CCNA

ICND-1 Command Reference

Somenath Sinha

Nov 2018

Contents

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 16^{th} port on the 3^{rd} 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:

```
sw1>show line
   Tty Typ     Tx/Rx    A Modem  Roty AccO AccI   Uses   Noise  Overruns   Int
*    0 CTY              -    -      -    -    -      0       0     0/0       -
     1 AUX   9600/9600  -    -      -    -    -      0       0     0/0       -
     2 VTY              -    -      -    -    -      1       0     0/0       -
     3 VTY              -    -      -    -    -      0       0     0/0       -
     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 configuratin 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** :

```
sw1(config)#int vlan 1
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:

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

```
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:

```
sw1(config-line)#password cisco
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:

```
sw1(config-line)#login
% 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:

```
sw1(config)#line vty 0 5
sw1(config-line)#password cisco
sw1(config-line)#login
sw1(config-line)#end
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:

```
sw1#sh run
Building configuration...

Current configuration : 2911 bytes
!
! Last configuration change at 22:48:16 UTC Sun Nov 11 2018
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
service compress-config
!
hostname sw1
...
!
interface Vlan1
 ip address 192.168.1.11 255.255.255.0
```

```
!
ip default-gateway 192.168.1.1
ip forward-protocol nd
!
...
!
line con 0
 exec-timeout 0 0
 password cisco
 login
line aux 0
line vty 0 4
 password cisco
 login
line vty 5
 password cisco
 login
line vty 6
 login
!
!
end

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:

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

```
sw1#sh line vty 0
   Tty Typ     Tx/Rx    A Modem  Roty AccO AccI   Uses   Noise  Overruns   Int
     2 VTY             -    -      -    -    -      1       0     0/0       -

Line 2, Location: "", Type: ""
Length: 24 lines, Width: 80 columns
Baud rate (TX/RX) is 9600/9600
Status: Ready, No Exit Banner
Capabilities: none
Modem state: Ready
Group codes:    0
Special Chars: Escape  Hold  Stop  Start  Disconnect  Activation
                ^^x    none   -     -       none
Timeouts:      Idle EXEC    Idle Session   Modem Answer  Session   Dispatch
               00:10:00        never                       none     not set
                            Idle Session Disconnect Warning
                              never
                            Login-sequence User Response
                             00:00:30
                            Autoselect Initial Wait
                              not set
Modem type is unknown.
Session limit is not set.
Time since activation: never
Editing is enabled.
History is enabled, history size is 20.
DNS resolution in show commands is enabled
Full user help is disabled
Allowed input transports are lat pad telnet rlogin nasi ssh.
Allowed output transports are lat pad telnet rlogin nasi ssh.
Preferred transport is lat.
Shell: enabled
Shell trace: off
No output characters are padded
No special data dispatching characters
```

The two lines of special interest here are:

```
Allowed input transports are lat pad telnet rlogin nasi ssh.
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:

```
sw1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw1(config)#line vty 0 6

sw1(config-line)#transport input ?
  all     All protocols
  lat     DEC LAT protocol
  nasi    NASI protocol
  none    No protocols
```

```
  pad     X.3 PAD
  rlogin  Unix rlogin protocol
  ssh     TCP/IP SSH protocol
  telnet  TCP/IP Telnet protocol
  udptn   UDPTN async via UDP protocol
```

To only allow telnet and SSH, we use:

```
sw1(config-line)#transport input telnet ssh
sw1(config-line)#end
sw1#sh line vty 0
...
Allowed input transports are telnet ssh.
Allowed output transports are lat pad telnet rlogin nasi ssh.
...
```

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:

```
sw2>telnet 192.168.1.11
Trying 192.168.1.11 ... Open
...
sw1>sh line
   Tty Typ     Tx/Rx    A Modem  Roty AccO AccI   Uses   Noise  Overruns   Int
*    0 CTY              -    -       -    -    -      0       0     0/0       -
     1 AUX   9600/9600  -    -       -    -    -      0       0     0/0       -
*    2 VTY              -    -       -    -    -      2       0     0/0       -
     3 VTY              -    -       -    -    -      0       0     0/0       -
     4 VTY              -    -       -    -    -      0       0     0/0       -
...
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:

```
sw1#show history
  sh line
  conf t
  en
  conf t
  sh run
  sh ver | i up
  sh line status
  sh line
  sh controllers gigabitEthernet 0/0
  sh cont g 0/0 tab
  sh cont g 0/0 tabular
  sh diag
  sh line
  sh line vty 0
  conf t
  sh line vty 0
  sh ip int br
  sh run
  sh mem
  sh history
sw1#
```

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

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

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.

```
sw1#sh ip int brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0     unassigned      YES unset  up                    up
GigabitEthernet0/1     unassigned      YES unset  up                    up
GigabitEthernet0/2     unassigned      YES unset  up                    up
GigabitEthernet0/3     unassigned      YES unset  up                    up
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:

```
sw1(config)#username cisco password cisco
sw1(config)#ip domain-name somuvmnet.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 $\geq$ 768 bits for SSH 2.

```
sw1(config)#crypto key generate rsa
The name for the keys will be: sw1.somuvmnet.com
Choose the size of the key modulus in the range of 360 to 4096 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 0 seconds)

sw1(config)#
*Nov 12 01:57:14.624: %SSH-5-ENABLED: SSH 1.99 has been enabled
sw1(config)#
```

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

```
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*.

```
sw1(config)#line vty 0 6
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:

```
sw2#ssh -l cisco 192.168.1.11
...
Password:
...
sw1>exit

[Connection to 192.168.1.11 closed by foreign host]
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).

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

sw2#
```

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

```
sw1#sh run
Building configuration...

Current configuration : 3169 bytes
!
! Last configuration change at 02:12:13 UTC Mon Nov 12 2018
!
version 15.2
...
hostname sw1
...
!
username cisco password 0 cisco
...
!

sw1#sh run | i username
username cisco password 0 cisco
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.

```
sw1#sh run
Building configuration...
...
username cisco password 0 cisco
no aaa new-model
!
/vty
filtering...
line vty 0 4
 password cisco
 login local
 history size 50
 transport input telnet ssh
line vty 5
 password cisco
 login local
 history size 50
 transport input telnet ssh
!
end

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:

```
sw1#sh mac address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    0cee.bab5.5f00    DYNAMIC     Gi0/0
```

```
   1    0cee.bacd.5000    DYNAMIC     Gi0/1
   1    0cee.bacd.5001    DYNAMIC     Gi0/0
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:

```
sw1#show mac address-table aging-time
Global Aging Time:  300
Vlan    Aging Time
----    ----------
```

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:

```
sw1(config)#mac address-table static a820.6332.0087 vlan 1 interface gi 0/3
sw1(config)#end
sw1#sh mac address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    0cee.bab5.5f00    DYNAMIC     Gi0/0
   1    0cee.bacd.5000    DYNAMIC     Gi0/1
   1    0cee.bacd.5001    DYNAMIC     Gi0/0
   1    a820.6332.0087    STATIC      Gi0/3
Total Mac Addresses for this criterion: 4
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:

```
sw(config)#hostname sw1
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:

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

```
sw1#sh run | i cisco
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:

```
sw1(config)#enable secret ?
  0      Specifies an UNENCRYPTED password will follow
  5      Specifies a MD5 HASHED secret will follow
  8      Specifies a PBKDF2 HASHED secret will follow
  9      Specifies a SCRYPT HASHED secret will follow
  LINE   The UNENCRYPTED (cleartext) 'enable' secret
  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 it's 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:

```
sw1(config)#enable secret somu
sw1(config)#end
sw1#sh run | i enable
enable secret 5 $1$/Rxb$NofwEzG6FNEc2v9TiU1VA1
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` :

```
sw1(config)#enable algorithm-type ?
  md5     Encode the password using the MD5 algorithm
  scrypt  Encode the password using the SCRYPT hashing algorithm
  sha256  Encode the password using the PBKDF2 hashing algorithm

sw1(config)#enable algorithm-type sha256 secret cisco
sw1(config)#end
sw1#sh run | i enable
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:

```
sw1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw1(config)#line con 0
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.

```
sw1#sh run | s line con
line con 0
 exec-timeout 0 0
 password cisco
 login
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:

```
sw1(config)#service password-encryption
sw1(config)#end
sw1#sh run | s line con
line con 0
 exec-timeout 0 0
 password 7 070C285F4D06
 login
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:

```
sw1(config)#username somu privilege 15 password cisco
sw1(config)#end
sw1#sh run | i somu
username somu privilege 15 password 0 cisco
sw1#
```

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

```
sw1(config)#serv password-encryption
sw1(config)#username somu privilege 15 password cisco
sw1(config)#end
sw1#sh run | i somu
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:

```
sw1(config)#username somu algorithm-type scrypt secret cisco
sw1(config)#end
sw1#sh run | i somu
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:

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

```
sw1(config)#banner ?
  LINE            c banner-text c, where 'c' is a delimiting character
  config-save     Set message for saving configuration
  exec            Set EXEC process creation banner
  incoming        Set incoming terminal line banner
  login           Set login banner
  motd            Set Message of the Day banner
  prompt-timeout  Set Message for login authentication timeout
  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:

```
sw1(config)#banner login `
Enter TEXT message.  End with the character '`'.
+-------------------------+
| sw1 :      [ Swtich 1 ] +------------>
| Authorized Access Only! +--------------->
+-------------------------+
`
sw1(config)#end
sw1#sh run | b banner login
banner login ^C
+-------------------------+
| sw1 :      [ Swtich 1 ] +------------>
| Authorized Access Only! +--------------->
+-------------------------+
^C
!
```

During the login sessions, the banner will look like:

```
sw1 con0 is now available
Press RETURN to get started.

+-------------------------+
| sw1 :      [ Swtich 1 ] +------------>
| Authorized Access Only! +--------------->
+-------------------------+

User Access Verification
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:

```
sw1(config)#int g 0/1
sw1(config-if)#speed ?
  10    Force 10 Mbps operation
  100   Force 100 Mbps operation
  1000  Force 1000 Mbps operation
  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.

```
sw1(config-if)#duplex ?
  auto  Enable AUTO duplex configuration
  full  Force full duplex operation
  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`:

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

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

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

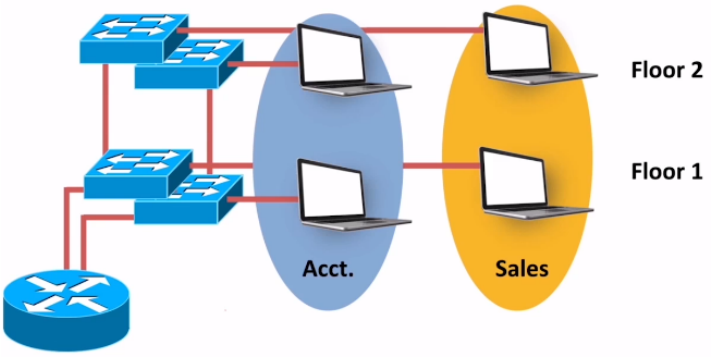

Figure 2.1: Without VLANs

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.

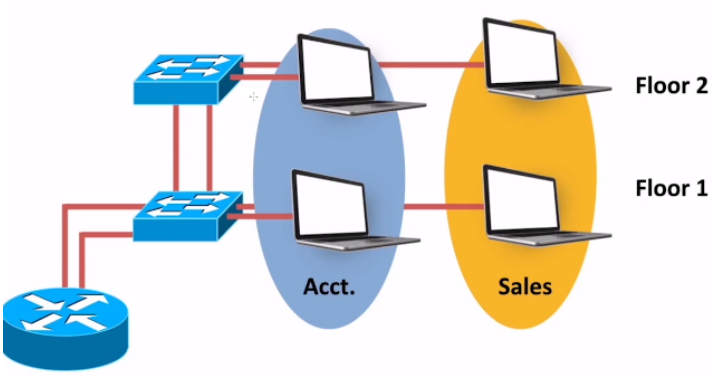

Figure 2.2: With VLANs

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 PC_1 connected to $VLAN_{100}$, and PC_2 connected to $VLAN_{200}$, 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).

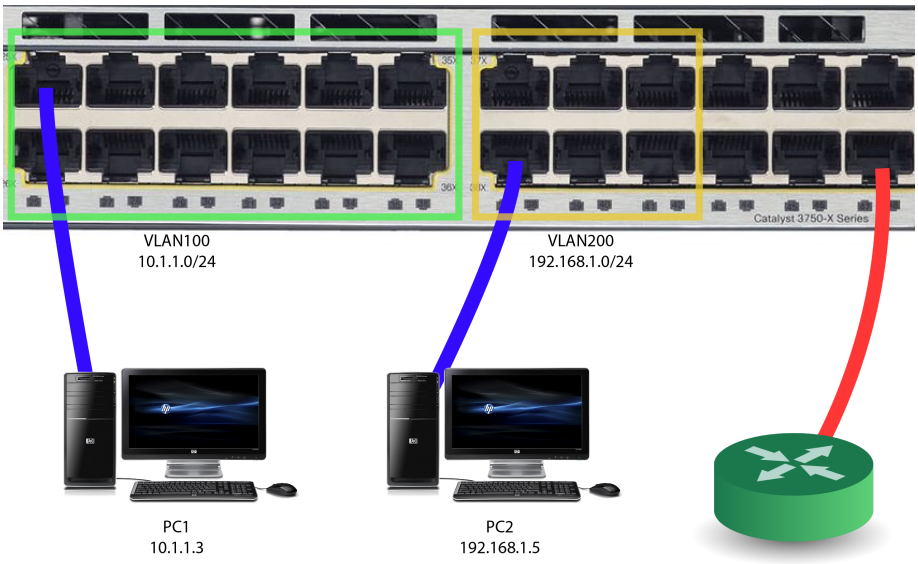

Figure 2.3: Routing between VLANs

When data has to go from VLAN_{100} to VLAN_{200}, 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 VLAN_{100}, 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 VLAN_{200} (`192.168.1.0/24`), back across the trunk and then the switching fabric to reach the **egress port** in VLAN_{200}.

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:

```
sw1#sh vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi0/0, Gi0/1, Gi0/2, Gi0/3
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
```

The fddi, token ring VLANs can be ignored. The main VLAN is the default, VLAN_1.

2.3.2 Creating a new VLAN

Let us say we want a couple of new VLANs - VLAN_{100} for the accounts department and VLAn_{200} 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:

```
sw1(config)#vlan 100
sw1(config-vlan)#name ACCT
sw1(config-vlan)#exit
sw1(config)#vlan 200
sw1(config-vlan)#name SALES
sw1(config-vlan)#end
sw1#sh vlan br

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi0/0, Gi0/1, Gi0/2, Gi0/3
100  ACCT                             active
200  SALES                            active
...
```

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:

```
sw1(config)#vlan 300
sw1(config-vlan)#name TEST
sw1(config-vlan)#end
sw1#sh vlan br | i TEST
300  TEST                             active
sw1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw1(config)#no vlan 300
sw1(config)#end
sw1#sh vlan br | i TEST
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:

```
sw1(config)#int gi0/0
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 teh *interface* config mode. To put all ports between Gi0/1 and Gi0/3 in VLAN 300, we use:

```
sw1(config)#interface range gi0/1 - 2
sw1(config-if-range)#switchport access vlan 200
sw1(config-if-range)#end
sw1#sh vlan br

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi0/3
100  ACCT                             active    Gi0/0
200  SALES                            active    Gi0/1, Gi0/2
...
```

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

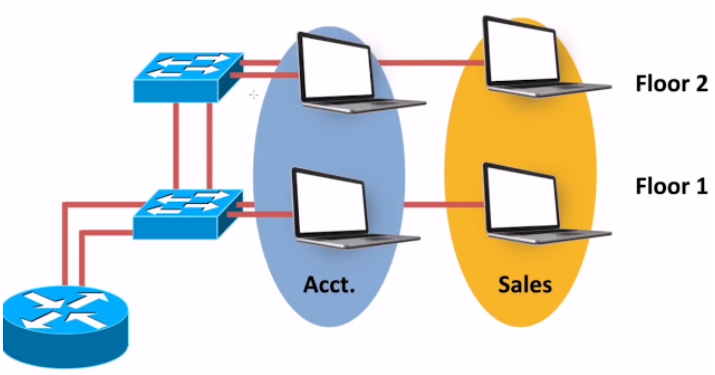

Figure 3.1: Using Access Ports

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

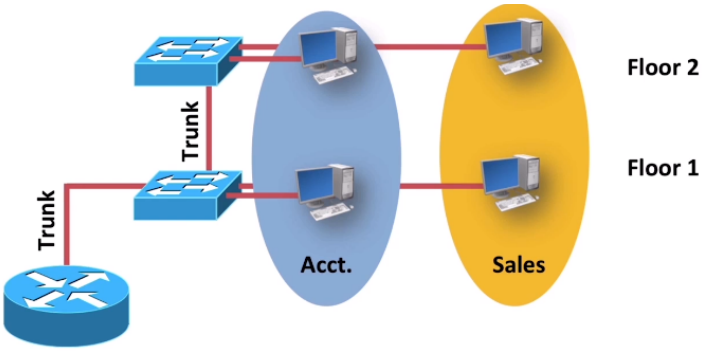

Figure 3.2: Using Trunk Ports

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:

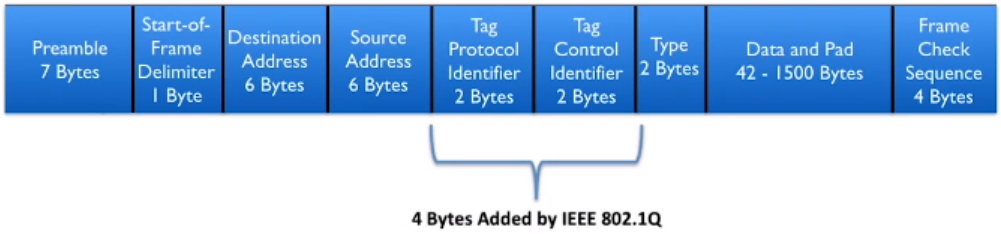

Figure 3.3: 802.1q Frame

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	*ANY*	**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:

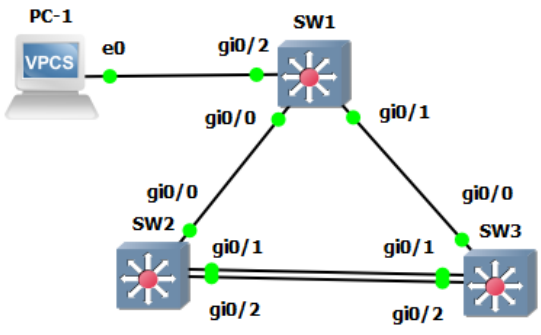

Figure 3.4: Trunk Topology

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 Chaning 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:

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

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

```
sw1(config-if)#switchport mode ?
  access        Set trunking mode to ACCESS unconditionally
  dot1q-tunnel  set trunking mode to TUNNEL unconditionally
  dynamic       Set trunking mode to dynamically negotiate access or trunk mode
  private-vlan  Set private-vlan mode
  trunk         Set trunking mode to TRUNK unconditionally

sw1(config-if)#switchport mode dynamic ?
  auto       Set trunking mode dynamic negotiation parameter to AUTO
  desirable  Set trunking mode dynamic negotiation parameter to DESIRABLE

sw1(config-if)#switchport mode dynamic desirable
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:

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

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

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:

```
sw2#sh int trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       auto             n-802.1q       trunking      1
...
sw2#
*Nov 21 08:03:44.500: %CDP-4-NATIVE_VLAN_MISMATCH: Native VLAN mismatch discovered on
↪  GigabitEthernet0/0 (1), with sw1.somuvmnet.com GigabitEthernet0/0 (100).
sw2#sh int g0/0 sw
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: trunk
Administrative Trunking Encapsulation: negotiate
Operational Trunking Encapsulation: dot1q
Negotiation of Trunking: On
Access Mode VLAN: 1 (default)
Trunking Native Mode VLAN: 1 (default)
```

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

```
sw2#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw2(config)#int g0/0
sw2(config-if)#sw t en dot
sw2(config-if)#sw t nat v 100
sw2(config-if)#end
sw2#sh int tr

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       auto             802.1q         trunking      100
...
sw2#
*Nov 21 08:08:07.859: %SPANTREE-2-UNBLOCK_CONSIST_PORT: Unblocking GigabitEthernet0/0 on
↪  VLAN0001. Port consistency restored.
sw2#
```

3.4.5 Hard-coded Trunk mode

Let us quickly configure *sw3* with the following:

```
sw3#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw3(config)#int g0/0
sw3(config-if)#sw mo dyn auto
sw3(config-if)#sw tr encap dot
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:

```
sw1#sh int tr

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       desirable        802.1q         trunking      100
Gi0/1       on               802.1q         trunking      200

Port        Vlans allowed on trunk
Gi0/0       1-4094
Gi0/1       1-4094

Port        Vlans allowed and active in management domain
Gi0/0       1,100,200
Gi0/1       1,100,200

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       1,100,200
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`:

```
sw1#sh interfaces g0/0 trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       desirable        802.1q         trunking      100

Port        Vlans allowed on trunk
Gi0/0       1-4094

Port        Vlans allowed and active in management domain
Gi0/0       1,100,200

Port        Vlans in spanning tree forwarding state and not pruned
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 VLAN_{100} over the interface `Gi0/0` and the VLAN_{200} 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.

```
sw1(config)#int g0/0
sw1(config-if)#sw tr allowed vlan 1,100
sw1(config-if)#end
sw1#sh int g0/0 tr

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       desirable        802.1q         trunking      100

Port        Vlans allowed on trunk
Gi0/0       1,100

Port        Vlans allowed and active in management domain
Gi0/0       1,100

Port        Vlans in spanning tree forwarding state and not pruned
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`.

```
sw1(config)#int g0/1
sw1(config-if)#sw tr allowed vlan except 100
sw1(config-if)#end
sw1#sh interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       desirable        802.1q         trunking      100
Gi0/1       on               802.1q         trunking      200

Port        Vlans allowed on trunk
Gi0/0       100
Gi0/1       1-99,101-4094

Port        Vlans allowed and active in management domain
Gi0/0       100
Gi0/1       1,200

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       100
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.

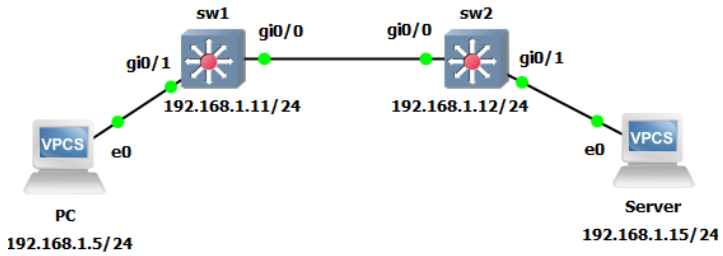

Figure 4.1: Switching Topology

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.
```

```
Sending 5, 100-byte ICMP Echos to 192.168.1.12, timeout is 2 seconds:
!!!!!
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:

```
sw1#ping 192.168.1.12
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.12, timeout is 2 seconds:
!..!!
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 fist 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:

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

```
Sending 5, 100-byte ICMP Echos to 192.168.1.12, timeout is 1 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 6/7/9 ms
sw1#sh arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.11             -   0c73.a0bb.8001  ARPA   Vlan1
Internet  192.168.1.12             0   0c73.a075.8001  ARPA   Vlan1
sw1#ping 192.168.1.12 timeout 1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.12, timeout is 1 seconds:
!!!!!
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:

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

```
sw2(config-if)#no neg auto
sw2(config-if)#speed 100
sw2(config-if)#duplex full
sw2(config-if)#end
sw2#
*Nov 26 06:38:17.399: %SYS-5-CONFIG_I: Configured from console by console
sw2#sh int status

Port      Name               Status       Vlan       Duplex  Speed Type
Gi0/0                        connected    trunk        full    100 RJ45
Gi0/1                        connected    1          a-full   auto RJ45
Gi0/2                        connected    1          a-full   auto RJ45
Gi0/3                        connected    1          a-full   auto RJ45
```

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

```
sw1#sh int g0/0 status

Port      Name               Status       Vlan       Duplex  Speed Type
Gi0/0                        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 an non-jumbo Ethernet network) and has a bad CRC.
Input Error	Sum of all interface errors (runts, giants, no buffer, CRC, frame, overrun 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 it's 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:

```
sw1#sh vlan br

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Gi0/1, Gi0/2, Gi0/3
1002 fddi-default                      act/unsup
1003 token-ring-default                act/unsup
1004 fddinet-default                   act/unsup
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:

```
sw1#sh int trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       desirable        802.1q         trunking      1

Port        Vlans allowed on trunk
Gi0/0       1

Port        Vlans allowed and active in management domain
Gi0/0       1

Port        Vlans in spanning tree forwarding state and not pruned
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 Attach

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`:

```
sw1(config-if)#sw mo access
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:

```
sw1(config-if)#sw port-sec ?
  aging        Port-security aging commands
  mac-address  Secure mac address
  maximum      Max secure addresses
  violation    Security violation mode
  <cr>

sw1(config-if)#sw port-sec max ?
  <1-4097>  Maximum addresses

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:

```
sw1(config-if)#sw port-sec mac-address ?
  H.H.H      48 bit mac address
  forbidden  Configure mac address as forbidden on this interface
  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 star` 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**:

```
sw1(config-if)#sw port-sec violation ?
  protect   Security violation protect mode
  restrict  Security violation restrict mode
  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.
Restirct	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:

```
sw1#sh int g0/1
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:

```
sw1(config)#int g0/1
sw1(config-if)#shut
sw1(config-if)#no shut
*Nov 27 03:28:01.322: %LINK-5-CHANGED: Interface GigabitEthernet0/1, changed state to
↪  administratively down
*Nov 27 03:28:04.454: %LINK-3-UPDOWN: Interface GigabitEthernet0/1, changed state to up
*Nov 27 03:28:05.456: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1,
↪  changed state to up
sw1(config-if)#end
sw1#sh int g0/1
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 `errdiabled` command:

```
sw1(config)#errdisable ?
  detect        Error disable detection
  flap-setting  Error disable flap detection setting
  recovery      Error disable recovery

sw1(config)#errdisable recovery ?
  cause     Enable error disable recovery for application
  interval  Error disable recovery timer value

sw1(config)#errdisable recovery cause ?
  all                   Enable timer to recover from all error causes
  arp-inspection        Enable timer to recover from arp inspection error
                        disable state
  bpduguard             Enable timer to recover from BPDU Guard error
```

```
  channel-misconfig     Enable timer to recover from channel misconfig error
                        (STP)
  dhcp-rate-limit       Enable timer to recover from dhcp-rate-limit error
  dtp-flap              Enable timer to recover from dtp-flap error
  gbic-invalid          Enable timer to recover from invalid GBIC error
  inline-power          Enable timer to recover from inline-power error
  l2ptguard             Enable timer to recover from l2protocol-tunnel error
  link-flap             Enable timer to recover from link-flap error
  link-monitor-failure  Enable timer to recover from link monitoring failure
  loopback              Enable timer to recover from loopback error
  mac-limit             Enable timer to recover from mac limit disable state
  oam-remote-failure    Enable timer to recover from OAM detected remote
                        failure
  pagp-flap             Enable timer to recover from pagp-flap error
  port-mode-failure     Enable timer to recover from port mode change failure
  pppoe-ia-rate-limit   Enable timer to recover from PPPoE IA rate-limit error
  psecure-violation     Enable timer to recover from psecure violation error
  psp                   Enable timer to recover from psp
  security-violation    Enable timer to recover from 802.1x violation error
  sfp-config-mismatch   Enable timer to recover from SFP config mismatch error
  storm-control         Enable timer to recover from storm-control error
  udld                  Enable timer to recover from udld error
  unicast-flood         Enable timer to recover from unicast flood error
  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:

```
sw1(config)#errdisable recovery cause psecure-violation
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:

```
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:

```
sw1#sh errdisable recovery
ErrDisable Reason            Timer Status
-----------------            --------------
arp-inspection               Disabled
bpduguard                    Disabled
channel-misconfig (STP)      Disabled
dhcp-rate-limit              Disabled
dtp-flap                     Disabled
gbic-invalid                 Disabled
inline-power                 Disabled
l2ptguard                    Disabled
link-flap                    Disabled
mac-limit                    Disabled
link-monitor-failure         Disabled
loopback                     Disabled
oam-remote-failure           Disabled
pagp-flap                    Disabled
port-mode-failure            Disabled
```

```
pppoe-ia-rate-limit          Disabled
psecure-violation            Enabled
security-violation           Disabled
sfp-config-mismatch          Disabled
storm-control                Disabled
udld                         Disabled
unicast-flood                Disabled
vmps                         Disabled
psp                          Disabled
dual-active-recovery         Disabled
evc-lite input mapping fa    Disabled
Recovery command: "clear     Disabled

Timer interval: 30 seconds

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 :

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

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

```
sw1#sh port-sec addr
               Secure Mac Address Table
-------------------------------------------------------------------------------
Vlan    Mac Address       Type                          Ports   Remaining Age
                                                                    (mins)
----    -----------       ----                          -----   -------------
   1    0050.7966.6802    SecureSticky                  Gi0/1        -
   1    0c73.a08e.5400    SecureSticky                  Gi0/1        -
-------------------------------------------------------------------------------
Total Addresses in System (excluding one mac per port)     : 1
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:

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

```
SecureStatic Address Aging : Disabled
Maximum MAC Addresses      : 2
Total MAC Addresses        : 2
Configured MAC Addresses   : 0
Sticky MAC Addresses       : 2
Last Source Address:Vlan   : 0c73.a08e.5400:1
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:

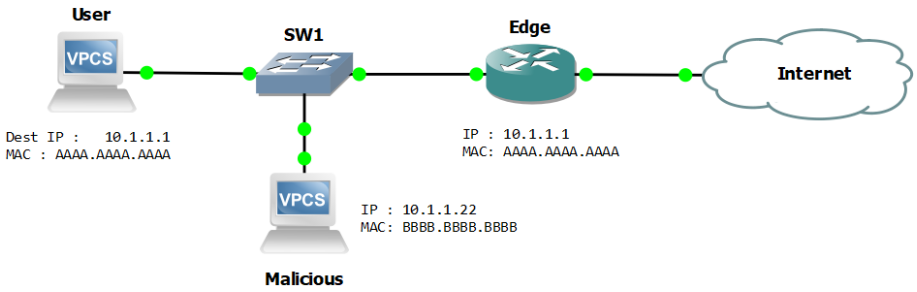

Figure 5.1: Man in the Middle attack

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:

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

```
GigabitEthernet0/0     unassigned      YES unset  up                    up
GigabitEthernet0/1     unassigned      YES unset  up                    up
GigabitEthernet0/2     unassigned      YES unset  up                    up
GigabitEthernet0/3     unassigned      YES unset  up                    up
Vlan1                  10.1.1.5        YES manual up                    up
sw1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw1(config)#int range g0/0 - 3
sw1(config-if-range)#shut
*Nov 27 05:25:07.098: %LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to
↪  administratively down
*Nov 27 05:25:07.143: %LINK-5-CHANGED: Interface GigabitEthernet0/1, changed state to
↪  administratively down
*Nov 27 05:25:07.186: %LINK-5-CHANGED: Interface GigabitEthernet0/2, changed state to
↪  administratively down
*Nov 27 05:25:07.230: %LINK-5-CHANGED: Interface GigabitEthernet0/3, changed state to
↪  administratively down
*Nov 27 05:25:08.104: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0,
↪  changed state to down
*Nov 27 05:25:08.145: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1,
↪  changed state to down
*Nov 27 05:25:08.186: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/2,
↪  changed state to down
*Nov 27 05:25:08.234: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/3,
↪  changed state to down
sw1(config-if-range)#exit
sw1(config)#int g0/0
sw1(config-if)#no shut
sw1(config-if)#end
sw1#
*Nov 27 05:25:36.362: %LINK-3-UPDOWN: Interface GigabitEthernet0/0, changed state to up
*Nov 27 05:25:37.364: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0,
↪  changed state to up
sw1#sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0     unassigned      YES unset  up                    up
GigabitEthernet0/1     unassigned      YES unset  administratively down down
GigabitEthernet0/2     unassigned      YES unset  administratively down down
GigabitEthernet0/3     unassigned      YES unset  administratively down down
Vlan1                  10.1.1.5        YES manual up                    up
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*:

```
sw1(config)#vlan 999
sw1(config-vlan)#name NULL_VLAN
sw1(config-vlan)#end
sw1#sh ip int br
```

```
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0     unassigned      YES unset  up                    up
GigabitEthernet0/1     unassigned      YES unset  administratively down down
GigabitEthernet0/2     unassigned      YES unset  administratively down down
GigabitEthernet0/3     unassigned      YES unset  administratively down down
Vlan1                  10.1.1.5        YES manual up                    up
sw1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw1(config)#int ran g0/1 - 3
sw1(config-if-range)#sw acc vlan 999
sw1(config-if-range)#end
sw1#sh vlan br

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi0/0
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 Seting 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:

```
sw1(config-if)#switchport mode access
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:

```
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 $1518 + 4 = 1522$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:

```
sw1(config-if)#switchport mode access
sw1(config-if)#switchport access vlan 300
sw1(config-if)#switchport voice vlan 400
% 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:

```
sw1(config-if)#switchport trunk encapsulation dot1q
sw1(config-if)#switchport mode trunk
sw1(config-if)#switchport trunk native vlan 300
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:

```
sw1(config-if)#switchport trunk allowed vlan 300,400
sw1(config-if)#end
sw1#sh int g0/1 switchport
Name: Gi0/1
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
Administrative Trunking Encapsulation: dot1q
Operational Trunking Encapsulation: dot1q
Negotiation of Trunking: On
Access Mode VLAN: 1 (default)
Trunking Native Mode VLAN: 300 (DATA)
Administrative Native VLAN tagging: enabled
Voice VLAN: 400 (VOICE)
...
Trunking VLANs Enabled: 300,400
...
```

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.