunking encapsulation when trunking
4   negotiate       Device will negotiate trunking encapsulation with peer on
5                   interface
```

ISL is a proprietary encapsulation by Cisco that Cisco itself doesn't recommend any more. **dot1q** is the standard we want to use and *negotiate* will have the switch port negotiate and set up the encapsulation based on the partner's encapsulation settings.

We can set it to *dot1q* by using the `switchport trunk encapsulation dot1q` command, or `sw tr en dot` for short:

```
1 sw1(config-if)#sw tr e dot
2 sw1(config-if)#
```

Changing the native VLAN

The native VLAN can also be changed in a similar fashion using the `switchport trunk native vlan` command (`sw tr nat vlan` for short):

```
1 sw1(config-if)#sw tr n vlan 100
2 sw1(config-if)#
```

Changing the switchport trunking mode

We can change the current *dynamic auto* trunking mode to **dynamic desirable** using the `switchport mode dynamic desirable` command (`sw m dy des` for short):

```
1 sw1(config-if)#switchport mode ?
2   access      Set trunking mode to ACCESS unconditionally
3   dot1q-tunnel set trunking mode to TUNNEL unconditionally
4   dynamic     Set trunking mode to dynamically negotiate access or trunk mode
5   private-vlan Set private-vlan mode
6   trunk       Set trunking mode to TRUNK unconditionally
7
8 sw1(config-if)#switchport mode dynamic ?
9   auto        Set trunking mode dynamic negotiation parameter to AUTO
10  desirable    Set trunking mode dynamic negotiation parameter to DESIRABLE
11
12 sw1(config-if)#switchport mode dynamic desirable
13 sw1(config-if)#
```

3.4.4 Trunk settings mismatch

The effect of incompatibility of trunk settings on both sides of the link is immediate errors. As such, the moment we set the trunk's native VLAN on sw1's gi0/0 to **VLAN100** while the one on sw2's gi0/0 is still set to the default of **VLAN1**, we get:

```
1 sw1(config-if)#switchport mode dynamic desirable
2 sw1(config-if)#
3 *Nov 21 07:53:56.502: %SPANTREE-2-RECV_PVID_ERR: Received BPDU with inconsistent peer
   ↪ vlan id 1 on GigabitEthernet0/0 VLAN100.
4 *Nov 21 07:53:56.504: %SPANTREE-2-BLOCK_PVID_PEER: Blocking GigabitEthernet0/0 on
   ↪ VLAN0001. Inconsistent peer vlan.
5 *Nov 21 07:53:56.505: %SPANTREE-2-BLOCK_PVID_LOCAL: Blocking GigabitEthernet0/0 on
   ↪ VLAN0100. Inconsistent local vlan.
```

```

6  *Nov 21 07:54:25.670: %CDP-4-NATIVE_VLAN_MISMATCH: Native VLAN mismatch discovered on
   ↪ GigabitEthernet0/0 (100), with sw2 GigabitEthernet0/0 (1).
7  *Nov 21 07:55:18.671: %CDP-4-NATIVE_VLAN_MISMATCH: Native VLAN mismatch discovered on
   ↪ GigabitEthernet0/0 (100), with sw2 GigabitEthernet0/0 (1).
8  ...

```

This can be changed by setting up the partner's (sw2 gi0/0) trunk config appropriately and match the native VLAN. On sw2, we have:

```

1  sw2#sh int trunk
2
3  Port      Mode      Encapsulation  Status      Native vlan
4  Gi0/0     auto      n-802.1q       trunking    1
5  ...
6  sw2#
7  *Nov 21 08:03:44.500: %CDP-4-NATIVE_VLAN_MISMATCH: Native VLAN mismatch discovered on
   ↪ GigabitEthernet0/0 (1), with sw1.somuvmmnet.com GigabitEthernet0/0 (100).
8  sw2#sh int g0/0 sw
9  Name: Gi0/0
10 Switchport: Enabled
11 Administrative Mode: dynamic auto
12 Operational Mode: trunk
13 Administrative Trunking Encapsulation: negotiate
14 Operational Trunking Encapsulation: dot1q
15 Negotiation of Trunking: On
16 Access Mode VLAN: 1 (default)
17 Trunking Native Mode VLAN: 1 (default)

```

We see that a trunk has formed, but there's a VLAN mismatch. We match the settings with:

```

1  sw2#conf t
2  Enter configuration commands, one per line.  End with CNTL/Z.
3  sw2(config)#int g0/0
4  sw2(config-if)#sw t en dot
5  sw2(config-if)#sw t nat v 100
6  sw2(config-if)#end
7  sw2#sh int tr
8
9  Port      Mode      Encapsulation  Status      Native vlan
10 Gi0/0     auto      802.1q         trunking    100
11 ...
12 sw2#
13 *Nov 21 08:08:07.859: %SPANTREE-2-UNBLOCK_CONSIST_PORT: Unblocking GigabitEthernet0/0 on
   ↪ VLAN0001. Port consistency restored.
14 sw2#

```

3.4.5 Hard-coded Trunk mode

Let us quickly configure sw3 with the following:

```

1  sw3#conf t
2  Enter configuration commands, one per line.  End with CNTL/Z.
3  sw3(config)#int g0/0
4  sw3(config-if)#sw mo dyn auto
5  sw3(config-if)#sw tr encap dot
6  sw3(config-if)#sw tr native vlan 200

```

So, the trunk mode is set to **dynamic auto**, encapsulation to **dot1q** and the native VLAN to 200. Now, when we create a similar profile in *sw1* but set the mode to **trunk** instead of a dynamic type, and execute the `sh int tr`, we get:

```
1 sw1#sh int tr
2
3 Port      Mode      Encapsulation  Status      Native vlan
4 Gi0/0     desirable 802.1q         trunking    100
5 Gi0/1     on         802.1q         trunking    200
6
7 Port      Vlans allowed on trunk
8 Gi0/0     1-4094
9 Gi0/1     1-4094
10
11 Port      Vlans allowed and active in management domain
12 Gi0/0     1,100,200
13 Gi0/1     1,100,200
14
15 Port      Vlans in spanning tree forwarding state and not pruned
16 Gi0/0     1,100,200
17 Gi0/1     1,100,200
```

Note that the mode for *sw1 gi0/1* states **on** instead of *auto/desirable*. This indicates that the DTP frames are being sent by this particular interface to form a trunk and it's a static trunking.

3.5 VLAN pruning

By default, all VLANs are allowed over all trunks. This can be bad for security, given that all unknown broadcast, multicast and unicast traffic is sent over trunks. Thus, it'd enable malicious attackers to snoop on the traffic flowing through the trunk. Further, limiting VLANs reduces bandwidth utilization on the trunk and thus helps from a **Quality of Service (QoS)** perspective.

3.5.1 Showing VLANs allowed on a trunk

To see the particular VLANs allowed on a trunk, we use the `show interfaces <int#> trunk`. To see allowed VLANs for all trunks, we use the normal version, `sh int tr`:

```
1 sw1#sh interfaces g0/0 trunk
2
3 Port      Mode      Encapsulation  Status      Native vlan
4 Gi0/0     desirable 802.1q         trunking    100
5
6 Port      Vlans allowed on trunk
7 Gi0/0     1-4094
8
9 Port      Vlans allowed and active in management domain
10 Gi0/0     1,100,200
11
12 Port      Vlans in spanning tree forwarding state and not pruned
13 Gi0/0     1,100,200
```

We can see that over *Gi0/0*, all VLANs are allowed (VLANs 1-4094).

3.5.2 Allowing only certain VLANs over a certain trunk

Let us consider in the last example, sw1 only wants the VLAN₁₀₀ over the interface Gi0/0 and the VLAN₂₀₀ over the interface Gi0/1. To allow a VLAN over a switch port, we use: `switchport trunk allowed vlan <vlan numbers: 1,2,3...>` in the interface config mode.

```
1 sw1(config)#int g0/0
2 sw1(config-if)#sw tr allowed vlan 1,100
3 sw1(config-if)#end
4 sw1#sh int g0/0 tr
5
6 Port      Mode      Encapsulation  Status      Native vlan
7 Gi0/0     desirable  802.1q         trunking    100
8
9 Port      Vlans allowed on trunk
10 Gi0/0     1,100
11
12 Port      Vlans allowed and active in management domain
13 Gi0/0     1,100
14
15 Port      Vlans in spanning tree forwarding state and not pruned
16 Gi0/0     1,100
```

In the case we only wanted VLAN 100 to be banned, we could've used the command: `sw tr allowed vlan except 100`.

```
1 sw1(config)#int g0/1
2 sw1(config-if)#sw tr allowed vlan except 100
3 sw1(config-if)#end
4 sw1#sh interfaces trunk
5
6 Port      Mode      Encapsulation  Status      Native vlan
7 Gi0/0     desirable  802.1q         trunking    100
8 Gi0/1     on         802.1q         trunking    200
9
10 Port      Vlans allowed on trunk
11 Gi0/0     100
12 Gi0/1     1-99,101-4094
13
14 Port      Vlans allowed and active in management domain
15 Gi0/0     100
16 Gi0/1     1,200
17
18 Port      Vlans in spanning tree forwarding state and not pruned
19 Gi0/0     100
20 Gi0/1     1
```

Chapter 4

Troubleshooting Switch Operation

4.1 Isolating the Issue

Let us consider in the following topology, there's a PC connected to one switch and a server connected to the other, and they can't connect to each other, or the speed is very slow.

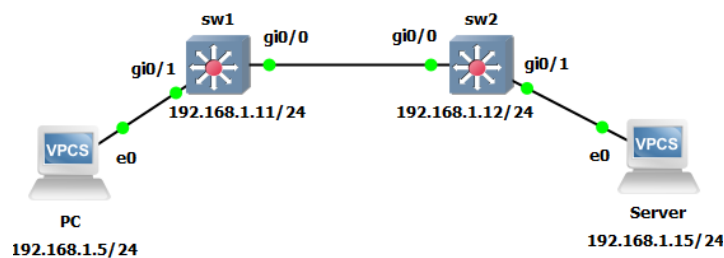


Figure 4.1: Switching Topology

The first step would be to isolate the issue. Some of the possible steps may be:

- Since the end devices can't ping each other, try pinging one switch from the other.
- If the switches are close by, try swapping the network cable
- To eliminate possible issues with the port, try changing to another port on each switch.
- Check the switch configuration for speed/duplex errors
- Try swapping the switch itself.

4.1.1 Pinging to test network connectivity

One of the first steps to check the connectivity between two devices is to **ping** one device from another. The ping command sends ICMP echo packets and waits for ICMP echo replies from the partner to establish connectivity. The ping command can be used by:

```
1 sw1#ping 192.168.1.12
2 Type escape sequence to abort.
```

```
3 Sending 5, 100-byte ICMP Echos to 192.168.1.12, timeout is 2 seconds:
4 !!!!!
5 Success rate is 100 percent (5/5), round-trip min/avg/max = 7/8/12 ms
```

Each **!** or *bang* indicates that a ping was successful, i.e., an echo reply was received for an echo. In case the reply packet gets lost/dropped, the output would be a *.* or *dot*. Thus, a connection that suffers from 40% packet loss *might* look like:

```
1 sw1#ping 192.168.1.12
2 Type escape sequence to abort.
3 Sending 5, 100-byte ICMP Echos to 192.168.1.12, timeout is 2 seconds:
4 !...!
5 Success rate is 60 percent (3/5), round-trip min/avg/max = 5/9/16 ms
```

It also tells us about the round-trip time. In this case, the minimum was *5ms* while the average was *9ms*. Generally, anything under **10ms** is considered a fine, but substantially higher ones would indicate *delays/latency*.

4.1.2 Extended Ping and Ping timeout due to ARP

Sometimes, we may have the first packet time-out, but the rest succeed. This may be because at the time of the first ping, the MAC address of the destination host was unknown to the switch and it had to learn the MAC address by using **ARP(Address Resolution Protocol)**.

Since our switch doesn't know the MAC address for the machine with the IP, it can't fill the destination address in the ICMP echo's layer 2 frame. Thus, the switch will first send a broadcast ARP packet since it wants to know the MAC address of machine with the IP 192.168.1.15. Only the machine with this IP would respond, and our switch can store the MAC address corresponding to it in its MAC address table. The delay in this may cause the first ping to time-out. Successively, a ping is only sent to the right MAC address and we get a reply.

We can actually test this with an extended ping, which is a ping command without any parameters. First, we need to clear the ARP Cache, which is where the known MAC addresses are stored and mapped to IP addresses, unlike the MAC/CAM table, which only maps which MAC addresses can be reached via which interface. Thus, the MAC address table stores Layer 2 information while the ARP table/cache stores Layer 3 mappings. The ARP cache can be cleared with the `clear arp-cache` command, and can be viewed with a `show arp` command.

Once cleared, we can use the extended ping to reduce the timeout to 1 second, which is enough time for the ping to fail the first time due to ARP still working:

```
1 sw1#clear arp-cache
2 sw1#sh arp
3 Protocol Address Age (min) Hardware Addr Type Interface
4 Internet 192.168.1.11 - 0c73.a0bb.8001 ARPA Vlan1
5 sw1#ping
6 Protocol [ip]:
7 Target IP address: 192.168.1.12
8 Repeat count [5]:
9 Datagram size [100]:
10 Timeout in seconds [2]: 1
11 Extended commands [n]:
12 Sweep range of sizes [n]:
13 Type escape sequence to abort.
```



```

14 Sending 5, 100-byte ICMP Echos to 192.168.1.12, timeout is 1 seconds:
15 .!!!!
16 Success rate is 80 percent (4/5), round-trip min/avg/max = 6/7/9 ms
17 sw1#sh arp
18 Protocol Address Age (min) Hardware Addr Type Interface
19 Internet 192.168.1.11 - 0c73.a0bb.8001 ARPA Vlan1
20 Internet 192.168.1.12 0 0c73.a075.8001 ARPA Vlan1
21 sw1#ping 192.168.1.12 timeout 1
22 Type escape sequence to abort.
23 Sending 5, 100-byte ICMP Echos to 192.168.1.12, timeout is 1 seconds:
24 !!!!!
25 Success rate is 100 percent (5/5), round-trip min/avg/max = 8/10/15 ms

```

In the above case, we can see that once the *ARP Cache* has been cleared and the ping command is used with a time-out of only 1 second, the first ping fails, after which the ARP table gets an entry for the pinged IP, and all subsequent pings succeed.

NOTE: The clear arp-cache command may simply send an ARP broadcast at the end of clearing the cache to get the new MAC addresses for the connected devices, which means instead of clearing the table, the age of the MAC address would only be shown as 0min. To prevent this for the above demonstration, the interface(s) were administratively shut down before clearing the cache, and then brought back up.

4.2 Checking Interface Status

The show interfaces command by itself would show me the detailed information about every single interface available, but we generally want to focus on a particular interface, the name of which we can provide as the parameter to get information about only that specific interface:

```

1 sw1#sh int g0/0
2 GigabitEthernet0/0 is up, line protocol is up (connected)
3 Hardware is iGbE, address is 0c73.a0bb.f700 (bia 0c73.a0bb.f700)
4 MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
5 reliability 255/255, txload 1/255, rxload 1/255
6 Encapsulation ARPA, loopback not set
7 Keepalive set (10 sec)
8 Auto Duplex, Auto Speed, link type is auto, media type is unknown media type
9 output flow-control is unsupported, input flow-control is unsupported
10 Full-duplex, Auto-speed, link type is auto, media type is RJ45
11 input flow-control is off, output flow-control is unsupported
12 ARP type: ARPA, ARP Timeout 04:00:00
13 Last input 00:00:00, output 00:00:02, output hang never
14 Last clearing of "show interface" counters never
15 Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
16 Queueing strategy: fifo
17 Output queue: 0/0 (size/max)
18 5 minute input rate 0 bits/sec, 0 packets/sec
19 5 minute output rate 0 bits/sec, 0 packets/sec
20 3686 packets input, 241522 bytes, 0 no buffer
21 Received 3609 broadcasts (3609 multicasts)
22 0 runs, 0 giants, 0 throttles
23 0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
24 0 watchdog, 3609 multicast, 0 pause input
25 715 packets output, 92171 bytes, 0 underruns
26 0 output errors, 0 collisions, 3 interface resets
27 0 unknown protocol drops

```

```

28      0 babbles, 0 late collision, 0 deferred
29      0 lost carrier, 0 no carrier, 0 pause output
30      0 output buffer failures, 0 output buffers swapped out

```

The first line of the output can be interpreted says: GigabitEthernet0/0 is up, which indicates that the layer 1 of the interface is up. This means that a **carrier detect** signal is being received from the far end of this link. The next part, line protocol is up (connected) is the data link layer status, and it means layer 2 is up as well, which is why it says that the interface is connected. This is the case when *keep-alives* are being received in the link layer. When an interface is operational, we see that it's in an **up/up** state, but it can be in several others:

| Interface | Line protocol | Meaning |
|------------------------------|-------------------------|--|
| Up | Up(connected) | Up and Functional |
| Up | Down | Connection Issue (Layer 2 keep-alives not being received). |
| Down | Down(notconnect) | Cable isn't connected to at least one switch or the far end switch port has been administratively shut down. |
| Down | Down | Layer 1 hardware/interface issue. |
| Administratively Down | Down | Interface set to down state by admin, perhaps for security reasons because the port is unused. |

4.2.1 Show interfaces status

The `show interfaces status` command is very useful since it shows us whether that interface is connected, which VLAN it belongs to, or if it's a trunk, and most notably, the speed and duplex settings on the interface as well as the media type used, for all available interfaces:

```

1  sw1#sh int status
2
3  Port      Name              Status      Vlan      Duplex  Speed  Type
4  Gi0/0                        connected   trunk     a-full  auto   RJ45
5  Gi0/1                        connected   1         a-full  auto   RJ45
6  Gi0/2                        connected   1         a-full  auto   RJ45
7  Gi0/3                        connected   1         a-full  auto   RJ45

```

In the case above, we can see that both the duplex and the speed settings have been set to auto-negotiation. Auto-negotiation can however, sometimes cause unexpected problems. Let us set up sw2 with a hard-coded 100Mbps speed on the trunk port and hard-coded full duplex settings. First when we try to do it, we get:

```

1  sw2#conf t
2  Enter configuration commands, one per line. End with CNTL/Z.
3  sw2(config)#int g0/0
4  sw2(config-if)#duplex full
5  Autoneg enabled. Duplex cannot be set

```

It says that we can't manually set the duplex/speed settings since the auto-negotiation is turned on. Thus, we need to turn off auto-negotiation on **sw2** using `no negotiation auto` and then change the duplex and speed settings:

```

1  sw2(config-if)#no neg auto
2  sw2(config-if)#speed 100
3  sw2(config-if)#duplex full
4  sw2(config-if)#end
5  sw2#
6  *Nov 26 06:38:17.399: %SYS-5-CONFIG_I: Configured from console by console
7  sw2#sh int status
8
9  Port      Name      Status      Vlan      Duplex  Speed Type
10 Gi0/0      Status      connected   trunk     full    100 RJ45
11 Gi0/1      Status      connected   1         a-full  auto RJ45
12 Gi0/2      Status      connected   1         a-full  auto RJ45
13 Gi0/3      Status      connected   1         a-full  auto RJ45

```

Back on **sw1**, we still have the port set to auto-negotiation:

```

1  sw1#sh int g0/0 status
2
3  Port      Name      Status      Vlan      Duplex  Speed Type
4  Gi0/0      Status      connected   trunk     a-full  auto RJ45

```

Given that one side is set to full-duplex, and the other side to auto, it's reasonable to expect that the auto-negotiation will resolve to set full duplex on **sw1**, but that's not the case. When one side is hard coded to full duplex, it *might not* participate in auto-negotiation (at least in older versions of IOS).

Under such circumstances, the switch with the auto-negotiation settings turned on will fall back to half-duplex, shown as a-half because the other side is hard-coded to full duplex and didn't participate in auto-negotiation. When this happens, collisions will increase, and users will complain about slowness. Thus, traffic will still flow, but with severely degraded performance.

4.3 Checking for Interface Errors

Interface errors may appear for a variety of Layer 1 or 2 issues. Layer 2 issues are typically misconfiguration. In copper cabling, some typical causes are wrong cable type, damaged cables, etc. In case of fibre optic cables, there's a **minimum bend ratio** that has to be maintained to prevent light from leaking into the cladding from the core. Some of the types of errors we see are:

| | |
|------------------------|--|
| Runts | A frame that has a smaller size than the medium's minimum frame size (ex. 64B on Ethernet network) and has a bad CRC. |
| Giants | A frame that has a larger size than the medium's maximum frame size (ex. 1518B on a non-jumbo Ethernet network) and has a bad CRC. |
| Input Error | Sum of all interface errors (runts, giants, no buffer, CRC, frame, over-run and ignore) |
| Output Error | Total number of errors that caused a failure in the transmission of packets. |
| Collisions | Total number of collisions (ex. in Ethernet networks working in half-duplex mode) |
| Late Collisions | Number of times a collision occurred late in the transmission, i.e., number of collisions after the medium's slot time , i.e., a transmission that occurred after the Tx NIC had completed transmitting. |
| CRC | Number of times the CRC value at the receiving end didn't match the CRC value at the transmitting end. |

Slot Time: It is the minimum amount of time required to wait for the medium to be free of transmissions (and hence of collisions). Let us consider an Ethernet cable. It has a maximum transmission distance of about 100m. Let us call the time taken by a single bit (i.e., an electronic pulse) to travel this distance the Propagation time (T_p). Then the slot time (T_s) can be any time greater than twice the propagation time $T_s > 2T_p$.

A late collision is any collision that occurs after a NIC has completed transmitting. Thus, to prevent them, we have to wait for all transmission to stop on the shared network medium segment before retransmitting. We also have to allow for any/all replies to the original segment to pass through the network. Hence the value of $2T_p$.

Such collisions shouldn't even be possible, and hence, these frames aren't retransmitted, and are left to the mercy of the higher level protocols to detect a collision and retransmit. Late collisions occur because of duplex mismatches, when an Ethernet cable exceeds its length limits, defective hardware such as incorrect cabling is used, non-compliant number of hubs are used in the network, or a **NIC (Network Interface Card)** malfunctions.

Since a switch operating in a full-duplex manner effectively has no collision domain, there isn't any chances of collisions occurring. Hence, 10Gig ports which can't work in half-duplex have no slot time and don't have any collision errors. The typical value for slot times are:

| Speed | Slot Time | Time Interval |
|------------|----------------|--------------------------|
| 10 Mbit/s | 512 bit times | 51.2 microseconds |
| 100 Mbit/s | 512 bit times | 5.12 microseconds |
| 1 Gbit/s | 4096 bit times | 4.096 microseconds |
| 2.5 Gbit/s | (onward) | no half-duplex operation |

A **bit time** is the time taken by the NIC to eject a bit onto the wire. A point to be noted is that there aren't even any 1Gig interfaces capable of operating in half-duplex. Hence, for all practical purposes, all devices operating on a speed of 1Gbps or greater always operate in full-duplex.

CRC: A **CRC (Cyclical Redundancy Check)** is a type of **FCS (Frame Check Sequence)**, a mathematical operation much like hashing that's done on the contents of a frame that is calculated at both ends of the link. If they match, we are sure that the frame hasn't degraded in transit. If they don't, the packet is useless and retransmission is required.

4.3.1 Duplex Mismatch

As described previously, a duplex mismatch can cause late collisions. This is because one side of the network thinks it's communicating on a medium in full duplex and doesn't run CSMA/CD or wait for retransmission after collisions, while the other does. This means when the half duplex side detects a collision, it starts jamming but the full duplex side remains clueless and continues transmission as usual.

At the end of the wait period (for retransmission), i.e., after the slot time has passed, when late collisions can occur, the half duplex side still sees the full duplex end transmitting, causing further collisions. Thus, the half-duplex side will see an increase in the number of late collisions.

The full duplex side doesn't even realize that collisions have occurred, and thinks the jamming signal from the collision is a response to its transmission and generates a lot of CRC errors. Thus, the tell-tale of a duplex mismatch is late collisions on one end and CRC errors on the other.

4.4 Checking a Port's VLAN Membership

Since VLANs separate different subnets, if the two devices are on both sides of an interface are on different VLANs, they won't be able to communicate. We can see which ports are assigned to which VLAN using the `sh vlan br` command:

```
1 sw1#sh vlan br
2
3 VLAN Name                                Status    Ports
4 -----
5 1      default                            active    Gi0/1, Gi0/2, Gi0/3
6 1002 fddi-default                        act/unsup
7 1003 token-ring-default                  act/unsup
8 1004 fddinet-default                     act/unsup
9 1005 trnet-default                       act/unsup
```

Notice that it only shows access ports, since trunk ports may carry data for multiple VLANs. Another issue is when a VLAN is deleted (ex. `no vlan 100`) and then later reused, they initially don't belong to any VLAN, and thus will be isolated and unable to carry traffic. Thus the recommendation is to always reassign ports on a VLAN before deleting the VLAN itself.

4.5 Checking a Trunk's Status

To start troubleshooting trunks on switches, one of the most useful commands is the `show interface trunk` command:

```
1 sw1#sh int trunk
2
3 Port      Mode           Encapsulation  Status        Native vlan
4 Gi0/0     desirable     802.1q         trunking      1
5
6 Port      Vlans allowed on trunk
7 Gi0/0     1
8
9 Port      Vlans allowed and active in management domain
10 Gi0/0     1
11
12 Port      Vlans in spanning tree forwarding state and not pruned
13 Gi0/0     1
```

Some possible issues with trunks may be:

- Mismatch in trunk mode (dynamic auto/dynamic auto, etc.)
- Mismatch in encapsulation
- Mismatch in native VLAN.

The last item in the list, a mismatch in the *Native VLAN* may cause **VLAN hopping**, which at best makes the traffic flow impossible (since it's impossible to communicate between different subnets without a router to translate IPs) and at worst may be a vulnerable point of attack and a security risk.

Chapter 5

Basic Switch Security

5.1 Physical Security

Instead of a single layer of security, Cisco recommends overlapping layers of security. Physical threats include dropping/damaging devices and/or network components. This is handled best with extensive employee training. There may also be electrical threats such as not being properly grounded while installing a card on a device and consequently damaging the equipment with static shocks. There may even be unstable voltage spikes/thunderstorms. This again is handled best by employee training and uninterrupted power supplies ensuring the equipment doesn't shut down unexpectedly. Then, to protect equipment against malicious users and/or prevent device theft, strategies such as biometric locks to grant access to the network cabinet, etc. may be considered. Finally there's also environmental hazards, such as a HVAC/air conditioning system that causes problems with the equipment by straying too far from their operating temperature. In such cases, it's beneficial to have monitoring systems to automatically detect and/or fix the issue.

5.2 Switch Port Security

5.2.1 MAC Flooding Attack

A malicious user may flood the port of a switch with spoofed MAC addresses using some software, thus filling up the CAM table of the switch with bogus MAC addresses. After this point, when a new device has to be reached, since the CAM table is already filled, the switch begins operating like a hub and broadcasts every packet it receives across all the other ports but the receive port. This allows the attacker to capture packets flowing across the switch. This can be prevented with **Port Security**.

5.2.2 Port Security

With port security, we can ensure that we don't have too many MAC addresses or any disallowed MAC addresses connected off a specific port. The port security feature can also allow things like limiting a single port to just 1 MAC addresses so that users can't connect their own switch to split the internet connection. The pre-requisite for port security is that the port has to be a static access port and not any dynamic/static trunk port. This is done using the `switchport mode access` command. Then we simply use the command `switchport port-security`:

```
1 sw1(config-if)#sw mo access
2 sw1(config-if)#sw port-sec
```

Port Security Setup

Now we can set the maximum permitted number of MAC addresses that can be learnt by this port and also set up disallowed MAC addresses on this port. We can set the maximum number of MAC addresses by:

```
1 sw1(config-if)#sw port-sec ?
2   aging      Port-security aging commands
3   mac-address Secure mac address
4   maximum    Max secure addresses
5   violation   Security violation mode
6   <cr>
7
8 sw1(config-if)#sw port-sec max ?
9   <1-4097>   Maximum addresses
10
11 sw1(config-if)#sw port-sec max 2
```

This makes it so that the switch can only learn 2 MAC addresses from that port. We can now let these be the first two MAC addresses learnt on the port or statically set the MAC address. In case we want the two addresses to be specified statically, we use:

```
1 sw1(config-if)#sw port-sec mac-address ?
2   H.H.H      48 bit mac address
3   forbidden   Configure mac address as forbidden on this interface
4   sticky      Configure dynamic secure addresses as sticky
```

Thus, we simply need to issue the command replacing the ? with a MAC address and run the command twice. But running the command for each MAC address isn't scalable and hence we have the option for sticky learning, where the first two MAC addresses will be the only ones that are allowed. These, however, will only reside in the running-configuration and will be reset with a reload. We can also set up forbidden MAC addresses. To make them persistent past reboots, we need to execute a `copy run start` command to save the settings to disk.

Port Security Violation

There are three options available to us when a port-security violation occurs, which are: **protect**, **restrict** and **shutdown**:

```
1 sw1(config-if)#sw port-sec violation ?
2   protect    Security violation protect mode
3   restrict    Security violation restrict mode
4   shutdown    Security violation shutdown mode
```

While both *protect* and *restrict* mode allow genuine traffic from allowed MAC addresses to flow, the last option, *shutdown* shuts down the port due to the security violation. The action of each mode is explained in the table below:

| Mode | Explanation |
|------------------|---|
| Protect | It simply drops all frames from a non-allowed MAC address while the traffic from allowed ports flows through unaffected. |
| Restirtct | Just like protect it lets traffic from allowed MAC addresses pass through while dropping frames from those MAC addresses that are not allowed, but at the same time, it increases a counter called the Security Violation counter each time a violation occurs. |
| Shutdown | When a security violation occurs it shuts down the port and puts the interface in a Error Disabled state, since it's detected that malicious activities are being performed. Further, it sends an SNMP trap if SNMP (Simple Network Management Protocol) is configured on the switch. |

When a port is shut-down due to a port-security violation, the first line of the show interface command shows:

```

1 sw1#sh int g0/1
2 GigabitEthernet0/1 is down, line protocol is down (err-disabled)

```

To bring back an interface into a connected state, if the underlying problem has been sorted, then a simple shut followed by no shut has to be done on the interface to bring it back up to operational status:

```

1 sw1(config)#int g0/1
2 sw1(config-if)#shut
3 sw1(config-if)#no shut
4 *Nov 27 03:28:01.322: %LINK-5-CHANGED: Interface GigabitEthernet0/1, changed state to
  ↳ administratively down
5 *Nov 27 03:28:04.454: %LINK-3-UPDOWN: Interface GigabitEthernet0/1, changed state to up
6 *Nov 27 03:28:05.456: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1,
  ↳ changed state to up
7 sw1(config-if)#end
8 sw1#sh int g0/1
9 GigabitEthernet0/1 is up, line protocol is up (connected)

```

This however leads to a lot of administrative burden, and the ports can be configured to automatically come back up after violations, using **Error Disabled Port Automatic Recovery**. This setting has to be configured globally and allows any port on the switch to try to come back up once the condition causing the port to be in *Error-Disabled* has been resolved.

There's a lot of reasons why a port might go into the *Error-Disabled* state. The options of how to disable and what to do after disabling are configured using the errdisabled command:

```

1 sw1(config)#errdisable ?
2   detect          Error disable detection
3   flap-setting    Error disable flap detection setting
4   recovery        Error disable recovery
5
6 sw1(config)#errdisable recovery ?
7   cause          Enable error disable recovery for application
8   interval       Error disable recovery timer value
9
10 sw1(config)#errdisable recovery cause ?
11   all            Enable timer to recover from all error causes
12   arp-inspection Enable timer to recover from arp inspection error
13                 disable state
14   bpduguard      Enable timer to recover from BPDU Guard error

```

| | | |
|----|----------------------|--|
| 15 | channel-misconfig | Enable timer to recover from channel misconfig error |
| 16 | | (STP) |
| 17 | dhcp-rate-limit | Enable timer to recover from dhcp-rate-limit error |
| 18 | dtp-flap | Enable timer to recover from dtp-flap error |
| 19 | gbic-invalid | Enable timer to recover from invalid GBIC error |
| 20 | inline-power | Enable timer to recover from inline-power error |
| 21 | l2ptguard | Enable timer to recover from l2protocol-tunnel error |
| 22 | link-flap | Enable timer to recover from link-flap error |
| 23 | link-monitor-failure | Enable timer to recover from link monitoring failure |
| 24 | loopback | Enable timer to recover from loopback error |
| 25 | mac-limit | Enable timer to recover from mac limit disable state |
| 26 | oam-remote-failure | Enable timer to recover from OAM detected remote |
| 27 | | failure |
| 28 | pagp-flap | Enable timer to recover from pagp-flap error |
| 29 | port-mode-failure | Enable timer to recover from port mode change failure |
| 30 | pppoe-ia-rate-limit | Enable timer to recover from PPPoE IA rate-limit error |
| 31 | psecure-violation | Enable timer to recover from psecure violation error |
| 32 | psp | Enable timer to recover from psp |
| 33 | security-violation | Enable timer to recover from 802.1x violation error |
| 34 | sfp-config-mismatch | Enable timer to recover from SFP config mismatch error |
| 35 | storm-control | Enable timer to recover from storm-control error |
| 36 | udld | Enable timer to recover from udld error |
| 37 | unicast-flood | Enable timer to recover from unicast flood error |
| 38 | vmps | Enable timer to recover from vmps shutdown error |

Since our cause for disabling the port is a violation of the port security, we use the option `psecure-violation`. Thus, to enable the auto-resuscitation of a error disabled port, we just need to use the command:

```

1 sw1(config)#errdisable recovery cause psecure-violation
2 sw1(config)#

```

Now the system will try to bring the port back up at regular intervals, which is by default 300 seconds, i.e., 5 mins. This can be changed using:

```

1 sw1(config)#errdisable recovery interval 30

```

We can also see which conditions are set to automatically come out of *error-disable* using: `show errdisable recovery` command:

```

1 sw1#sh errdisable recovery
2 ErrDisable Reason      Timer Status
3 -----
4 arp-inspection          Disabled
5 bpduguard               Disabled
6 channel-misconfig (STP) Disabled
7 dhcp-rate-limit         Disabled
8 dtp-flap                Disabled
9 gbic-invalid            Disabled
10 inline-power            Disabled
11 l2ptguard               Disabled
12 link-flap               Disabled
13 mac-limit               Disabled
14 link-monitor-failure    Disabled
15 loopback                Disabled
16 oam-remote-failure      Disabled
17 pagp-flap               Disabled
18 port-mode-failure       Disabled

```

```

19 pppoe-ia-rate-limit      Disabled
20 psecure-violation        Enabled
21 security-violation        Disabled
22 sfp-config-mismatch      Disabled
23 storm-control            Disabled
24 udld                     Disabled
25 unicast-flood            Disabled
26 vmps                     Disabled
27 psp                      Disabled
28 dual-active-recovery      Disabled
29 evc-lite input mapping fa Disabled
30 Recovery command: "clear  Disabled
31
32 Timer interval: 30 seconds
33
34 Interfaces that will be enabled at the next timeout:

```

Port Security specific Verification Commands

To see which ports have port security enabled we use the `show port-security` command, which shows us how many MAC addresses are allowed, how many that port currently knows and in case of *restricted mode*, the number of times the security violation has occurred :

```

1 sw1#sh port-sec
2 Secure Port  MaxSecureAddr  CurrentAddr  SecurityViolation  Security Action
3              (Count)        (Count)        (Count)
4 -----
5      Gi0/1          2          2          0          Shutdown
6 -----
7 Total Addresses in System (excluding one mac per port)      : 1
8 Max Addresses limit in System (excluding one mac per port) : 4096

```

We use the `show port-security addresses` command to see which MAC addresses are currently allowed on the port:

```

1 sw1#sh port-sec addr
2          Secure Mac Address Table
3 -----
4 Vlan    Mac Address      Type                Ports  Remaining Age
5                                     (mins)
6 -----
7      1    0050.7966.6802  SecureSticky        Gi0/1    -
8      1    0c73.a08e.5400  SecureSticky        Gi0/1    -
9 -----
10 Total Addresses in System (excluding one mac per port)      : 1
11 Max Addresses limit in System (excluding one mac per port) : 4096

```

We use `show port-security interface <interfaceNum>` command to see the details of a port that has port-security enabled:

```

1 sw1#sh port-sec int g0/1
2 Port Security          : Enabled
3 Port Status           : Secure-up
4 Violation Mode         : Shutdown
5 Aging Time             : 0 mins
6 Aging Type             : Absolute

```

```

7 SecureStatic Address Aging : Disabled
8 Maximum MAC Addresses      : 2
9 Total MAC Addresses        : 2
10 Configured MAC Addresses   : 0
11 Sticky MAC Addresses       : 2
12 Last Source Address:Vlan   : 0c73.a08e.5400:1
13 Security Violation Count   : 0

```

5.3 Shutting down Unused Ports

A best-practice for networking equipment is to administratively shut-down unused ports. This may be to prevent unauthorized users from plugging in to an unused/un-configured port to launch attacks. Further, it also disables anyone from using any RJ-45 connector on the wall to plug-in to the network and intercept packets. Let us consider the topology below:

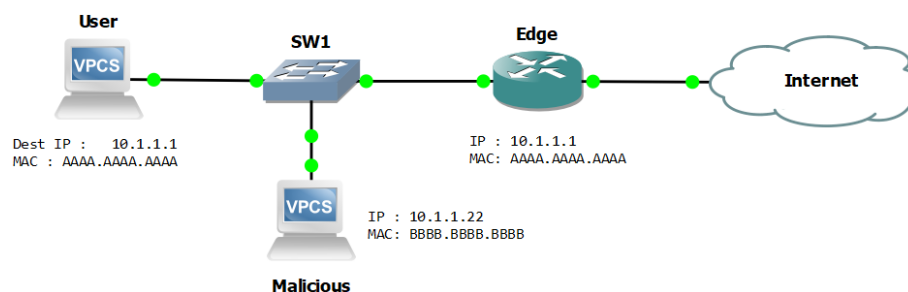


Figure 5.1: Man in the Middle attack

Let us consider the computer marked *user* has default gateway of 10.1.1.1, i.e., the edge router. It passes through a switch, **sw1** to which another malicious user is also connected with an IP of 10.1.1.22 and a MAC address of BBBB.BBBB.BBBB. Also note that the MAC Address of the Edge router is AAAA.AAAA.AAAA.

Now, if *user* wants to send some data through the *edge* router, it's gonna have to first send an ARP request to get the MAC address of the machine with the IP 10.1.1.1. This would pass through switch *sw1* and reach *edge*, and ideally generate an ARP reply from *edge* with it's MAC address, AAAA.AAAA.AAAA that the switch *sw1* will store in its CAM table.

The *malicious* user has gained access through the network via an unused port on *sw1*. If *malicious* user knows the IP address of the *user*, it can send an un-solicited ARP reply, also called a gratuitous ARP stating that the IP address 10.1.1.1 of the *edge* router is actually assigned to a host with the MAC address BBBB.BBBB.BBBB.

Now, *user* will believe that the gateway is *malicious* user and send all outgoing packets to it, which the malicious user can relay to the *edge* router. At the same time, *malicious* user can now intercept every packet destined from the *user* to the *edge* router.

5.3.1 Shutting down all ports on a switch

The recommendation to prevent man-in-the-middle attacks, etc. from succeeding is to administratively shut-down all the ports on a switch out-of-the-box and then perform a `no shut` on them as and when needed. All of the ports on the switch can be shut down together by:

```

1 sw1#sh ip int br
2 Interface          IP-Address      OK? Method Status Protocol

```

```

3  GigabitEthernet0/0      unassigned      YES unset  up
4  GigabitEthernet0/1      unassigned      YES unset  up
5  GigabitEthernet0/2      unassigned      YES unset  up
6  GigabitEthernet0/3      unassigned      YES unset  up
7  Vlan1                   10.1.1.5        YES manual up
8  sw1#conf t
9  Enter configuration commands, one per line. End with CNTL/Z.
10 sw1(config)#int range g0/0 - 3
11 sw1(config-if-range)#shut
12 *Nov 27 05:25:07.098: %LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to
   ↪ administratively down
13 *Nov 27 05:25:07.143: %LINK-5-CHANGED: Interface GigabitEthernet0/1, changed state to
   ↪ administratively down
14 *Nov 27 05:25:07.186: %LINK-5-CHANGED: Interface GigabitEthernet0/2, changed state to
   ↪ administratively down
15 *Nov 27 05:25:07.230: %LINK-5-CHANGED: Interface GigabitEthernet0/3, changed state to
   ↪ administratively down
16 *Nov 27 05:25:08.104: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0,
   ↪ changed state to down
17 *Nov 27 05:25:08.145: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1,
   ↪ changed state to down
18 *Nov 27 05:25:08.186: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/2,
   ↪ changed state to down
19 *Nov 27 05:25:08.234: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/3,
   ↪ changed state to down
20 sw1(config-if-range)#exit
21 sw1(config)#int g0/0
22 sw1(config-if)#no shut
23 sw1(config-if)#end
24 sw1#
25 *Nov 27 05:25:36.362: %LINK-3-UPDOWN: Interface GigabitEthernet0/0, changed state to up
26 *Nov 27 05:25:37.364: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0,
   ↪ changed state to up
27 sw1#sh ip int br
28 Interface                IP-Address      OK? Method Status      Protocol
29 GigabitEthernet0/0      unassigned      YES unset  up
30 GigabitEthernet0/1      unassigned      YES unset  administratively down down
31 GigabitEthernet0/2      unassigned      YES unset  administratively down down
32 GigabitEthernet0/3      unassigned      YES unset  administratively down down
33 Vlan1                    10.1.1.5        YES manual up
34 sw1#

```

Now, we can individually bring up the required interfaces if and when required.

5.4 Putting Unused Ports in an Unused VLAN

Another approach to better secure our switches is to put unused ports in an unused VLAN, which ensures that the port is in a whole other subnet that has no ability to intercept data from any of our useful/production subnets, including the one containing the management IP of our switch. Now if anyone is able to gain access to a port that isn't administratively shut down, they still can get to any of our production traffic. For this purpose, we create a **Null VLAN** and give it a memorable number like 999:

```

1  sw1(config)#vlan 999
2  sw1(config-vlan)#name NULL_VLAN
3  sw1(config-vlan)#end
4  sw1#sh ip int br

```

```

5  Interface          IP-Address      OK? Method Status          Protocol
6  GigabitEthernet0/0  unassigned      YES unset    up              up
7  GigabitEthernet0/1  unassigned      YES unset    administratively down down
8  GigabitEthernet0/2  unassigned      YES unset    administratively down down
9  GigabitEthernet0/3  unassigned      YES unset    administratively down down
10 Vlan1               10.1.1.5        YES manual  up              up
11 sw1#conf t
12 Enter configuration commands, one per line. End with CNTL/Z.
13 sw1(config)#int ran g0/1 - 3
14 sw1(config-if-range)#sw acc vlan 999
15 sw1(config-if-range)#end
16 sw1#sh vlan br
17
18 VLAN Name                Status    Ports
19 -----
20 1    default                active    Gi0/0
21 999  NULL_VLAN                active    Gi0/1, Gi0/2, Gi0/3

```

Now, if and when we need those ports, we can individually move those ports to a production VLAN.

Chapter 6

Voice VLANs

6.1 Voice VLANs: Introduction

Many Cisco IP phones are built with a RJ-45 Ethernet port that allows us to connect our computers to the desk phone, so that we don't need two separate cables to carry our voice and computer's data:



Figure 6.1: Cisco IP Phone Ports

In the above diagram, port 3 connects the PC to the IP phone and port 2 is connected to the switch via an Ethernet cable. Thus, the data from the computer and the voice are kept on separate VLANs. The port on the switch to which this cable from the phone (port #2) connects thus acts similar to a trunk port, where data from multiple VLANs are carried over the same cable.

This port on the switch is however, different from a trunk port and is a special type of port called a **Multiple VLAN access port**, an access port that supports 2 VLANs (if and only when one of those VLANs is a voice VLAN).

6.2 Voice VLAN Theory

There's 3 ways in which we can set up the port on the switch that connects to an IP phone:

- A traditional, Single VLAN Access Port,
- A Multi-VLAN Access port, or
- A Trunk port.

In a Single VLAN Access port, both data and voice are carried over the same VLAN. In case of Multi-VLAN Access ports, two separate VLANs are passed on the same cable to the switch - one for data and one for voice from IP phone. Finally, it's also possible to have the IP phone connected to the switch via a 802.1q Trunk port.

6.2.1 Single VLAN Access port

This is the least preferred method among the three options. It uses a normal access port on the switch. A single VLAN carries both the voice and the data from the PC. This is not good for security due to the lack of separation. This means that laptops on the network might be able to eavesdrop on the traffic for voice. Additionally, it's better for Quality of Service(QoS) because the data traffic won't starve the voice traffic and vice versa.

It might however become necessary to use an access port for special cases like software-based IP phones on laptops and/or 3rd party applications. Another case may be use for a Cisco Jabber client that uses a soft-phone. In this configuration, it's still possible to improve the QoS by using **IEEE 802.1p** markings. Using this, we can ask the IP phone to tag traffic with a priority value going in to the switch. If our switch knows to look for the priority marking, it can then give a higher priority to the voice traffic when needed since it's for a real-time application.

6.2.2 Multi-VLAN Access port

This is a special kind of access port which supports 2 VLANs instead of one - but only when one of the VLANs is a voice VLAN. The Cisco IP phone uses a Cisco Proprietary protocol called the **CDP (Cisco Discovery Protocol)** to learn from the switch which VLAN it belongs to, via *CDP messages*. This however, isn't compatible with the industry variant of CDP called **LLDP-MED**.

The frames themselves look a lot like **dot1q** trunk frames. The frames will also have a priority value which for voice is higher than data (typically). The voice frames will get tagged but the data frames, much like the frames for a native VLAN will still pass through untagged.

6.2.3 Trunk port

We may have to connect IP phones to trunk ports on a switch in special cases like older Cisco IP phones, or cases where we have to use the **Link Layer Discovery Protocol (LLDP)** instead of CDP, which is used by devices to announce/advertise their neighbours, identity and capabilities. This is compatible with both CDP and LLDP given that's it's a true *dot1q* trunk.

Since it's a trunk however, data for all the VLANs will flow through it, which is not desirable since it reduces separation. Thus, we need to prune all un-needed VLANs till the data for the PC and the voice from the IP phone are the only traffic on this trunk port.

6.3 Voice VLAN Configuration

6.3.1 Setting up a Single VLAN access port

We can set up a single VLAN access port by first setting a switch-port to access mode, and then choosing the appropriate VLAN for it:

```
1 sw1(config-if)#switchport mode access
2 sw1(config-if)#switchport access vlan 300
```

Now to ensure that the voice traffic is given priority, we have to set up **IEEE 802.1p** marking for priority:

```
1 sw1(config-if)#switchport voice vlan dot1p
```

Now the switch-port knows that whenever any traffic comes in with a **dot1p** marking, it's part of our voice VLAN and a priority value/marking can be embedded inside those tag bytes. Now given that our maximum frame size on Ethernet (for non-jumbo frames) can be 1518 bytes with 1500B of data and 18B of Layer 2 header, the additional 4B for the dot1p marking would typically be discarded by the switch as a *giant*. However, the switch can be trained to accept these frames, each of $1518 + 4 = 1522\text{B}$, called **baby giants** by enabling *dot1p*.

6.3.2 Setting up a Multi-VLAN access port

Now we can have two separate VLANs: an data VLAN and a voice VLAN on the same port without trunking. Hence, the mode of the port still has to be set as an access port:

```
1 sw1(config-if)#switchport mode access
2 sw1(config-if)#switchport access vlan 300
3 sw1(config-if)#switchport voice vlan 400
4 % Voice VLAN does not exist. Creating vlan 400
```

Notice we can create VLANs on the fly like this and then name them later. Now, CDPv2 (CDP *version 2*) will tell the IP phone that the VLAN for voice is 400 and for data (from the PC) is 300. **CDPv2** also tells the phone it's subnet, the subnet mask and the default gateway when it sends a DHCP request at boot. Typically another DHCP request is also sent, called a **DHCP option 150** request that the phone uses to get the IP address of a **TFTP (Trivial File Transfer Protocol)** server from which it can download all its configuration.

6.3.3 Setting up a Trunk Port for voice

In case we're not running CDP but something like **LLDP-MED (Link Layer Discovery Protocol for Media Endpoint Devices)**, we need to set this port up as a dot1q trunk (ISL won't work with this). First, we set up the switch-port as a trunk and then assign the data VLAN as the native VLAN. In our case, it'd be VLAN 300:

```
1 sw1(config-if)#switchport trunk encapsulation dot1q
2 sw1(config-if)#switchport mode trunk
3 sw1(config-if)#switchport trunk native vlan 300
4 sw1(config-if)#switchport voice vlan 400
```

We finished off by setting the voice VLAN to 400. While this configuration by itself would still work, we need to prune off unnecessary VLANs for security and performance. We do this by:

```
1  sw1(config-if)#switchport trunk allowed vlan 300,400
2  sw1(config-if)#end
3  sw1#sh int g0/1 switchport
4  Name: Gi0/1
5  Switchport: Enabled
6  Administrative Mode: trunk
7  Operational Mode: trunk
8  Administrative Trunking Encapsulation: dot1q
9  Operational Trunking Encapsulation: dot1q
10 Negotiation of Trunking: On
11 Access Mode VLAN: 1 (default)
12 Trunking Native Mode VLAN: 300 (DATA)
13 Administrative Native VLAN tagging: enabled
14 Voice VLAN: 400 (VOICE)
15 ...
16 Trunking VLANs Enabled: 300,400
17 ...
```

Now we can see that the trunk is properly set up to allow either CDP or LLDP-MED to configure the IP phone as well as carry both the data and voice.

Part II

ICND1 - Routers

Chapter 7

Basic Router Operation

7.1 Packet flow through a Routed Network

Let us consider the case below where the user/client (192.168.1.2 | AAAA.AAAA.AAAA) wants to communicate with the server (192.168.2.2 | BBBB.BBBB.BBBB). The packet goes through several devices: sw1, R1, sw2, R2 and sw3 to reach the server. The transmission can be broken down into two layers of operation: Layer 3 and Layer 2.

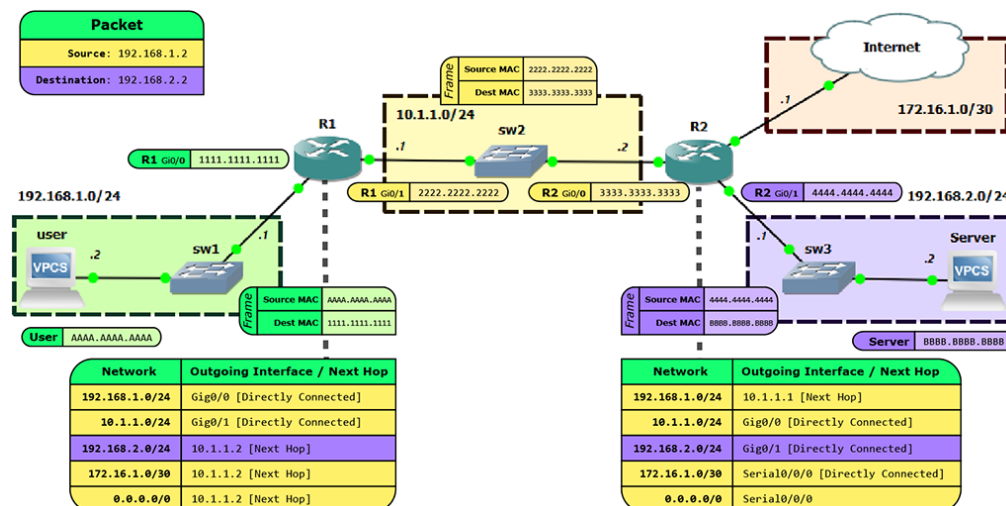


Figure 7.1: Packet Flow

7.1.1 Layer 3 Packet flow

At **Layer 3**, the *user* forms a packet with the source IP of itself, and the destination IP of the server (which is known to it). *Throughout the transmission the source and destination IPs remain unchanged.* For the communication to occur, the packet flows through three networks: 192.168.1.0/24 network, the 10.1.1.0/24 network and finally, the 192.168.2.0/24 network. Since a router separates one broadcast domain/subnet from another, there's two routers required: **R1** that links 192.168.1.0/24 and 10.1.1.0/24 networks, and **R2** that facilitates communication between 10.1.1.0/24 and 192.168.2.0/24 networks.

In the beginning, the user sends the packet to it's gateway, R1. Since switches can't see, i.e., make forwarding decisions on the basis of IPs / switch packets in layer 3, they

act more or less like direct links. Once R1 receives the packet, it searches for the destination network in its **routing table**, which is a table of interface to network mappings. It sees that the 192.168.2.0/24 network isn't directly connected to it, but is reachable via the 10.1.1.0/24 network, called the **next hop**. Now it searches its routing table for the 10.1.1.0/24 network, which is connected via the interface **Gi0/1**. This now becomes the egress port and the packet is sent out, **unchanged** to **R2** which has **R2 Gi0/0** connected to **R1 Gi0/1** via sw2.

When the ingress port of **R2 Gi0/0** gets the inbound packet destined for 192.168.2.0/24, it searches its own routing table for the destination network, which is directly connected to it via **R2 Gi0/2**. So, it sends out the packet through the egress port and the packet reaches the Server.

7.1.2 Layer 2 Packet Flow

Unlike the operation in layer 3, the source and destination *MAC addresses are re-written every time there's a change in the subnet*. This means that the user has to first send an **ARP request** to get the MAC address of R1's ingress port of **R1 Gi0/0**, which is it's gateway. The ARP packet reaches *sw1*, which forwards the ARP broadcast on all other ports but its ingress port. Router R1 replies with its MAC address, 1111.1111.1111, which gets stored in the **ARP cache** of user so that it can remember the MAC address instead of ARP-ing again. Thus, the user sends out a frame with the source MAC address of AAAA.AAAA.AAAA and destination MAC of 1111.1111.1111. Sw1 meanwhile, makes entries for both *user* and *R1* in its CAM table.

When the frame reaches the ingress port on **R1 Gi0/0** with MAC 1111.1111.1111, the router finds out that the packet has to be sent out to another network, where the source MAC of AAAA.AAAA.AAAA is *un-knowable* by the devices on the network. Thus, it replaces the source MAC with the MAC address of it's own egress port (2222.2222.2222), and sends out the packet to the MAC address 3333.3333.3333 of **R2 Gi0/0**.

Finally at R3, the source MAC is changed to 4444.4444.4444, the egress **R2 Gi0/2** port's MAC and the destination MAC is set to that of the server (BBBB.BBBB.BBBB), known to R2 (via ARP broadcast). Thus the packet reaches from the user to the server without either knowing the other's MAC address, since they're re-written by the routers at each step.

7.2 Sources of Route Information

There can be several sources of routing information. The subnets can be directly connected via an interface, There can be static routes that are hard-coded, dynamic routes learnt via a configured dynamic routing protocol or the default route to a *gateway of last resort*, when the pathway to network is unknown.

| Type | Description |
|------------------|--|
| Connected | The network (ex. 192.168.2.0/24) of the IP address assigned to an interface (ex. 192.168.2.5/24). |
| Static | Administratively i.e., manually configured route to a network. Doesn't scale very well for large number of routes. |
| Dynamic | Typically used in larger network, these routes are <i>learnt</i> by the use of one or more dynamic routing protocols. |
| Default | When the network is unknown, we use this route which typically leads to a <i>gateway of last resort</i> . This can be for all unknown networks (0.0.0.0/0) and may lead to the internet. |

The existing routes known to a route, i.e., the IP routing table can be seen by using the `show ip route` command:

```

1  r2#sh ip route
2  Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
3          D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
4          N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
5          E1 - OSPF external type 1, E2 - OSPF external type 2
6          i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
7          ia - IS-IS inter area, * - candidate default, U - per-user static route
8          o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
9          a - application route
10         + - replicated route, % - next hop override, p - overrides from PfR
11
12  Gateway of last resort is not set
13
14         10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
15  C         10.1.1.0/24 is directly connected, GigabitEthernet0/0
16  L         10.1.1.2/32 is directly connected, GigabitEthernet0/0
17         172.16.0.0/16 is variably subnetted, 2 subnets, 2 masks
18  C         172.16.1.0/24 is directly connected, GigabitEthernet0/2
19  L         172.16.1.1/32 is directly connected, GigabitEthernet0/2
20  S         192.168.1.0/24 [1/0] via 10.1.1.1
21         192.168.2.0/24 is variably subnetted, 2 subnets, 2 masks
22  C         192.168.2.0/24 is directly connected, GigabitEthernet0/1
23  L         192.168.2.1/32 is directly connected, GigabitEthernet0/1

```

7.2.1 IP Routing Table codes

The code in a routing table describes how the route was learnt. It may be C, L, S, O, etc. The meaning of some of them is:

| Code | Meaning | Description |
|----------|-----------------------------------|--|
| C | <i>Directly Connected network</i> | The network noted is connected directly via an interface. |
| L | <i>Local Interface</i> | This is the IP address configured on an interface on the router. The subnet mask length is set to /32 to indicate that this isn't a network that's connected but a single IP for the interface. |
| S | <i>Static Route</i> | This is a route that was administratively set. This may also be the gateway of last resort if the destination network is 0.0.0.0/0, i.e., all unknown networks. |
| O | <i>OSPF-Learnt Route</i> | A route acquired by the use of the Open Shortest Path First (OSPF) dynamic routing protocol. These dynamic routing entries are automatically updated based on changing network conditions. If a link goes down, the associated entry for the connected network will dynamically disappear since the network is no longer reachable. |
| D | <i>EIGRP-Learnt Route</i> | A route learnt by using the Enhanced Interior Gateway Routing Protocol (EIGRP) , which is another dynamic routing protocol. |

7.2.2 Administrative Distance

In case multiple routes are learnt to a network via multiple routing protocols, such as one learnt via OSPF and another via EIGRP, then the clash is resolved (and the route chosen) on the basis of the **Administrative Distance (AD)** of the available routes. The AD is the *believability* of each route with lower values being more believable than higher ones.

The Administrative distances for each route is linked with how it was learnt. Note that Static routes have a default AD of 1. This is so that statically configured routes are given a higher preference than dynamic routes, but it can be changed by administrators to have a value greater than that presented by a dynamic routing protocol, such that the static route acts as a backup route. The default values for administrative distances of some types of these route sources are:

| Route Source | AD | Explanation |
|----------------------|-----|---|
| Connected (C) | 0 | Most believable since the network is directly connected. |
| Static (S) | 1 | Set to such a high value to override any dynamically learned routes by default. |
| eBGP | 20 | Route advertisement from another <i>BGP-speaking Autonomous System (AS)</i> obtained via eBGP (external Border Gateway Protocol) . |
| EIGRP | 90 | |
| OSPF | 110 | |
| RIP | 120 | Routing Information Protocol (RIP) . |

7.3 IP Routing Table

A router makes all forwarding decisions based on the contents of its IP routing table. This can be seen with `show ip route`, which shows all routes injected into the table. Typical contents are:

```
1  r1#sh ip route
2  Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
3          D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
4          N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
5          E1 - OSPF external type 1, E2 - OSPF external type 2
6          i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
7          ia - IS-IS inter area, * - candidate default, U - per-user static route
8          o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
9          a - application route
10         + - replicated route, % - next hop override, p - overrides from PfR
11
12  Gateway of last resort is not set
13
14         10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
15  C         10.1.1.0/24 is directly connected, GigabitEthernet0/0
16  L         10.1.1.1/32 is directly connected, GigabitEthernet0/0
17         192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
18  C         192.168.1.0/24 is directly connected, GigabitEthernet0/1
19  L         192.168.1.1/32 is directly connected, GigabitEthernet0/1
20  S         192.168.2.0/24 [1/0] via 10.1.1.2
```

Notice that some routes have a [1/0] after their IP network address. The first number is the **AD (Administrative Distance)** which sets the believability of the route and the second number is the **metric** used by a dynamic routing protocol, that acts like a cost associated to

using that route to connect to a remote network. Since this is a statically configured route, the metric has been set to (0).

In case of multiple routes to the same network via different routes is discovered by the same routing protocol (e.g., OSPF), then the one with the lower cost, i.e., lower metric will be used. Different dynamic routing protocols use different ways of calculating the metric. Thus, EIGRP may have a metric in millions while RIP has an equivalent metric of a single digit for the same route. Thus, **metrics must never be compared between different routing protocols**.

Another important piece of information from routing tables is the next hop network that connects us to another network. For example, in this case, the static route has been configured such that the network 192.168.2.0/24 is reached via the interface connected to the network 10.1.1.2.

7.4 Packet Forwarding

The packet forwarding options in a router determine how the packets are routed from the ingress port of the router to the egress port in terms of architecture. Other than those listed below, there can be more such as *Optimum switching*, *silicon switching*, etc. In all these cases, the term *switching* refers to the packet switching or layer 3 routing of the packets from the ingress to the egress port of the router.

7.4.1 Process Switching

With **Process Switching**, the processor of the router gets involved in every packet switching decision. The router's architecture can be broadly divided into two sections: the **control plane** and the **data plane**. A third plane is sometimes referred to in literature called the *management plane* which is used when we telnet/ssh into the router, but in this case, we're only going to consider the first two.

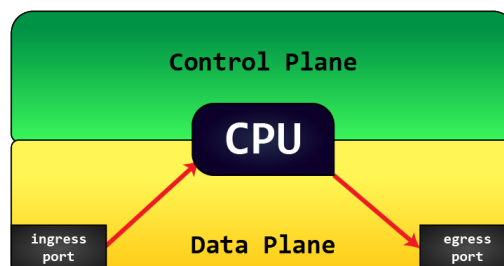


Figure 7.2: Process Switching

The **data plane** is where the traffic flows through the router and the elements of the **control plane** controls the *behind-the-scene* tasks required to control the flow through the data plane. The CPU can operate in both planes. However, if the *CPU is used in every single packet forwarding decision*, as is the case in process switching, then it can be overwhelmed, thus causing performance degradation. This is the oldest approach to packet forwarding.

7.4.2 Fast Switching

This method is considerably faster than *process switching*. Here, a new component called the **route cache** is utilized to store and remember the packet forwarding decisions made

by the CPU for known networks. In this case, the CPU is involved in packet forwarding decisions only the first time when an unknown network is detected:

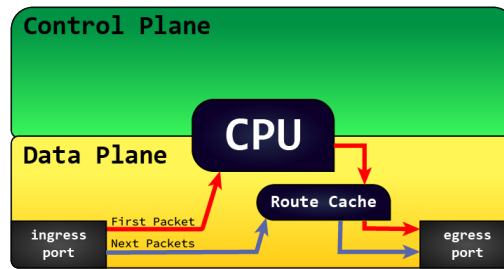


Figure 7.3: Fast Switching

When a packet for a network we haven't yet route to is detected and the router doesn't know where it should forward the packet, it asks the CPU to look in the routing table. Once the CPU has resolved the egress port, the information is stored in the *route cache* and the data is sent out the egress port. From then onwards, the route cache is consulted instead of the CPU to find the appropriate egress port. This reduces the load on the CPU, improving performance. Eventually the data in the route cache, of course, times out.

7.4.3 Cisco Express Forwarding (CEF)

Among the three, **Cisco Express Forwarding (CEF)** is the most efficient and fastest, and hence the one we should be using. With CEF, we don't have to involve the CPU even once in the packet forwarding decisions. Instead, the processor pro-actively fills up the **Forwarding Information Base (FIB)** - a construct to store the layer 3 routing info learnt from the routing table.

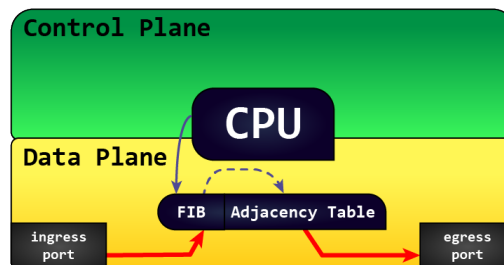


Figure 7.4: Cisco Express Forwarding (CEF)

The FIB is essentially the routing table information stored in a high-speed memory containing a very efficient table structure. This however, is only the Layer 3 information. We also need layer 2 information to quickly form a frame, which is stored in the **adjacency table**. For example, the FIB lets the router know that to go to *network A* we need to go to this *next hop ip address, B*. Then the adjacency table gives us the layer 2 info to reach *IP B* - such as the interface name, encapsulation and destination MAC address. The FIB and the adjacency table together form the CEF system. CEF also updates itself with changes in the network automatically.

Chapter 8

Basic Router Configuration and Verification

8.1 Router LEDs

A lot of info about the status of the router is conveyed via the LEDs at the front and the back of the router. While these vary by model of the router, some of the common router LED indicators are:

| LED Label | Status | Explanation |
|-------------------------------------|----------------|--|
| Sys (System) | Solid | The router is up & operational. |
| | Blinking | The router is booting. |
| | Off | The router is either turned off or there's a problem with the system board. |
| Act (Activity) | Solid/Blinking | The router is currently routing traffic. |
| | Off | No packets are currently flowing through the router. |
| PoE (Power over Ethernet) | Solid | Power over Ethernet is provided to devices that need it such as Cisco IP phones. |
| | Amber | IP Phone power is turned off. |
| RPS (Redundant Power Supply) | Solid | The router is currently being powered by an external redundant power supply. |
| | Off | Redundant power supply isn't being used. |
| PS (Power Supply) | Solid | The router is currently being powered by the internal power supply. |
| | Amber | The router isn't powered by the main power supply. |

In addition to these LEDs on the front panel, there's also LEDs on the back panel of the router, which are largely dependent upon which modules are installed. However, there are some common LEDs which are present in the rear panel:

8.1.1 GigabitEthernet port LED

The **Speed LED** (left) of the GigabitEthernet port is configured to blink in a certain pattern to indicate the speed. These are:

| Pattern | Speed |
|------------------|---------------------------------------|
| Blink + Pause | 10Mbps, i.e., Ethernet Speed |
| 2x Blink + Pause | 100Mbps, i.e., FastEthernet Speed |
| 3x Blink + Pause | 1000Mbps, i.e., GigabitEthernet Speed |

There's also the Link LED that's going to be solid green if FastEthernet or GigabitEthernet speeds are established, and off if either there's no link or the speed is 10Mbps.

8.2 Assigning an IPv4 Address to an interface

We can assign an IPv4 address to an interface the same way we added one to *VLAN1* to set the management IP address on a switch, through the interface configuration mode:

```

1 r2#sh ip int br g0/3
2 Interface IP-Address OK? Method Status Protocol
3 GigabitEthernet0/3 unassigned YES NVRAM administratively down down
4 r2#conf t
5 Enter configuration commands, one per line. End with CNTL/Z.
6 r2(config)#int g0/3
7 r2(config-if)#ip addr 172.16.5.1 255.255.255.0
8 r2(config-if)#no shut
9 *Dec 3 03:02:56.193: %LINK-3-UPDOWN: Interface GigabitEthernet0/3, changed state to up
10 *Dec 3 03:02:57.196: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/3,
    ↪ changed state to up
11 r2(config-if)#end
12 r2#sh ip int br g0/3
13 Interface IP-Address OK? Method Status Protocol
14 GigabitEthernet0/3 172.16.5.1 YES manual up up

```

8.3 Assigning an IPv6 Address to an interface

Before assigning an IPv6 Address to an interface, we should first ensure that IPv6 unicast routing is enabled on the router. We do this from the global configuration mode, using `ipv6 unicast-routing`:

```

1 r2#conf term
2 Enter configuration commands, one per line. End with CNTL/Z.
3 r2(config)#ipv6 unicast-routing

```

When it comes to the show commands, the IPv6 equivalents are almost identical to the IPv4 commands. So, instead of using `sh ip int br` we'd use `show ipv6 interface brief`:

```

1 r2#sh ipv6 int br
2 GigabitEthernet0/0 [up/up]
3 unassigned
4 GigabitEthernet0/1 [up/up]
5 unassigned
6 GigabitEthernet0/2 [up/up]
7 unassigned
8 GigabitEthernet0/3 [up/up]
9 unassigned

```

This shows that we currently don't have any IPv6 addresses assigned to any of the interfaces. However, just by enabling IPv6 on an interface, we can auto-assign it a IPv6 link local address:

```
1 r2#sh ipv6 int br g0/3
2 GigabitEthernet0/3      [up/up]
3     unassigned
4 r2#sh ipv6 int g0/3
5 r2#
6 r2#conf t
7 Enter configuration commands, one per line.  End with CNTL/Z.
8 r2(config)#int g0/3
9 r2(config-if)#ipv6 enable
10 r2(config-if)#end
11 r2#sh ipv6 int g0/3
12 GigabitEthernet0/3 is up, line protocol is up
13     IPv6 is enabled, link-local address is FE80::ED7:78FF:FE5D:FC03
14     No Virtual link-local address(es):
15     No global unicast address is configured
16     Joined group address(es):
17         FF02::1
18         FF02::2
19         FF02::1:FF5D:FC03
20     MTU is 1500 bytes
21     ICMP error messages limited to one every 100 milliseconds
22     ICMP redirects are enabled
23     ICMP unreachables are sent
24     ND DAD is enabled, number of DAD attempts: 1
25     ND reachable time is 30000 milliseconds (using 30000)
26     ND advertised reachable time is 0 (unspecified)
27     ND advertised retransmit interval is 0 (unspecified)
28     ND router advertisements are sent every 200 seconds
29     ND router advertisements live for 1800 seconds
30     ND advertised default router preference is Medium
31     Hosts use stateless autoconfig for addresses.
```

The IPv6 link-local address automatically created using the MAC address of the interface is FE80::ED7:78FF:FE5D:FC03. So now, even without specifically assigning an IPv6 address the interface can communicate with a neighbour using the local link. Additionally, we've also automatically joined a couple of multicast groups: FF02::1 which is the **all-nodes** multicast address, as well as FF02::2 which is the **all-routers** multicast group address.

However, we don't have to manually enable ipv6 on an interface when assigning an actual IPv6 address to an interface, but use `ipv6 address <ipv6Addr>/<subnet-bits>` to assign the IPv6 address. Notice that IPv6 address assignment supports the *slash notation* unlike IPv4 address assignment:

```
1 r2#conf t
2 Enter configuration commands, one per line.  End with CNTL/Z.
3 r2(config)#int g0/3
4 r2(config-if)#ipv6 address 2000::4/64
5 r2(config-if)#end
6 r2#sh ipv6 int g0/3
7 GigabitEthernet0/3 is up, line protocol is up
8     IPv6 is enabled, link-local address is FE80::ED7:78FF:FE5D:FC03
9     No Virtual link-local address(es):
10     Global unicast address(es):
11         2000::4, subnet is 2000::/64
12     Joined group address(es):
13         FF02::1
```

```

14     FF02::2
15     FF02::1:FF00:4
16     FF02::1:FF5D:FC03
17     MTU is 1500 bytes
18     ...

```

We can see that the IPv6 address 2000::4, subnet is 2000::/64 has been assigned to the interface. We can also see the IPv6 addresses using `show ipv6 interfaces brief`:

```

1  r2#sh ipv6 int br
2  GigabitEthernet0/0    [up/up]
3      unassigned
4  GigabitEthernet0/1    [up/up]
5      unassigned
6  GigabitEthernet0/2    [up/up]
7      unassigned
8  GigabitEthernet0/3    [up/up]
9      FE80::ED7:78FF:FE5D:FC03
10     2000::4

```

8.4 Basic Router Configuration

This section illustrates a brief overview of the commands involved in setting up a brand new router that has no configuration. When we first boot the router, it has a default hostname of `router` and we get the non-privileged prompt:

```

1  Router>

```

First, we'd like to set up the hostname and disable **IP domain lookup**:

```

1  Router#conf t
2  Enter configuration commands, one per line.  End with CNTL/Z.
3  Router(config)#hostname R1
4  R1(config)#no ip domain lookup

```

The IP domain lookup is a feature that assumes everything that's written at the command prompt in a non-config environment is a DNS lookup. This means that if there's ever a typo in a command, for example, a `show` command, the router incorrectly assumes that the input is a domain (DNS) lookup. Quite a bit of time waste can be saved by thus turning off the ip domain lookup feature:

```

1  R1#end
2  Translating "end"...domain server (255.255.255.255)
3  (255.255.255.255)
4  Translating "end"...domain server (255.255.255.255)
5
6  % Bad IP address or host name
7  % Unknown command or computer name, or unable to find computer address
8  R1#conf t
9  Enter configuration commands, one per line.  End with CNTL/Z.
10 R1(config)#no ip domain-lookup
11 R1(config)#end
12 R1#end

```

```
13 Translating "end"
14
15 Translating "end"
16
17 % Bad IP address or host name
18 % Unknown command or computer name, or unable to find computer address
```

8.4.1 Logging Synchronous

Next, we want to disable the time-out of our connection since this is a **Lab environment** and set a password. Finally, we'd also like to set up synchronous logging.

```
1 R1(config)#line con 0
2 R1(config-line)#exec-timeout 0
3 R1(config-line)#password cisco
4 R1(config-line)#login
```

Many times, while we're still typing a command, we get an output from the router, messing up our command:

```
1 R1(config-if)#end
2 R1#sho
3 *Dec  3 04:07:51.554: %SYS-5-CONFIG_I: Configured from console by consolew run
```

With synchronous logging, this is no longer a problem, since we get a fresh line with what we entered *after* the log message from the router. This has to be enabled from the line configuration mode:

```
1 R1(config)#line con 0
2 R1(config-line)#logging synchronous
```

This stops our input from being garbled and intermixing with the router's output.

8.4.2 Running show commands from Configuration mode

Let us assume a case where we set the ip address of an interface g0/0 and we want to verify it before assigning it to a VLAN. Normally, we'd first complete the IP assignment, exit configuration mode, check the assignment with the `sh ip int br` command, re-enter the *global* and then the *interface* configuration mode, and then finally set the VLAN. The hassle of exiting and re-entering the configuration mode can be avoided using the `do show` commands.

Whenever we want to run a show command in configuration mode, we use the `do show` command instead of the `show` variant we'd have used in privileged mode:

```
1 R1(config)#int g0/0
2 R1(config-if)#do show ip int br g0/0
3 Interface                IP-Address      OK? Method Status          Protocol
4 GigabitEthernet0/0      unassigned      YES unset  administratively down down
5 R1(config-if)#ip add 192.168.1.1 255.255.255.0
6 R1(config-if)#do show ip int br g0/0
7 Interface                IP-Address      OK? Method Status          Protocol
```

```

8  GigabitEthernet0/0          192.168.1.1      YES manual administratively down down
9  R1(config-if)#no shut
10 *Dec 3 04:23:07.690: %LINK-3-UPDOWN: Interface GigabitEthernet0/0, changed state to up
11 *Dec 3 04:23:08.689: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0,
    ↪ changed state to up
12 R1(config-if)#do show ip int br g0/0
13 Interface                    IP-Address      OK? Method Status          Protocol
14 GigabitEthernet0/0          192.168.1.1     YES manual up             up

```

8.4.3 Interface Description

Now we assign the IPv4 and IPv6 addresses. Before we can start using IPv6 addressing, we have to turn on IPv6 unicast addressing globally.

```

1  R1(config-if)#ip addr 192.168.1.1 255.255.255.0
2  R1(config-if)#exit
3  R1(config)#ipv6 unicast-routing
4  R1(config)#int g0/0
5  R1(config-if)#ipv6 address 2000::4/64
6  R1(config-if)#do show ip int br g0/0
7  Interface                    IP-Address      OK? Method Status          Protocol
8  GigabitEthernet0/0          192.168.1.1     YES manual up             up
9  R1(config-if)#do show ipv6 int br g0/0
10 GigabitEthernet0/0          [up/up]
11     FE80::EC1:8FFF:FE5A:C100
12     2000::4

```

We can also add a description for each interface so that when we're viewing the running configuration we have a description for each interface directly in the running config. For example, if we want to mark the present interface as a 'connection to R2', we use:

```

1  R1(config-if)#description Connection to R2
2  R1(config-if)#do show int g0/0
3  GigabitEthernet0/0 is up, line protocol is up
4      Hardware is iGbE, address is 0cc1.8f5a.c100 (bia 0cc1.8f5a.c100)
5      Description: Connection to R2
6  ...
7  R1(config-if)#do sh run | s GigabitEthernet0/0
8  interface GigabitEthernet0/0
9      description Connection to R2
10     ip address 192.168.1.1 255.255.255.0
11     duplex auto
12     speed auto
13     media-type rj45
14     ipv6 address 2000::4/64

```

Finally we save the current config by saving it to the starting config:

```

1  R1#copy run start
2  Destination filename [startup-config]?
3  Building configuration...
4  [OK]
5  R1#
6  *Dec 3 04:39:17.972: %GRUB-5-CONFIG_WRITING: GRUB configuration is being updated on
    ↪ disk. Please wait...
7  *Dec 3 04:39:18.914: %GRUB-5-CONFIG_WRITTEN: GRUB configuration was written to disk
    ↪ successfully.

```

Chapter 9

Routing Fundamentals

9.1 Configuring Static Routes for IPv4

There are 4 different kinds of Static routes:

| Type | Description |
|------------------------------|--|
| Static Network Route | This kind of a static route points to a network address, and is also known as a network prefix . This can be specified by either specifying the next hop ip address or the egress port. |
| Static Host Route | Similar to static network routes, this is a static route defined for a single host and thus points to a specific ip address a 32-bit subnet mask. |
| Static Default Route | This is the static route that acts as the default route or a route that the router uses when the it doesn't have more specific (or any) routing information about how to reach a certain network (i.e., gateway of last resort). |
| Floating Static Route | A static route whose Administrative Distance (AD) is set higher than the AD of a dynamic routing protocol that knows a different route to the same destination, thus making the static route less attractive and letting it act as a backup route. |

9.1.1 Static Network Route Configuration

Let us consider in the topology below, the only networks known by each router are the ones directly connected to. As such, R1 doesn't know how to reach sw2 and sw3. If we provide static routes for the 198.51.100.0/24 and the 203.0.113.0/24 networks, both switches will be reachable from R1.

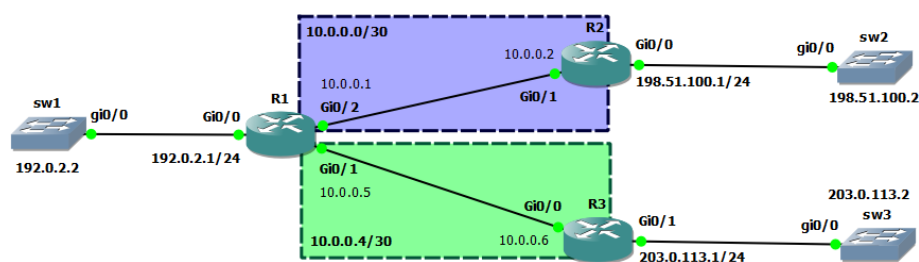


Figure 9.1: Sample Topology

Right now the routes known to **R1** are:

```
1 R1#sh ip route
2 ...
3     10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
4 C     10.0.0.0/30 is directly connected, GigabitEthernet0/2
5 L     10.0.0.1/32 is directly connected, GigabitEthernet0/2
6 C     10.0.0.4/30 is directly connected, GigabitEthernet0/1
7 L     10.0.0.5/32 is directly connected, GigabitEthernet0/1
8     192.0.2.0/24 is variably subnetted, 2 subnets, 2 masks
9 C     192.0.2.0/24 is directly connected, GigabitEthernet0/0
10 L     192.0.2.1/32 is directly connected, GigabitEthernet0/0
```

So, when we try to ping either *sw2* or *sw3*, it fails, because the router **R1** don't know how to reach the networks:

```
1 R1#ping 198.51.100.2
2 Type escape sequence to abort.
3 Sending 5, 100-byte ICMP Echos to 198.51.100.2, timeout is 2 seconds:
4 .....
5 Success rate is 0 percent (0/5)
6 R1#ping 203.0.113.2
7 Type escape sequence to abort.
8 Sending 5, 100-byte ICMP Echos to 203.0.113.2, timeout is 2 seconds:
9 .....
10 Success rate is 0 percent (0/5)
```

To make them reachable, we have to tell the R1 router how to get to the 198.51.100.0/24 and 203.0.113.0/24 networks respectively.

Static Routing with Exit Interface

We can tell the router which interface to send the traffic for a certain destination network on. It'll then perform an ARP broadcast request on that interface, get the IP addresses and MAC addresses for every device in the subnet of that interface and finally, send the packet using the destination MAC address of the appropriate device in the frame. To configure the static routing for the 198.51.100.0/24 using the exit interface GigabitEthernet0/0, we use:

```
1 R1(config)#ip route 198.51.100.0 255.255.255.0 10.0.0.2
2 R1(config)#end
3 R1#ping 198.51.100.2
4 Type escape sequence to abort.
5 Sending 5, 100-byte ICMP Echos to 198.51.100.2, timeout is 2 seconds:
6 !!!!!
7 Success rate is 100 percent (5/5), round-trip min/avg/max = 6/16/25 ms
```

The above of course, assumes that R2 knows how to route the ICMP Echo replies from *sw2* back to R1. Typically we specify the egress interface only in the case of serial interfaces, because a serial interface is a point-to-point interface, and thus there's only one possible device that's on the other end of the link.

When used with any type of Ethernet interface, since the MAC address of the destination device (**R2**) isn't known, R1 sends an ARP broadcast through this interface and queries each device on the network for their MAC addresses, thus unnecessarily growing the size of it's ARP cache/table.

An even more disastrous situation occurs when the egress port is the pathway to the gateway of last resort, i.e., the interface is used for all unknown traffic. In such cases, the ARP table can grow so large and the traffic may grow so high due to repeated ARPs, that it may cause the router to reload (*reboot*). Hence only a next hop IP should be specified for networks connected via Ethernet interfaces.

Static Routing with Next Hop IP Address

Instead of using the exit interface, we can also create static routes for both networks and hosts by specifying the next hop IP address, i.e., the name of the router that knows the path to the destination network. The next hop should of course be reachable via an associated interface. To be able to reach sw3 in the 203.0.113.0/24 network from R1, we use:

```
1 R1(config)#ip route 203.0.113.0 255.255.255.0 10.0.0.6
2 R1(config)#end
3 R1#
4 *Dec  4 04:58:46.964: %SYS-5-CONFIG_I: Configured from console by console
5 R1#ping 203.0.113.2
6 Type escape sequence to abort.
7 Sending 5, 100-byte ICMP Echos to 203.0.113.2, timeout is 2 seconds:
8 !!!!!
9 Success rate is 100 percent (5/5), round-trip min/avg/max = 13/15/20 ms
```

The consequent static routes added to be able to reach the two networks can be seen marked 'S' in the routing table:

```
1 R1#sh ip route | b Gateway
2 Gateway of last resort is not set
3
4      10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
5 C      10.0.0.0/30 is directly connected, GigabitEthernet0/2
6 L      10.0.0.1/32 is directly connected, GigabitEthernet0/2
7 C      10.0.0.4/30 is directly connected, GigabitEthernet0/1
8 L      10.0.0.5/32 is directly connected, GigabitEthernet0/1
9      192.0.2.0/24 is variably subnetted, 2 subnets, 2 masks
10 C      192.0.2.0/24 is directly connected, GigabitEthernet0/0
11 L      192.0.2.1/32 is directly connected, GigabitEthernet0/0
12 S      198.51.100.0/24 [1/0] via 10.0.0.2
13 S      203.0.113.0/24 [1/0] via 10.0.0.6
```

9.1.2 Configuring a Static Host Route

Let us consider the topology below. Here, we have two possible pathways to the 203.0.113.0/24 network: one via **R1** and another via **R2**. Let's say that we want to reach the *Server* (203.0.113.100), via R2, but all other traffic from R1 should pass through R3.

In such a case, we'd need to configure a couple of static routes - one to reach the server from R1 and another for all other traffic to reach the 203.0.113.0/24 network via R3. The latter will be a normal, static network route. After that we have to create another static host route for the server. A static host route is created with the same command, but since it's for a single host and not a network, the subnet mask has to change to 32-bit, i.e. /32 or 255.255.255.255.

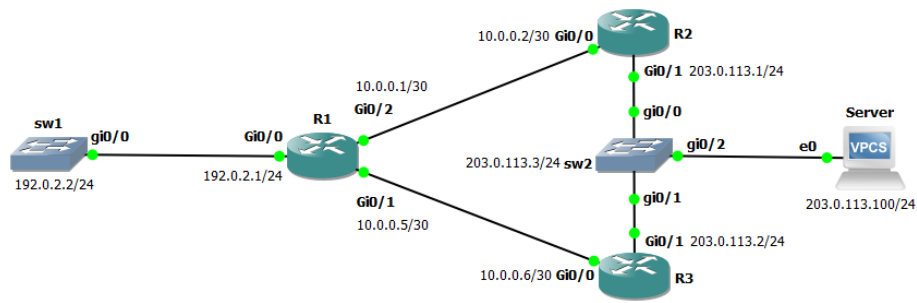


Figure 9.2: Static Host Routing topology

We can set up the network route using `ip route 203.0.113.0 255.255.255.0 10.0.0.6` and then verify that the correct route is being used using the `traceroute` command:

```

1 R1(config)#ip route 203.0.113.0 255.255.255.0 10.0.0.6
2 R1(config)#end
3 R1#ping 203.0.113.3
4 Type escape sequence to abort.
5 Sending 5, 100-byte ICMP Echos to 203.0.113.3, timeout is 2 seconds:
6 !!!!!
7 Success rate is 100 percent (5/5), round-trip min/avg/max = 11/15/21 ms
8 R1#traceroute 203.0.113.3
9 Type escape sequence to abort.
10 Tracing the route to 203.0.113.3
11 VRF info: (vrf in name/id, vrf out name/id)
12  1 10.0.0.6 9 msec 11 msec 7 msec
13  2 203.0.113.3 12 msec * 15 msec

```

Now we can create a dedicated pathway for traffic to the server using the static host route. We use the command `ip route 203.0.113.100 255.255.255.255 10.0.0.2` to create this route, since the IP of R2 is 10.0.0.2:

```

1 R1(config)#ip route 203.0.113.100 255.255.255.255 10.0.0.2
2 R1(config)#end
3 R1#ping 203.0.113.100
4 Type escape sequence to abort.
5 Sending 5, 100-byte ICMP Echos to 203.0.113.100, timeout is 2 seconds:
6 !!!!!
7 Success rate is 100 percent (5/5), round-trip min/avg/max = 16/20/29 ms
8 R1#traceroute 203.0.113.100
9 Type escape sequence to abort.
10 Tracing the route to 203.0.113.100
11 VRF info: (vrf in name/id, vrf out name/id)
12  1 10.0.0.2 11 msec 9 msec 7 msec
13  2 203.0.113.100 13 msec 12 msec 18 msec
14 R1#traceroute 203.0.113.3
15 Type escape sequence to abort.
16 Tracing the route to 203.0.113.3
17 VRF info: (vrf in name/id, vrf out name/id)
18  1 10.0.0.6 9 msec 10 msec 7 msec
19  2 203.0.113.3 12 msec * 18 msec

```

As seen in the lines 14-19 above, the normal traffic still passes through 10.0.0.6, the IP for R3 but the traffic to *Server* goes through R2. The routing table now looks like:

```

1 R1#sh ip route | b Gateway
2 Gateway of last resort is not set

```

```

3
4     10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
5 C     10.0.0.0/30 is directly connected, GigabitEthernet0/2
6 L     10.0.0.1/32 is directly connected, GigabitEthernet0/2
7 C     10.0.0.4/30 is directly connected, GigabitEthernet0/1
8 L     10.0.0.5/32 is directly connected, GigabitEthernet0/1
9     192.0.2.0/24 is variably subnetted, 2 subnets, 2 masks
10 C    192.0.2.0/24 is directly connected, GigabitEthernet0/0
11 L    192.0.2.1/32 is directly connected, GigabitEthernet0/0
12     203.0.113.0/24 is variably subnetted, 2 subnets, 2 masks
13 S    203.0.113.0/24 [1/0] via 10.0.0.6
14 S    203.0.113.100/32 [1/0] via 10.0.0.2

```

9.1.3 Keeping a route permanently in the Routing table

In the previous section, if the interface **R2 Gi0/0** were to go down, i.e., the router **R2** was no longer available, then the static host route to the *server* via R2 would also be automatically removed from the routing table, and the only route to the *server* would be through **R3**. However, there is a way to keep these static routes even when an interface/router fails, by including the keyword **permanent**. The entire command would then become:

```
ip route 203.0.113.100 255.255.255.255 10.0.0.2 permanent
```

9.1.4 Static Default Route Configuration

For the configuration of a default route, let us consider the following topology. Here, there's a router at the head-quarters of an office, marked **HQ** and another router at a branch, called **BR1**. There's a switch that connects others devices to BR1, but the only way/route out of the network or out to the internet for BR1 is via the **HQ** router:

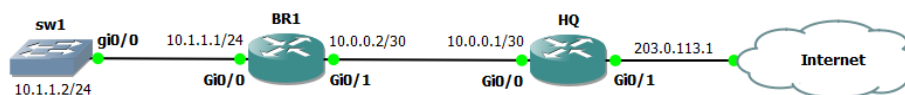


Figure 9.3: Static Default Route Topology

If the device isn't local, packets from **BR1** have to go through HQ to get there. Thus, we instead of configuring lots of static routes for all the other routers in the company (i.e., every router connected to **HQ**) or trying to *learn* (i.e., store) all of the routes in the BGP routing tables of the internet, we can simplify the configuration by making the router **HQ** the default gateway for BR1.

This is similar to when a host PC on a network gets the network info from a DHCP server and connects to its gateway router. It only sends all the relevant packets that are outside the local network to the gateway to route appropriately. The address for the default route is always **0.0.0.0**. Thus, we can set up a route for it using:

```

1 BR1#ping 203.0.113.1
2 Type escape sequence to abort.
3 Sending 5, 100-byte ICMP Echos to 203.0.113.1, timeout is 2 seconds:
4 ..
5 Success rate is 0 percent (0/2)
6 BR1#conf t
7 Enter configuration commands, one per line. End with CNTL/Z.

```

```

8  BR1(config)#ip route 0.0.0.0 0.0.0.0 10.0.0.1
9  BR1(config)#end
10 BR1#ping 203.0.113.1
11 Type escape sequence to abort.
12 Sending 5, 100-byte ICMP Echos to 203.0.113.1, timeout is 2 seconds:
13 !!!!!
14 Success rate is 100 percent (5/5), round-trip min/avg/max = 5/5/7 ms

```

As seen above the pings to the IP 203.0.113.1 which is outside the address space of the local links of **BR1** succeed once the **HQ** router has been configured as the default gateway. This is also called the **gateway of last resort** and the IP routing table now looks like:

```

1  BR1#sh ip route
2  Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
3          D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
4          N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
5          E1 - OSPF external type 1, E2 - OSPF external type 2
6          i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
7          ia - IS-IS inter area, * - candidate default, U - per-user static route
8          o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
9          a - application route
10         + - replicated route, % - next hop override, p - overrides from PfR
11
12 Gateway of last resort is 10.0.0.1 to network 0.0.0.0
13
14 S*    0.0.0.0/0 [1/0] via 10.0.0.1
15       10.0.0.0/8 is variably subnetted, 4 subnets, 3 masks
16 C      10.0.0.0/30 is directly connected, GigabitEthernet0/1
17 L      10.0.0.2/32 is directly connected, GigabitEthernet0/1
18 C      10.1.1.0/24 is directly connected, GigabitEthernet0/0
19 L      10.1.1.1/32 is directly connected, GigabitEthernet0/0

```

The * after S in the line S* 0.0.0.0/0 [1/0] via 10.0.0.1 tells us that not only is this a statically configured route but also the route to all other networks in the rest of the world. This is called the **Candidate Default** route.

9.1.5 Floating Static Route

Let us consider the following scenario, where the Router R1 has two paths to the internet, one through R2 and another through R3. Let us also consider we're running **Routing Information Protocol (RIP)** and a dynamic route to the internet was obtained via R3. As a backup, we want to set a static route through R2, but we don't want it to be used unless the route through R3 is down. We can do this by setting a greater **AD (Administrative Distance)** that the path obtained through RIP.

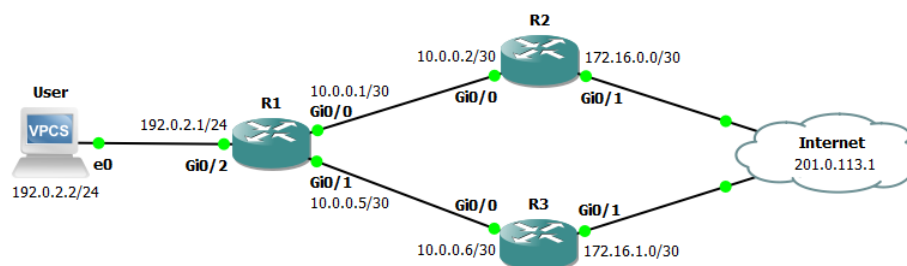


Figure 9.4: Floating Static Route

By default RIP has an AD of **120** and thus we can make our static route less desirable by setting any value greater than 120. Thus, we can create our floating static route by:

```
1 R1(config)#ip route 203.0.113.0 255.255.255.0 10.0.0.6 125
2 R1(config)#end
3 R1#sh ip route | b Gateway
4 Gateway of last resort is not set
5
6     10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
7 C       10.0.0.0/30 is directly connected, GigabitEthernet0/0
8 L       10.0.0.1/32 is directly connected, GigabitEthernet0/0
9 C       10.0.0.4/30 is directly connected, GigabitEthernet0/1
10 L      10.0.0.5/32 is directly connected, GigabitEthernet0/1
11     192.0.2.0/24 is variably subnetted, 2 subnets, 2 masks
12 C      192.0.2.0/24 is directly connected, GigabitEthernet0/2
13 L      192.0.2.1/32 is directly connected, GigabitEthernet0/2
14 S      203.0.113.0/24 [120/0] via 10.0.0.2
```

Notice that the floating route isn't displayed in the routing table because a better route to the destination network is known via RIP. Thus, only the best known route to a network is displayed in the IP routing table. However, if the best route, i.e., the one through R2 were to go down, for example, during a link failure, a new static route will be applied which will be our floating route:

```
1 R1(config)#int g0/0
2 R1(config-if)#shut
3 R1(config-if)#end
4 *Dec  4 11:58:59.430: %SYS-5-CONFIG_I: Configured from console by console
5 *Dec  4 11:58:59.809: %LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to
   ↪ administratively down
6 *Dec  4 11:59:00.809: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0,
   ↪ changed state to down
7 R1#sh ip route | b Gateway
8 Gateway of last resort is not set
9
10     10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
11 C       10.0.0.4/30 is directly connected, GigabitEthernet0/1
12 L       10.0.0.5/32 is directly connected, GigabitEthernet0/1
13     192.0.2.0/24 is variably subnetted, 2 subnets, 2 masks
14 C      192.0.2.0/24 is directly connected, GigabitEthernet0/2
15 L      192.0.2.1/32 is directly connected, GigabitEthernet0/2
16 S      203.0.113.0/24 [125/0] via 10.0.0.6
```

9.2 Configuring Static Routes for IPv6

Similar to the last section, we'll set up the IPv6 static routes for the 2004::/64 network via **R2** and for the 2005::/64 network via **R3**. Note that unlike IPv4, IPv6 addresses don't need a subnet mask in dotted notation and work fine with the *CIDR-style* subnet mask bit-lengths.

We have the two networks 2004::/64 and 2005::/64 hanging off of routers **R2** and **R3** respectively, which are both connected to **R1**. The topology looks like:

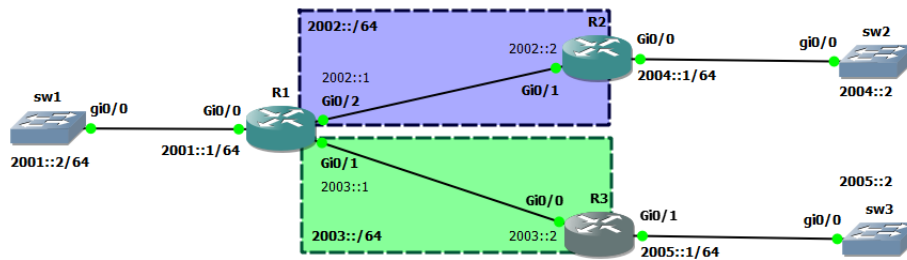


Figure 9.5: IPv6 Static Network Route

9.2.1 IPv6 Static Network Routes

IPv6 Static Routing with Exit Serial Interface

Just like in the case of IPv4 routing, the exit interface should only be used with serial interfaces since otherwise there's a chance of overflowing the ARP table, especially when the exit interface also leads to the default gateway. The set-up procedure is quite similar to the IPv4 static routing with an exit interface:

```
1 R1(config)#ipv6 route 2004::/64 s0/2
2 R1(config)#do sh ipv6 route | s 2004
3 S    2004::/64 [1/0]
4     via GigabitEthernet0/2, directly connected
```

If the above were to be used with an Ethernet interface, it wouldn't work at all. This is because Ethernet is a multi-access network and not a point-to-point connection. Plus, IPv6 doesn't have broadcast, but all nodes multicast. While for all practical purposes this may be the same, the Address Resolution Protocol will still fail due to the lack of broadcast - which is why **IPv6 doesn't use ARP**. To make it work, we need to include the next hop IP address *anyway* and use:

```
1 R1(config)#no ipv6 route 2004::/64 g0/2
2 R1(config)#ipv6 route 2004::/64 g0/2 2002::2
3 R1(config)#end
4 R1#ping 2004::1
5 Type escape sequence to abort.
6 Sending 5, 100-byte ICMP Echos to 2004::1, timeout is 2 seconds:
7 !!!!!
8 Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/8 ms
9 R1#traceroute 2004::1
10 Type escape sequence to abort.
11 Tracing the route to 2004::1
12
13 1 2002::2 8 msec 5 msec 8 msec
```

IPv6 Static Routing with Next Hop Address

Again, this is quite straightforward, just like in IPv4. The only thing that changes is the format of the IP address which adheres to the IPv6 address formatting:

```
1 R1(config)#ipv6 route 2005::/64 2003::2
2 R1(config)#do sh ipv6 route | s 2005
```

```

3 S 2005::/64 [1/0]
4   via 2003::2
5 R1(config)#do ping 2005::1
6 Type escape sequence to abort.
7 Sending 5, 100-byte ICMP Echos to 2005::1, timeout is 2 seconds:
8 !!!!!
9 Success rate is 100 percent (5/5), round-trip min/avg/max = 2/3/6 ms

```

IPv6 Static Routing with Link Local Address

All interfaces running IPv6, of course get a *link-local* address in addition to any other IPv6 address we may assign to it. To set the link-local address as the IPv6 address, we need to specify which interface it's connected to as well. This is because a link-local address is *only guaranteed to be unique on the local link*. This means it would violate no IPv6 rules if the same link-local address may show up across different links. The link local address for **R2 g0/1** can be obtained using:

```

1 R2#sh ipv6 int br
2   GigabitEthernet0/0      [up/up]
3       FE80::EC1:8FFF:FEC5:C700
4       2004::1
5   GigabitEthernet0/1      [up/up]
6       FE80::EC1:8FFF:FEC5:C701
7       2002::2
8   GigabitEthernet0/2      [administratively down/down]
9       unassigned
10  GigabitEthernet0/3      [administratively down/down]
11       unassigned
12 R2#sh int desc
13 Interface                Status      Protocol Description
14 Gi0/0                    up          up
15 Gi0/1                    up          up      Connects to R1
16 Gi0/2                    admin down  down
17 Gi0/3                    admin down  down

```

Now, we can use it to specify the static route on **R1**:

```

1 R1(config)#ipv6 route 2004::/64 FE80::EC1:8FFF:FEC5:C701
2 % Interface has to be specified for a link-local nexthop
3 R1(config)#ipv6 route 2004::/64 g0/2 FE80::EC1:8FFF:FEC5:C701
4 R1(config)#do ping 2004::1
5 Type escape sequence to abort.
6 Sending 5, 100-byte ICMP Echos to 2004::1, timeout is 2 seconds:
7 !!!!!
8 Success rate is 100 percent (5/5), round-trip min/avg/max = 3/4/5 ms

```

9.2.2 IPv6 Static Host Route

Similar to the example in the IPv4 Static Host Route section, if we want the traffic for a single host on a network (*server*) to go through **R2** but the rest to go through **R3**, then we first have to set up the IPv6 static network route through R3 and then the IPv6 static host route through **R2**. Let us consider we have the following IPs:

| | Device | IPv6 Address |
|--|----------------------------|--------------|
| | R1 Egress→R2 | 2002::1 |
| | R1→R2 Ingress | 2002::2 |
| | R2 Egress→Switch to server | 2004::1 |
| | R1 Egress→R3 | 2003::1 |
| | R1→R3 Ingress | 2003::2 |
| | R3 Egress→Switch to Server | 2004::2 |
| | Server | 2004::100 |

A host static route in IPv6 will have a **128-bit** subnet mask length to indicate it's the host's IPv6 address and not a network IPv6 Address. To set up the static routes, we use:

```
1 R1(config)#ipv6 route 2004::/64 2003::2
2 R1(config)#ipv6 route 2004::100/128 2002::2
```

9.2.3 IPv6 Static Default Route

To set up the default route, the IPv4 address of 0.0.0.0/0 becomes ::0 in IPv6. To set it up by associating it with an exit interface via serial link, we use:

```
1 BR1(config)#ipv6 route ::/0 s1/0
```

9.2.4 IPv6 Floating Static Route

For an IPv6 floating route, we simply change the IPv4 address to the IPv6 address assigned in the command. Everything else remain the same. For example, if we consider 2006::/64 to be the address of the destination network, and our backup route goes through 2002::2, with the alternate route obtained via RIP (*AD=120*), and our AD for the static route is *125*, then the command becomes:

```
1 R1(config)#ipv6 route 2006::/64 2002::2 125
```

9.3 Overview of Routing Protocols

There are several criteria that determine which routing protocol we should use. Some of these are:

9.3.1 Scalability

For tiny networks, we can get away with (and in fact, it's economical to) use static routes. For small-medium scale networks, RIP, OSPF, EIGRP and BGP - all perform well. However, when it comes to larger networks, RIP may not be such a good choice for a protocol due to its metric of **hop count**. The hop count is defined as the *number of routers that need to be crossed to reach a destination network*. **RIP** has a limit of 15-hops. This of course makes it unusable for anything that requires more routers to be connected.

OSPF and **EIGRP** contrastingly, can run very large enterprise-scale networks. **BGP** is the protocol that *runs the internet*. It is designed to interconnect multiple **Autonomous Systems (AS)**, where an AS is a network under a single administrative control. Thus, a network in an enterprise is considered an AS. Similarly, the network of the ISPs are their AS. BGP routes between these Autonomous Systems.

9.3.2 Vendor Interoperability and Familiarity

For many years, before 2013, EIGRP was a Cisco proprietary protocol, which meant other vendors didn't have it, and it couldn't be used where products from other vendors were used. This changed after Cisco made EIGRP available to all. Thus, using EIGRP is no longer an obstacle in a multi-vendor environment. Thus, the hardware has to be able to support the desired routing protocol.

Another important factor in choosing the routing protocol is the familiarity of the IT support staff with the protocol to be used. Thus, engineers who're more used to OSPF can tune it better than EIGRP for the network at hand.

9.3.3 Convergence time

The Convergence time of a dynamic routing protocol is the amount of time a protocol takes to re-route traffic when a network failure occurs. Let us consider that the primary path determined by a protocol is **R1**→**R3**→**R4**, with an alternate path available through **R1**→**R2**→**R4**. Then, if the primary router **R3** has a link failure, then the time taken by the protocol to switch over to the alternate path through **R2** is called the convergence time. Typically, OSPF and EIGRP have convergence time of just a few seconds, but protocols such as RIP and BGP can have a convergence time in the order of a minutes.

9.3.4 Summarization

This is a feature of routing protocols that allow multiple routes to be *summarized*, i.e., represented with a single summary route. Let us consider a router **R1** that has routes for 4 different subnets: 10.0.0.0/24 10.0.1.0/24, 10.0.2.0/24 and 10.0.3.0/24. These 4 networks all have a common arrangement of bits for their network bits till the 3rd octet, and thus can be represented by:

| 3rd Octet Value | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|-----------------|-----|----|----|----|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

As we can see, there's absolutely no difference up until the 22nd bit of the addresses. Thus we can represent all four networks as a single summary address of **10.0.0.0/22**, obtained by taking all the common bits in the network address and then making the host bits zeroes. This can not only reduce the size of our routing table, but can also help by reducing the load on the router given that it doesn't have to run, for example, the Dijkstra's Algorithm on a smaller number of routes.

9.3.5 Interior or Exterior Gateway Protocol

An important distinction must be made between Interior and Exterior Gateway Protocols. An **Interior Gateway Protocol (IGP)** is designed to run within an Autonomous System while an **Exterior Gateway Protocol (EGP)** is designed to run between Autonomous Systems.

So, within an Autonomous System like a company, we'll run an IGP like RIP, OSPF, EIGRP or IS-IS (Intermediate System to Intermediate System). The link between companies and their ISPs are however going to be communicating via an EGP - and BGP is practically the most common one. While there is an option of running BGP *within* an AS, typically it's used as an EGP between Autonomous Systems.

9.3.6 Protocol Classification

We have three main categories of classifying routing protocols:

- Distance Vector
- Link State
- Path Vector

Distance-Vector Routing Protocols

A Distance Vector routing protocol periodically sends a full copy of its routing table to its directly connected neighbours, regardless of network changes. Thus, a copy is still sent even if no changes have occurred. Routing Information Protocol (RIP) is an example of a Distance Vector protocol.

The data-structure of a distance-vector protocol stores two values: a distance and a vector. Thus, there's got to be a metric to tell us how far away a network is, which in the case of RIP is the **hop-count**. The *vector* is the direction or path that leads us to the destination network. This can be the exit interface or next-hop IP address, etc. Other than knowing the distance and the direction/vector, a distance-vector protocol doesn't ever have a map-of-the-terrain or an over-arching/comprehensive view of the network.

EIGRP is a form of an advanced distance-vector routing protocol that's not limited by the those of a normal distance-vector routing protocol. For example, it can send triggered updates, i.e., send an update to the adjacent neighbours when a network change occurs instead of unnecessarily sending updates periodically (e.g., every 30 secs) like RIP does.

Link State Routing Protocols

A link-state routing protocol, unlike distance-vector routing protocols, have a comprehensive view of the network. Much like a car's navigation systems/Google Maps, it knows all the nodes in the network and how they're interconnected. It can assign different values/weights/costs to different paths based on different criteria, such as bandwidth and then runs an algorithm such as **Dijkstra's Shortest Path** algorithm in case of **OSPF**, to determine the best route to a destination network. Similarly, **IS-IS** is also a link-state routing protocol.

Path Vector Routing Protocol

A Path-Vector routing protocol stores information about the exact path that packets will take to reach the destination network. **BGP** is a path-vector routing protocol and the BGP table contains a list of Autonomous Systems that we have to enter to get to the destination network and together they formulate the exact path.

9.4 Administrative Distance

The **Administrative Distance (AD)** is the trustworthiness/believability of a route. Lower values are better, i.e., more trustworthy. The AD for different methods of learning a route vary. This means that some routing protocols have lower ADs, i.e., higher believability, and are thus preferred over others. Thus, we get the following table:

| Route Source | Administrative Distance (AD) |
|--------------------|------------------------------|
| Directly Connected | 0 |
| Static Route | 1 |
| EIGRP | 90 |
| OSPF | 110 |
| RIP | 120 |
| External EIGRP | 170 |
| Unknown | 255 |

External EIGRP - External EIGRP is the case when a route has been learnt by EIGRP from an external source. This is the case when an OSPF-learnt route is *redistributed* into EIGRP.

9.5 Split Horizon

Let us consider the following network in the topology below. The first two entries in the routing table of each router are for the networks that are directly connected to the interfaces. Let us also consider they're using some kind of dynamic routing protocol to exchange routes with their neighbours. **R1** will tell **R2** and **R3** about the 172.16.1.0/24 network, **R2** will tell **R3** about the 10.1.1.0/30 network and **R1** about the 10.1.4.0/30 network and finally, **R3** will tell **R2** and **R1** about the 192.168.1.0/24 network.

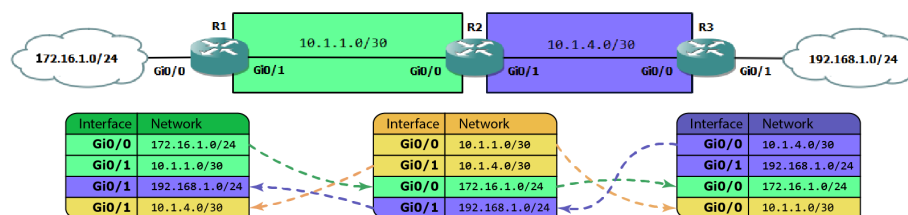


Figure 9.6: Split Horizon Rule

At this point, all routers know about all routes in the network. However, some routing protocols such as RIP and EIGRP advertise their current routing table to the neighbours. Let us consider that **R2** advertises the 172.16.1.0/24 network on both its interfaces. This is useful for **R3** since it allows **R3** to learn about the 10.1.1.0/30 network. However, the route for 172.16.1.0/24 was itself learnt from **R2**.

Originally, **R2** learnt about the 172.16.1.0/24 network from **R1** via interface **Gi0/0**. Now if the route is advertised by **R3** on port **R2 Gi0/1**, **R2** will update its routing table to

show that the 172.16.1.0/24 network is reachable from **R2 Gi0/1** instead of **R2 Gi0/0** - which is wrong! Thus, these protocols follow the **split horizon rule**. This states that a router will not advertise a route out on the interface through which it learnt the route in the first place. Thus this rule prevents routing loops and corruption of IP Routing tables.

9.6 Metric

Let us consider the following topology, where **R1** communicates to **R3** using OSPF and **R2** uses EIGRP as the IP routing protocol to speak to **R3**. The **metric** is an important factor when working in a network with multiple routing protocols. A metric is a measurement, that's IP routing protocol specific, of how far a destination network/host (i.e., node) is from a router.

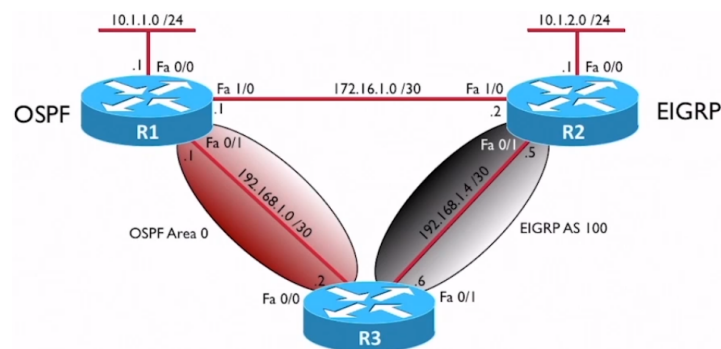


Figure 9.7: Metric

Different routing protocols use varying network parameters to calculate their metric. For example, the following table shows the different parameters used by OSPF and EIGRP to calculate their metrics:

| IP Routing Protocol | Criteria/Parameters for Metric Calculation |
|---------------------|---|
| OSPF | Bandwidth. |
| EIGRP | Bandwidth and Delay. Can also consider reliability, load and MTU (Maximum Transmission Unit) size. |

Given that the parameters for the calculation of the metric for each protocol is different as well as the methodology for the calculation is also very different, the metrics can't be compared between protocols, but only within a protocol.

9.7 Next Hop Address

A lot of routing protocols keep track of the **next-hop address**. This is the next router to which the current router needs to send the packet so that the packet can reach the intended destination. Both the routing table and the individual protocols keep track of some information about the route, such as metric, AD and next-hop IP. To view the information in the routing table when dynamically generated routes are present, we need only use the `show ip route` command and check the entries for `O`, `D`, etc.

The command to see the routing information for **OSPF** is `show ip ospf rib`, which gives us a look at the **RIB (Routing Information Base)**. In it we can see that each network known to OSPF will have an associated next-hop IP address.

The equivalent command for EIGRP's routing information is `show ip eigrp topology`, which also stores the next-hop IP. Most modern Cisco routers will use **Cisco Express Forwarding (CEF)** to store the routing information in a **FIB (Forwarding Information Base)** instead of involving the processor to lookup routes. The FIB stores the Layer 3 information in a highly efficient manner promoting quick lookups for routes.

CEF also has an adjacency table to store Layer 2 adjacency information, i.e., *next-hop* information, which include the egress interface to reach that next hop and the Layer 2 header for the MAC address of the header. We can view the CEF table using `show ip cef` which shows us both the next hop as well as the egress interface. The adjacency table can be viewed with the `show adjacency` command.

9.7.1 Using ARP to get the MAC address

Even though the router knows about the next hop's IP address, it still needs the MAC address of the ingress interface on the far end of the link so that it can form the proper Layer 2 header. To find this, it sends an **ARP (Address Resolution Protocol)** broadcast, asking for the MAC address of the next hop. Only the device with a matching IP address (contained in the ARP broadcast) replies with its MAC address, which the router then stores in its ARP Cache. The ARP cache can be viewed with the command `show ip arp`.

9.8 Passive Interfaces

9.8.1 Network Command and Wildcard Mask

Whenever we're trying to configure a dynamic routing protocol, we usually have to use the **network** command/statement in router configuration mode (IPv4). This statement defines on which network the routing protocols will be working on, and thus defines a list of interfaces that will be participating in the routing process. For OSPF/EIGRP, etc., we also have to include a **wild-card mask**. Let us consider the topology below, where we have to configure router **R1**:

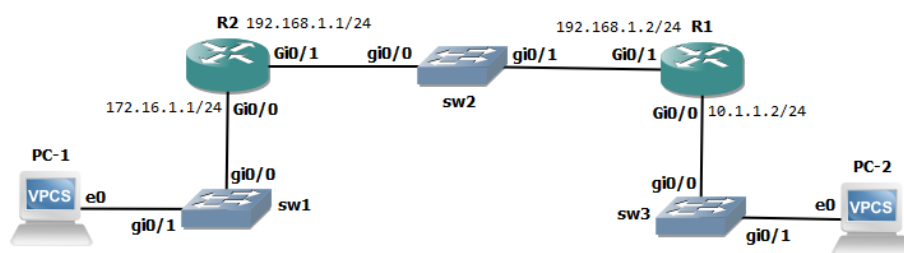


Figure 9.8: Passive Interfaces

The network statement will define the 192.168.1.0/24 network to which 192.168.1.1 (the interface's IP) belongs and the wild-card mask will be 0.0.0.255. The wild-card mask is calculated by subtracting (per octet basis) the subnet mask from 255.255.255.255. Thus, the wild-card mask for the 24-bit subnet is 0.0.0.255. The purpose of the network command is to:

- Define which interfaces will be listening for the dynamic routing protocol,
- Define which interfaces will be advertising the routes, and
- Define which networks/subnets will be advertised out.

Invariably, the only networks that'll be advertised are those that the interfaces belong to, i.e., the networks obtained from the IP address of an interface and the wild card mask for the protocol. Thus, the wild card mask is used to define which interfaces in the network we want to advertise and which we don't.

9.8.2 Need for Passive Interfaces

In the case of dynamic routing protocols, the routers form adjacency or become neighbours and start interchanging routing information. This is done by the use of **hello packets**. Each of the interfaces that are participating in the routing process (as defined by the network statement) will be sending out these *hello messages*.

In the topology above, the hello messages sent out on the **Gi0/0** interfaces of both routers: **R1** and **R2** are useless since they don't have another router on their network/subnet to utilize the hello messages because these two are end-user networks. But in protocols like OSPF and EIGRP, the hello messages are sent out on all participating interfaces by default. Further, it might be a security risk, since this means unauthorized devices may try to utilize them. So, we need to let these interfaces still participate in the routing process so that their networks can be advertised to the neighbours, but we don't want them to send out hello messages.

So, if we want the 172.16.1.0/24 network to be advertised from **R1** to **R2**, but we also don't want outgoing hello packets on **R1 Gi0/0**, then we can turn that interface into a **passive interface**. To do so, we use the `passive-interface Gi0/0` in router configuration mode.

If the router has many interfaces that are connected to end-user device/networks, then the `passive-interface default` command. This makes the router stop sending hello messages on all the interfaces. Now, to actually send hello messages to the neighbours, we can create exceptions. For example, we'd use `no passive-interface Gi0/0` to send hello messages to **R2**.

Chapter 10

Routing Information Protocol (RIP)