CCNA 200-301 :
Day 5: Ethernet LAN Switching Pt.1 :-

IP addresses are Layer3 not Layer2!

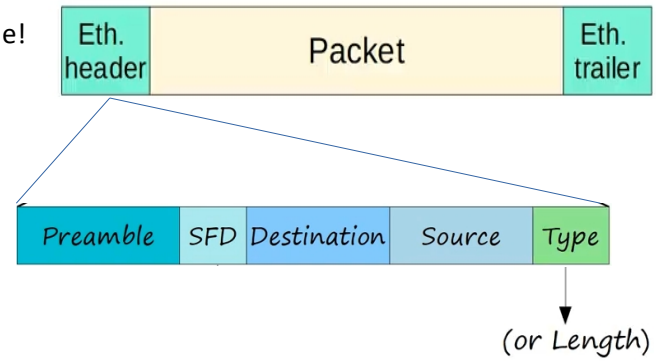
Ethernet involves Layer1 and Layer2 of the **OSI** Model.

LAN : is a network contained within a relatively small area, like an office floor or your home network!
Routers are used to connect separate LANs.
Switches do not separate LANs, but adding more can be used to expand an existing LAN.
If there are two switches connected to different router interfaces, they make two different LANs!

In this lesson: we'll look at how traffic is sent and received within a LAN:

As the Layer2 **PDU** is called a **frame**, we're going to focus on how switches receive and forward frames, specifically Ethernet frames, since it's the Layer2 protocol used in virtually every LAN in existence today!!

Encapsulating the Packet with a header and trailer, We got an Ethernet frame!



→ **Let's look at the header:** there are **5** fields in the header

SFD: Start Frame Delimiter

The **Preamble** and **SFD** are used for synchronization and to allow the receiving device to be prepared to receive the rest of the data in the frame.

Destination: the Layer2 address to which the frame is being sent.
Source: the Layer2 address of the device which sent the frame.

Type: indicates the Layer3 protocol used in the encapsulated Packet, which is almost always **Internet Protocol, IPv4** or **IPv6**. However, sometimes this is a **Length** field, indicating the length of the encapsulated data, depending on the version of Ethernet.

The **Ethernet trailer** has only 1 field, the **FCS** which stands for **Frame Check Sequence**.
FCS is used by the receiving device to detect any errors that might have occurred in transmission.

→ Let's talk in more detail:
Let's talk about the Preamble and the SFD first, which you can think of them as a **set**!

Preamble: is **7** bytes long (56 bits), and a series of **Alternating 0**'s and **1**'s like this **10101010 * 7**.
The purpose of this is that it allows devices to synchronize their receiver clocks, to make sure they're ready to receive the rest of the frame and the data inside

SFD: it's length is **1** byte (8 bits), its bit pattern is **10101011**, similar to each byte of the preamble but the last bit is a **1**.
It indicates or Mark the end of the Preamble and the beginning of the rest of the frame!

Destination and **Source**: indicate the devices sending and receiving the frame, like when you send an e-mail, both the source and destination are included in the e-mail.
The addresses are used in Ethernet are the destination and source **'MAC addresses'**, **MAC** stands for Media Access Control.
MAC is **6** bytes (48 bits), address of the **Physical device**, and is actually assigned to the device when it is made!!
This is separate from a logical address like an **IP** address.

Type or **Length**:
The last field of the Ethernet frame, It is **2** bytes (16 bits) length.
It can be used to represent either the type of the encapsulated packet, or the length of the encapsulated packet.
→ If the value in this field is **1500 or less**, this means it is indicating the **LENGTH** of the encapsulated packet (in bytes)!
For example, if the value in this field is **1400** it means that the encapsulated packet is 1400 bytes in length!
→ If the value was **1536 or greater**, this means it indicates the **TYPE** of the encapsulated packet (usually **IPv4** or **IPv6**).
And the length is determined via other methods.
For example, a value of **0x0800** which is written in hexadecimal and is equal to 2048 in decimal, indicates the version of the **IP**.
And the 2048 is greater than 1536, and identify that the version of the **IP** is version 4 (**IPv4**).
For example, a value of **0x86DD** which is written in hexadecimal and is equal to 34525 in decimal, indicates the version of the **IP**.
And the 34525 is greater than 1536, and identify that the version of the **IP** is version 6 (**IPv6**).

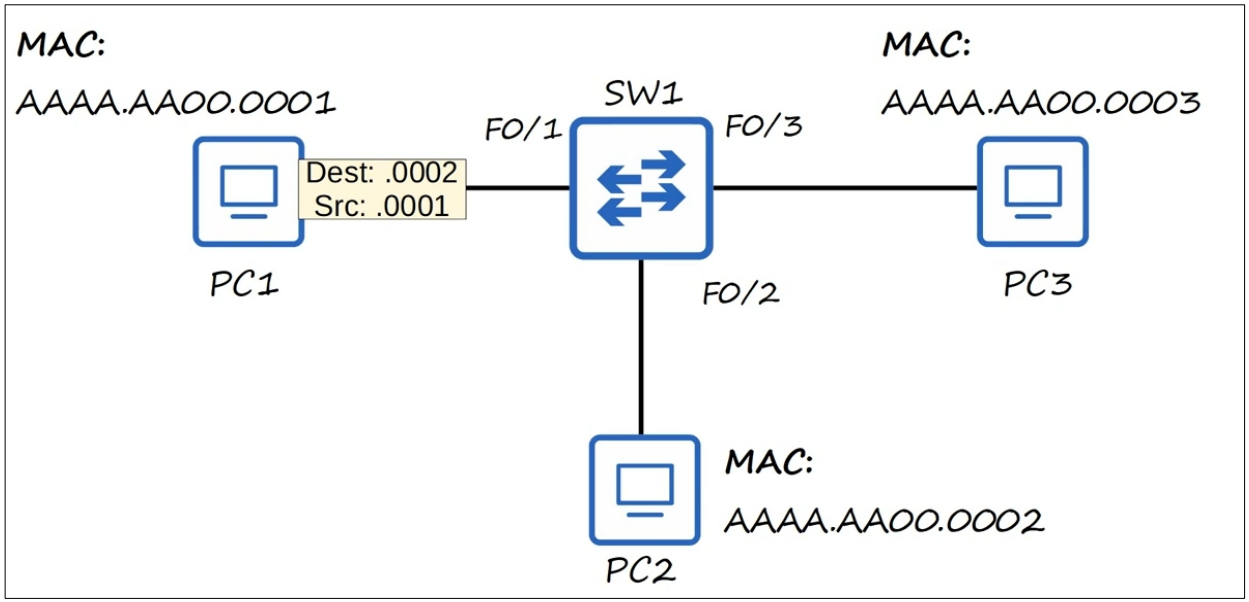
Try to remember the length of each field in the Ethernet frame!

The only field of the Ethernet trailer is the **FCS**, which stands for **Frame Check Sequence**.
It is **4** bytes (32 bits) in length. Its purpose is to detect corrupted data by running a '**CRC**' algorithm over the received data.
CRC means **Cyclic Redundancy Check**.
Cyclic refers to something called 'Cyclic Codes', **Redundancy** refers to the fact that these **4** bytes at the end of the message, enlarge the message without adding any new information, so they are **redundant**. **Check** refers to the fact that it checks, or verifies, the data for errors.

→ **Now**, the total size including ethernet header and trailer, to **26** bytes (208 bits).

→ MAC Address:-

MAC stands for Media Access Control.
MAC is **6** bytes (48 bits), address of the **Physical device**, and is actually assigned to the device when it is made!!
This is separate from a logical address like an **IP** address
A.K.A '**Burned-In Address**' (**BIA**), This is because the address is burned-in to the device as it is made!
The **MAC** address is *globally* **unique**! You'll never see two devices in the world have the same MAC address.
Although, there are MAC addresses known as '*locally-unique*' addresses, which don't have to be globally unique through out the world, However, In almost all cases MAC addresses are *globally unique*.
The first **3** bytes of the MAC address are the **OUI** '***Organizationally Unique Identifier***', which is assigned to the company making the device. So **Cisco** will have various **OUI**s which only Cisco can use!
The last **3** bytes are unique to the device itself.
MAC addresses are written as a series of **12 hexadecimal** characters.



This kind of frame is called **Unicast frame**: a frame destined for a single target.

So, **PC1** sends the frame through it's network interface card, which is connected to **SW1**, after the **SW1** receives the frame, it looks at the source MAC address field of the frame and then uses that information to ***learn*** where **PC1** is!

MAC Address Table

MAC	Interface
.0001	F0/1

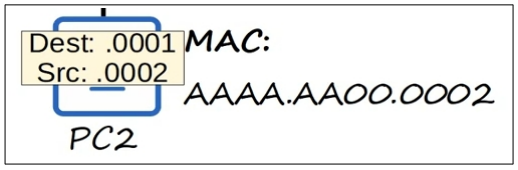
As you can see here, it adds the MAC address to it's MAC address table, and it associates that MAC address with its **F0/1** interface!
This is known as a ***Dynamically Learned MAC address***.
Or Just ***Dynamic MAC address***.
Because it wasn't manually configured on the Switch. The Switch learned it itself.
Every Switch will keep a MAC address table like this and fills it dynamically by looking at the source MAC address of frames it receives!

Since **SW1** received a frame from source MAC address AAAA.AA00.0001 on it's **F0/1** interface, It knows that it can ***reach*** that MAC address on **that** (**F0/1**) interface! And adds it to the MAC address table! *→ An Important Concept! ←*
→ This is how Switches dynamically learn where each device on the network is!, by looking at the source MAC address of the frame.

NOW, there is one problem:
The destination of the frame is AAAA.AA00.0002, but **SW1** doesn't know where that is!
This by the way, called an ***Unknown Unicast Frame***, a frame for which the Switch doesn't have an entry in its MAC address table.

This has only one option, that is to **flood** the frame!
Flood means to forward the frame out of ALL of its interfaces, except the one it received the packet on (**PC1**).
It will happen like this, **SW1** copies the frame and sends it out its **F0/2** and **F0/3** interfaces. (but not the **F0/1**)
After that, **PC3** ignores the packet because the destination MAC address doesn't match its own MAC address, and simply drops out the packet!
PC2, however, receives the packet, and then processes it normally, up the **OSI** stack.
However, unless **PC2** sends a reply of some sort, it stops here!

SW1 never receives a packet from **PC2**, so it can't learn **PC2**'s MAC address and use it to populate the MAC address table!
So, let's say **PC1** sends another frame to **PC2**, the same process happens again, **SW1** will flood the packet, PC3 drops the frame and **PC2** receives it and process it normally.



NOW, Let's say **PC2** wants to send some data to **PC1**, may be a reply to what it sent.
PC2 sends he frame out of its network interface, and **SW1** receives it.
Now **SW1** looks at the source MAC address of the frame, and then adds **PC2** MAC address to its MAC address table, associating it with the **F0/2** interface.
This time, however, it doesn't **flood** the frame. This is called a ***Known Unicast Frame***, because the destination is already in its MAC address table. And then **PC1** process the frame up the **OSI** stack, through the de-encapsulation process.
Whereas UNKNOWN Unicast Frames are *flooded*, KNOWN Unicat Frames are simply *forwarded*.

NOTE : Dynamic MAC addresses are removed from the MAC address table after **5 minutes** of inactivity!

→ Let's Look to an another example, with 2 Switches :

If **PC1** wants to send some data to **PC3**, the source MAC address is AAAA.AA00.0001 and the destination MAC address is AAAA.AA00.0003.
So, **PC1** sends the frame out of its network interface **F0/1** and it arrives at **SW1**, and then **SW1** learns **PC1**'s MAC address from the source address field of the frame, and associates it with the interface on which it was received **F0/1**. Now **SW1** has learned that **PC1** can be reached via it's **F0/1** interface, but still doesn't know where **PC3** is! This is an UNKNOWN Unicast Frame!

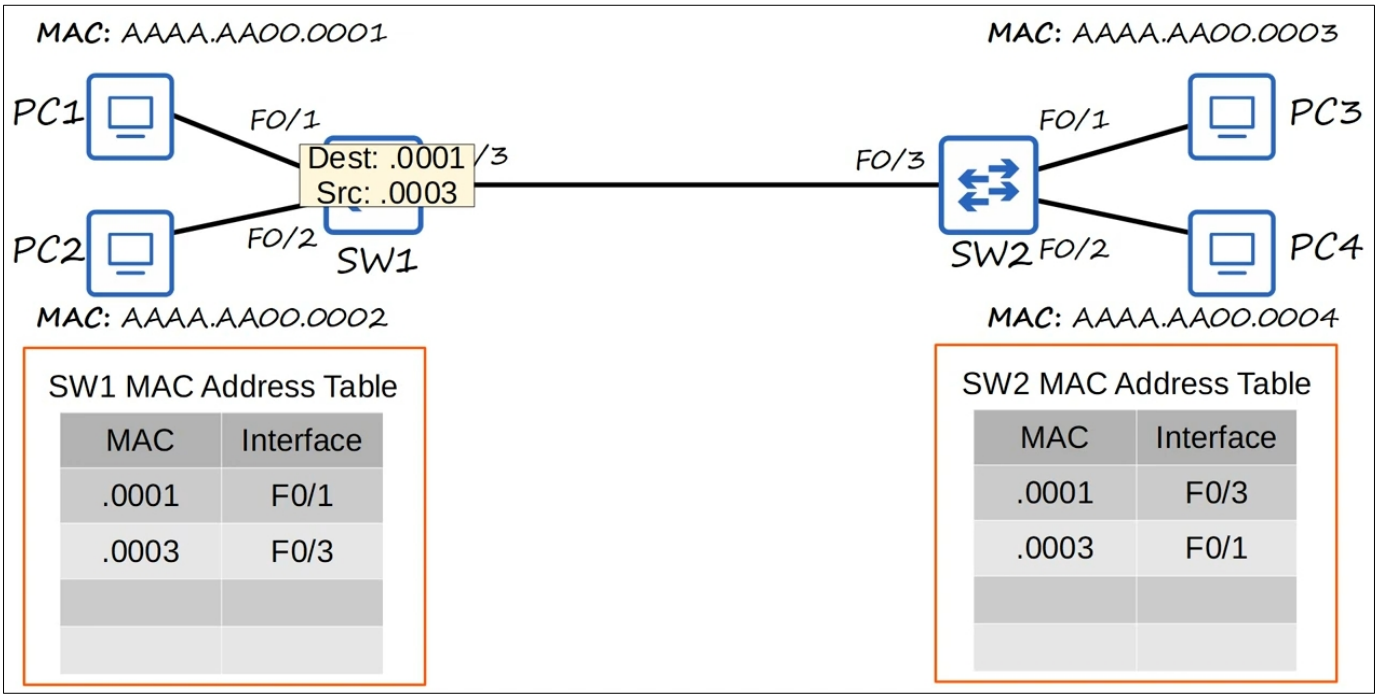
Now **SW1** *floods* out all of it's ports, except the one it was received on!
In this case it will *flood* the frame out of **F0/2** and **F0/3**, but not **F0/1**!
NOW **PC2** drops out the frame, and the exact same rules *apply*.
Just like **SW1** did, **SW2** uses the source MAC address field of the frame to dynamically learn **PC1**'s MAC address and the interface it can reach **PC1**. (associate it with **F0/3**)

NOTE that, unlike on **SW1, PC1** isn't actually directly connected to the interface **SW2** enters in its own MAC address table!
However, this is the interface which **SW2** will use to reach **PC1**.
And that is the meaning of the interface in the MAC address table, it doesn't mean the device is directly connected to this interface.

NOW **SW2** received a Unicast Frame, that is a frame destined for a single device, but still doesn't where the destination MAC address is!? Because it is not on its own MAC address table! So it will *flood*! **SW2** floods the frame out all interfaces, except the one it was received on (**F0/1** and **F0/2** but not **F0/3**)!
And Now **PC4** drops out the frame as the destination MAC address doesn't match its own!
PC3 receives the frame as matches its own MAC address! And let's say **PC3** wants to reply to **PC1**!
NOW the source and the destination MAC addresses will be reversed!
PC3 sends the frame out of its network interface and it is received by **SW2**.
SW2 learns **PC3**'s MAC address and associates its **F0/1** interface in its MAC address table.

The Switch uses the Source MAC address field to fill its MAC address table, because if it receives a frame from that source on the interface, it knows that it can reach that MAC address via that interface.

SW2 already has an entry for the destination MAC address in its table, so there is **no** need to *flood* the frame!
Now it is *forwarded* normally out of the corresponding interface in the MAC address table which is **F0/3**!
The frame is received by **SW1** which adds an entry for **PC3**'s MAC address in it own table with the interface **F0/3**.
Finally, since **SW1** has an entry for the destination MAC address in its own table, it will then *forward* the frame out of the corresponding interface **F0/1**, and it reaches its destination **PC1**.



And here are some information about the HEXADECIMAL

DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.
0	0	8	8	16	10	24	18
1	1	9	9	17	11	25	19
2	2	10	A	18	12	26	1A
3	3	11	B	19	13	27	1B
4	4	12	C	20	14	28	1C
5	5	13	D	21	15	29	1D
6	6	14	E	22	16	30	1E
7	7	15	F	23	17	31	1F

Each hexadecimal digit is **4** bits, so **2** digits equals **8** bits or **1** byte.

QUICK:

SFD signifies the end of the **Preamble**, and not used to provide receiver clock synchronization.
Preamble is an alternating series of **1**'s and **0**'s, allows the receiving device to synchronize its receive clock!
FCS is used to detect errors that occurred during transmission.
OUI is the first half (**24** bits) of a MAC address. It's a unique value assigned to the maker of the device.
Source is the field of the Ethernet frame which the Switch uses it to populate its MAC address table.
Destination field of the Ethernet frame also **specifies** a MAC address, it doesn't help the Switch to populate its MAC address table.
UNKNOWN Unicast frame is the kind of frame that Switch *floods* out of all interfaces except the one it was received on.
KNOWN Unicast frame is a frame for which the destination MAC address is already in the Switch's MAC address table.

Day 6: Ethernet LAN Switching Pt.2 :-

→ We will talk a little bit about the frames:
The **Preamble** and **SFD** are not considered part of the Ethernet header, although they're sent with every Ethernet frame!
So the Ethernet header consists of the three fields (**Destination**, **Source** and **Type**).
Therefore the size of the Ethernet header + trailer will be **18** bytes.
There is a minimum size for an Ethernet frame which is **64** bytes including (header + Payload [Packet] + trailer).
64 bytes – **18** bytes (header + trailer) = **46** bytes for the **Payload**.
The **64** bytes doesn't include the **Preamble** and **SFD** by the way.

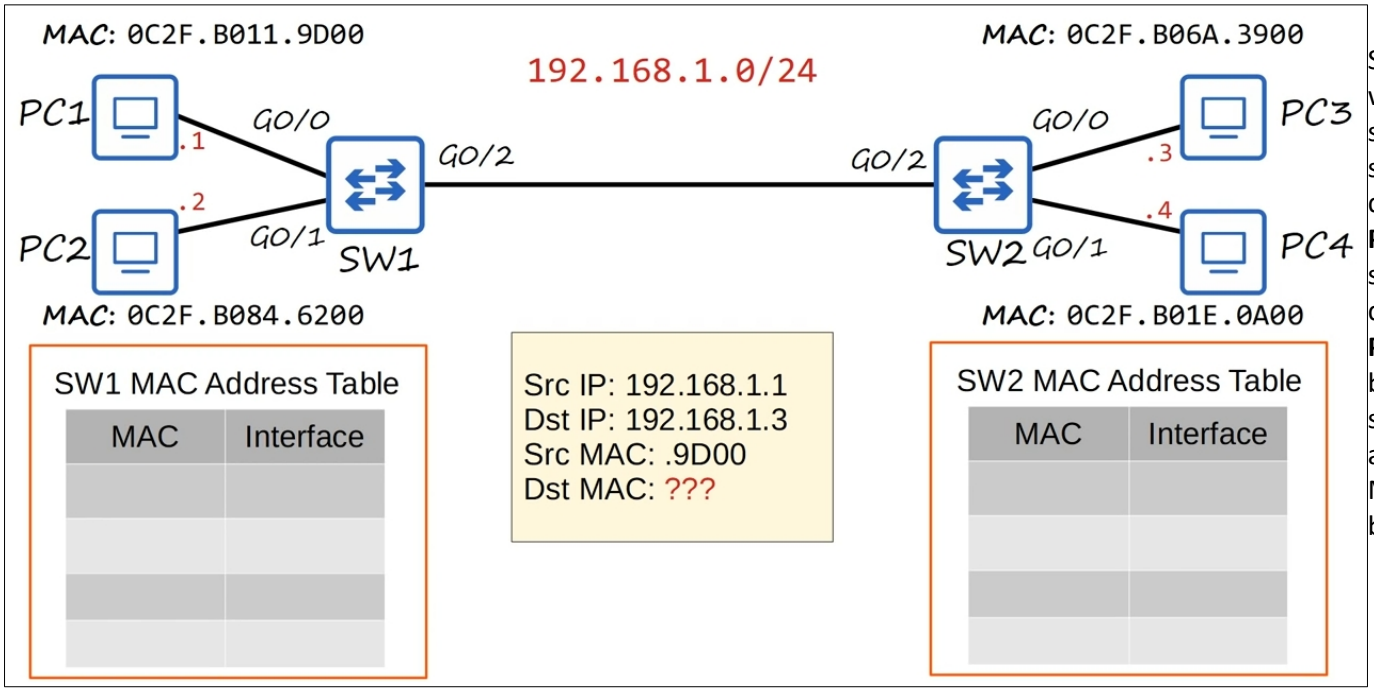
If the **Payload** is less than **46** bytes, *padding* bytes are added to make it equal to **46** bytes, and these bytes are **zeros** by the way!

Remember: that the **Preamble** and **SFD** might not be included as part of the Ethernet header, depending on how you define it.
But they are included in every Ethernet frame.

→ Let's get started:

We made a simple change to the previous network, we made the Interface as **Gigabit Ethernet** like (**G0/0**, **G0/1** and **G0/2**) and we also gave the PCs a more realistic MAC addresses.
And we'll add some IP addresses, **192.168.1.0/24**, this IP address represents the whole network here.
Then **192.168.1.1**, represents the **PC1** IP address and **192.168.1.2** represents the **PC2** IP address, etc.

When a device sends some data to another device, it doesn't just include a **Source** and **Destination** MAC address.
Encapsulated within that frame is an **Internet Protocol**, a.k.a, **IP** packet, and that IP Packet includes a **Source** and **Destination** IP address.



So, If **PC1** wants to send some data to **PC3**, the src and dest IP of **PC1** will be as shown and src MAC will be abbreviated as shown.
However, **PC1** doesn't know **PC3** MAC address.
When you send data to another computer, you enter the IP not the MAC address.
So the user enter the destination IP, but **PC1** has to discover **PC3**'s MAC by itself!

Remember: These 2 Switches operate at Layer2 not Layer3, so they need to use MAC addresses not IP addresses.

So, If **PC1** wants to send the frame to **PC3**, but first it has to learn **PC3**'s MAC address.
To do so, it uses something called **ARP**, **Address Resolution Protocol.**

Let's take a look at ARP:

→ It is used to discover Layer2 address (MAC) of a known Layer3 address (IP).

→ **ARP** process consists of 2 messages:

- **ARP Request:** sends by the device wants to know the MAC address of the other device.
- **ARP Reply:** which is sent to inform the requesting device of the MAC address.

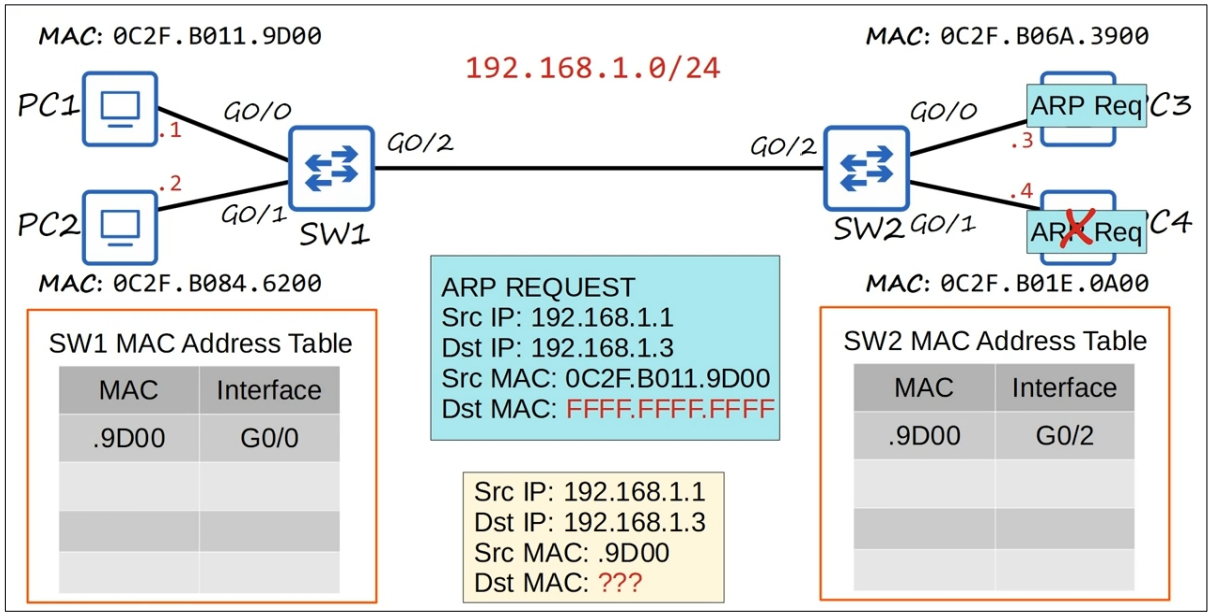
In our example, **PC1** would sent the **ARP Request** and **PC3** would sent the **ARP Reply**.

→ **ARP Request** is **broadcast**, which means it is sent to all hosts on the network.

Because the Layer2 address of the destination is unknown, it **broadcasts** the request and waits for a reply from the correct device.

→ **ARP Reply** is **unicast**, which means it is sent only to one host (the host that sent the ARP request).

→ **FFFF.FFFF.FFFF** is the broadcast MAC address that is sent as the destination MAC address in the **ARP Request**.



Now, **PC1** sends the frame out of its network interface and it is received by **SW1**, which is adding **PC1**'s MAC address to its table and associates it with the **G0/0** interface.

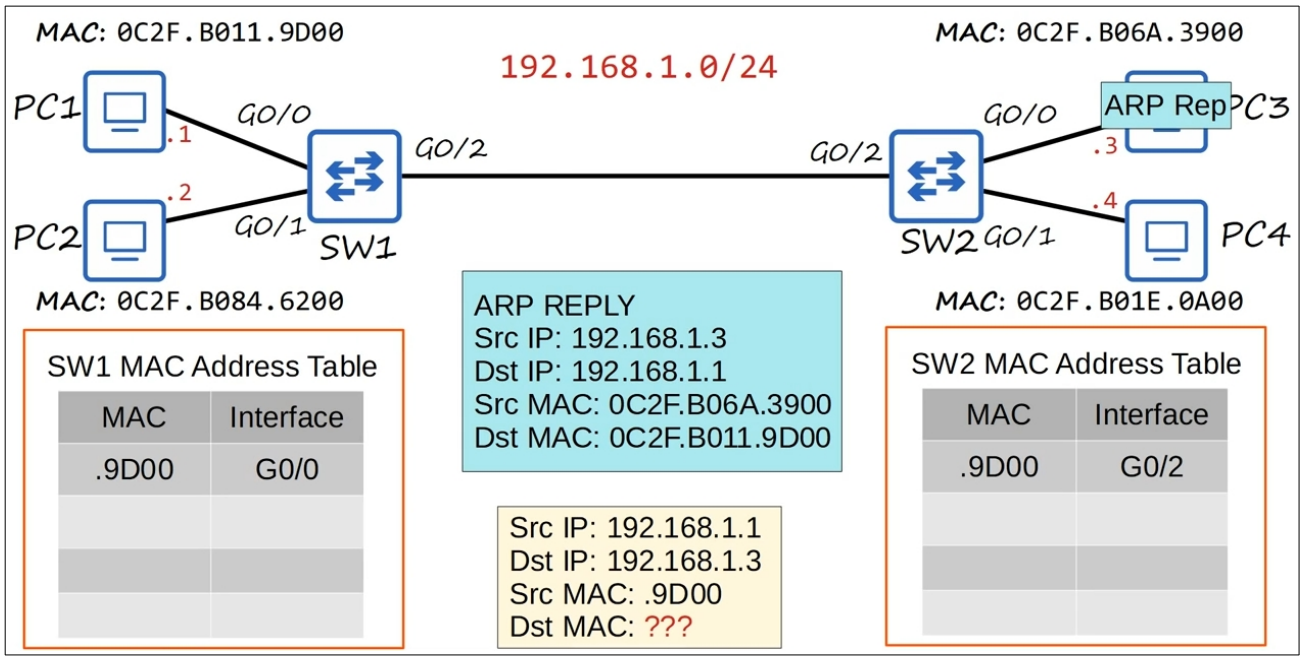
Since the destination MAC address is all **FFFF**'s, **SW1** broadcasts the frame out of all of its interfaces except the one it was received on. This is very like what a Switch does with an UNKNOWN Unicast frame. **PC2** receives it, but ignores the frame.

Then **SW2** learns **PC1**'s MAC address and adds it to its table associating it with **G0/2** interface.

Since the destination MAC address is all **FFFF**'s, SW2 also broadcasts the frame out of all of its interfaces except the one it was received on. Now **PC4** ignores the frame as the destination IP address doesn't match its own IP address.

However, **PC3** recognizes that he dest IP address does match its own, so it doesn't ignore the **ARP request**.

What **PC3** really does is send the other message, the **ARP reply**. Now we can see the **ARP reply** packet reversed to the previous **ARP request**.



Now the destination MAC address matches the own MAC of **PC1**.
Now **PC3** sends the frame out of its interface to **SW2** directly, as it has an entry for the destination of the MAC address of **PC1**.
This kind of frame is a KNOWN Unicast frame and **SW2** *floods* it directly.
SW2 learns the **PC3**'s MAC address and adds it to its own table associating it with its **G0/0** interface.
SW2 sends the frame towards **SW1** as it learns the interface of **PC1**, **SW1** receives the frame and since it is already learned **PC1**'s MAC address on the **G0/0** interface, it simply forwards the frame out of the interface and **PC1** finally receives the frame!
PC1 will then use that information to add an entry for **PC3** to its **ARP Table**. Which is used to store these IP address to MAC address associations.

→ Let's take a look at the ARP Table:
we use the command (`arp -a`) to view the ARP Table, On Windows, Linux or MacOS.
This screenshot is from my pc:

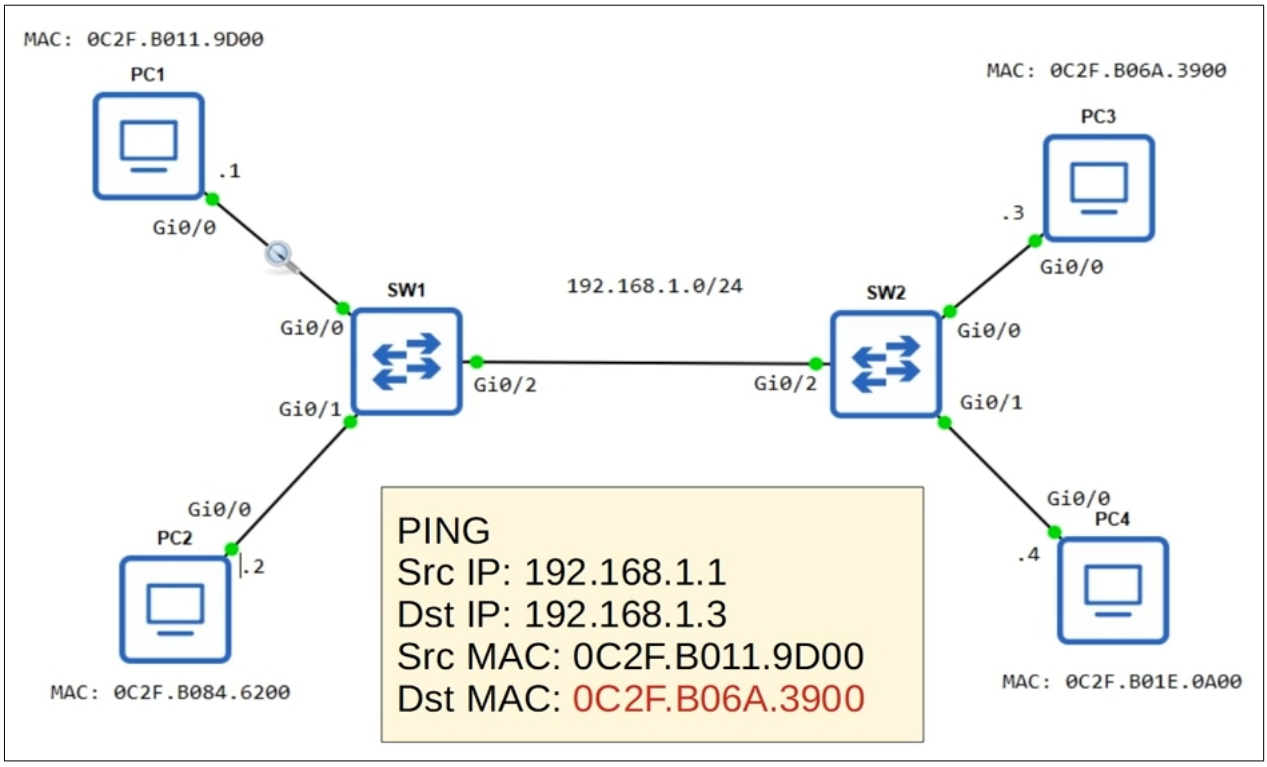
```
C:\Users\hp>arp -a

Interface: 192.168.242.1 --- 0x3
  Internet Address      Physical Address      Type
  192.168.242.255       ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 192.168.56.1 --- 0x8
  Internet Address      Physical Address      Type
  192.168.56.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static

Interface: 192.168.221.1 --- 0xf
  Internet Address      Physical Address      Type
  192.168.221.255       ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

The **Internet Address** column displays IP addresses (Layer3 addresses).
The **Physical Address** column displays MAC addresses that correspond to the IP addresses (Layer2 addresses).
If the **Type** column displays ***static***, it means that it is a default entry, it wasn't actually learned by sending an ARP request.
However, If the **Type** column displays ***dynamic***, it means that the entry was learned by sending an ARP request and receiving an ARP reply.
NOTE: Switch only floods the UNKNOWN Unicast frame and the Broadcast frame.
→ Packet tracer & GNS3 :
Packet tracer is a network simulator, it's a piece of software designed to simulate the operation of the real network.
GNS3, however, runs actual Cisco IOS software, so these are real Cisco switches running virtually.
GNS3 requires you to purchase your own copies of Cisco IOS, so although GNS3 itself is free, using Cisco IOS with GNS3 is not!



If **PC1** wants to *ping* **PC3**, the same previous operation will occur.

→ Ping:

- A network utility that is used to test reachability.
  For example, to test if two computers can reach each other
- Measures round-trip time.
  For example, time from **PC1** to **PC3**, then back to **PC1**.
- Uses two messages: somehow similar to **ARP**
  1. **ICMP Echo Request**, but the PC won't broadcast the request, it is sent to a *specific* host. It has to know the MAC address of the destination host, which is why **ARP** must be used first.
  2. **ICMP Echo Reply**

To use *ping*, we use the command (**`ping + ip`**)

```
PC1#
PC1#ping 192.168.1.3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.3, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 20/20/22 ms
PC1#
```

By default, a *ping* in Cisco IOS sends **5 ICMP echo requests**, and then you should get **5 ICMP echo replies** back. And the default size of each ping is **100**-bytes.

The period or the (**.**) indicates a failed ping, and the exclamation points (**!**) indicate a successful ping.

The first ping fails because of the **ARP**.

PC1 didn't know the destination MAC address, so it had to use **ARP**, and in that time the first ping failed.

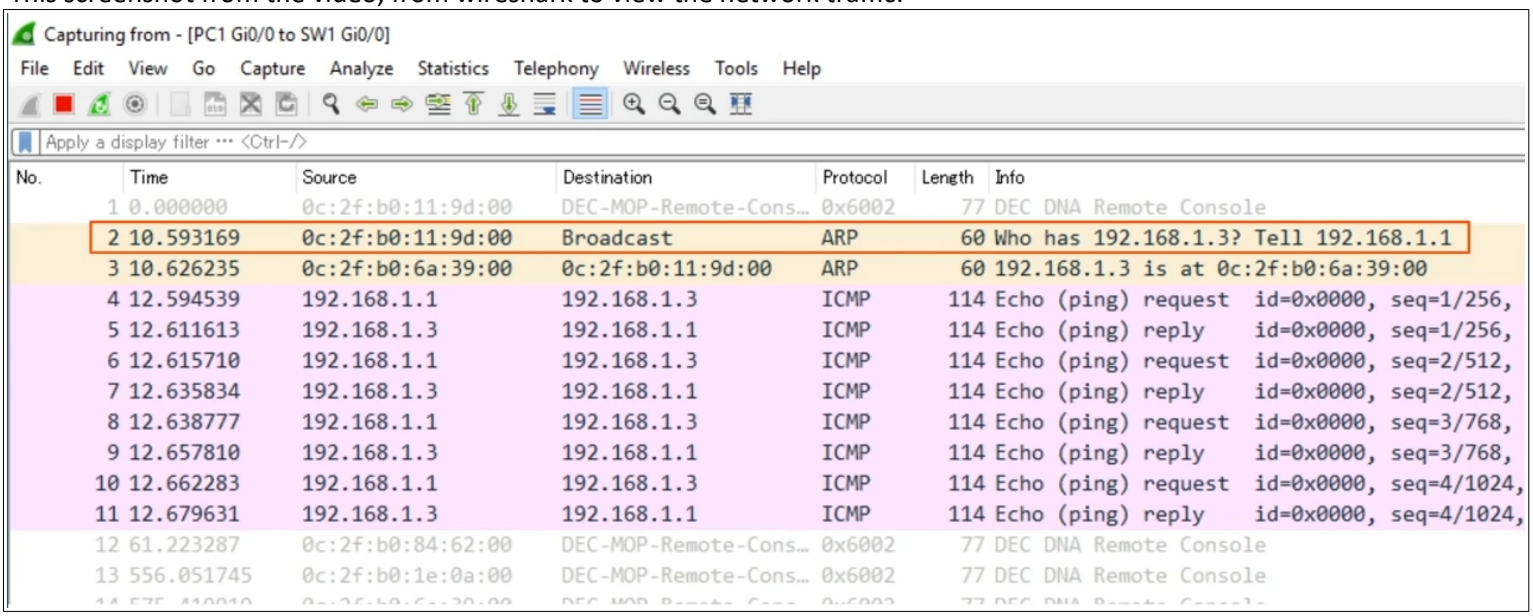
After **PC1** learned **PC3**'s MAC address, however, the ping succeeded !

Let's take a look at the **ARP** table, in Cisco IOS, like the previous screenshot:

The command only in Cisco IOS, is (**`show arp`**)

```
PC1#show arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1            -   0c2f.b011.9d00  ARPA   GigabitEthernet0/0
Internet  192.168.1.3           34   0c2f.b06a.3900  ARPA   GigabitEthernet0/0
PC1#
```

Wireshark allows you to perform '**packet captures**', to analyze the contents of network traffic.
This screenshot from the video, from wireshark to view the network traffic:



The **ARP request** is asking, which MAC address has an IP address of **192.168.1.3**?, and to send the reply to itself **192.168.1.1**.
It is a **broadcast**, all are **FFFF**'s.
Next is the **ARP reply**, from **PC3** with the destination to **PC1**'s MAC address, in the Info section, telling **PC1** its MAC address!
After that there are **4 ICMP echo requests** and **4 ICMP echo replies**!
Note that: the **ICMP echo requests** have a source IP of **PC1** and destination of **PC3**, and the **ICMP echo replies** have a source of **PC3** and destination of **PC1**.

Basically If device A wants to send some traffic to device B, which is on the same network, device A first has to use ARP to learn device B's MAC address, and then it can send traffic to device B.

→ Let's take a look at the MAC address table on a Cisco Switch:
to view it we use the command (**`show mac address-table`**)

```
SW1#show mac address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    0c2f.b011.9d00    DYNAMIC     Gi0/0
   1    0c2f.b06a.3900    DYNAMIC     Gi0/2
Total Mac Addresses for this criterion: 2
SW1#
```

Vlan means Virtual Local Area Network.
The default value of **VLAN** is **1**.
Dynamic MAC addresses display in the MAC address table as well. As the Switch learned them dynamically.
And we have **Ports**, which is another word for **Interfaces**.
Remember that these dynamic MAC address are removed from the MAC address table after **5** minutes of inactivity.
This is known as **Aging**.
However, you can also *manually* remove MAC addresses from the table by using the command;
(**`clear mac address-table`**).
If you don't want to clear all the MAC addresses from the table you can add some additional options to the command, like:

- **`clear mac address-table dynamic address 0c2f.b011.9d00`**
- **`clear mac address-table dynamic interface Gi0/0`**
  - This clears all MAC address table entries for a specific interface.

Some **Wireshark**:
Minute **24:10** to Minute **26:00**

NOTE: The **ARP** Ethernet type is (**0x0806**).
This indicates that an **ARP** packet is inside of this Ethernet frame.

QUIZ:

Both the **ICMP request and reply** are unicast messages.
The **ARP request** message is used to learn the Layer2 address of a host.

Day 7: IPv4 addressing Pt.1 :-

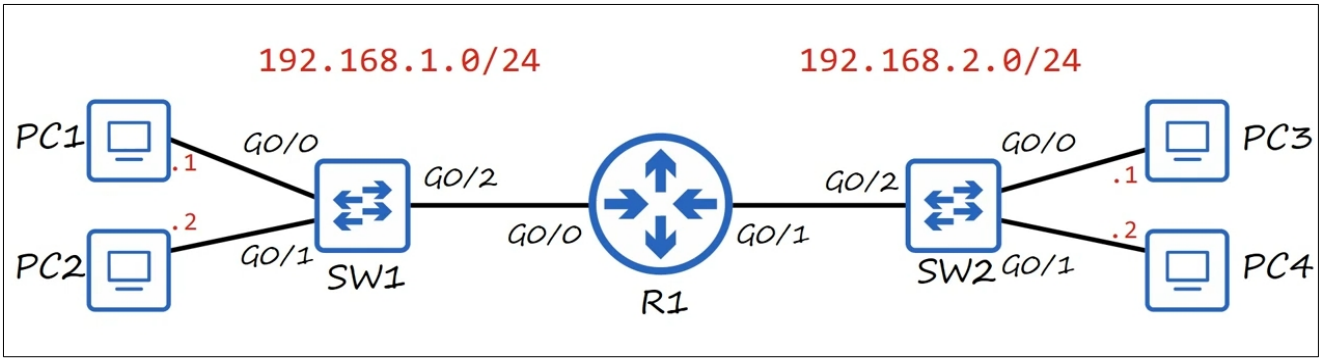
→ <u>How the traffic forwarded between different LANs</u>
→ Let's get started:

We are going up the **OSI** model, from Layer2, Data Link layer, to Layer 3, the Network Layer.

Some review on Layer3:
→ The network Layer provide connectivity between end hosts on *different* networks (outside of the LAN).
→ Provides logical addressing (**IP** addresses).
→ Provides path selection between source and destination.
→ Routers operate at Layer3.

IP addresses are logical addresses you assign when you configure the device.
Over larger or complex networks, like the Internet, there can be many different possible paths to a destination. Selecting the best path is part of Layer3's functionality.

→ Our focus will be specifically on the Logical Layer3 addresses (*IP Addresses*):



Now If we put a Router between the previous network with 2 Switches, The **R1** will split them into two separate networks!
<u>*NOTE*</u>: the first 3 groups of numbers of theses network addresses represent the network itself (**192.168.1** or **192.168.2**).
and only (**.0**) changes to represent the end hosts on the network.
(**/24**) at the end means, they are used to tell what part of the address represents the network and which part represents the end hosts, the PCs. (**/24**) means that the first **3** groups of numbers represent the network.

There is one thing is missing in this network diagram, the Router needs an IP address!
And not only one IP address, it needs an IP address for each network it is connected to.
So, let's give the **R1**'s **G0/0** interface an IP address of **192.168.1.254** and it's **G0/1** interface an IP address of **192.168.2.254**.

<u>This time:</u> If **PC1** sends a frame to the broadcast MAC address of all **FFFF**'s, **SW1** will receive the frame and it floods it out of all its interfaces except the one it was received on. So, it sends the frame out of **G0/1** and **G0/2**, and **PC2** and **R1** receive the frame. However, that's where it ends! The broadcast is *limited* to the Local network, it doesn't cross the Router and go to **SW2**, **PC3** and **PC4**.

IPv4 Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

This is a chart from Wikipedia showing the **IPv4** header. **IP**, or, **Internet Protocol**, is the primary Layer3 protocol in use today and version 4 is the version in use in most networks.

→ In this Day, we'll focus on 2 fields, the Source and Destination IP address:

→ These fields are both **4 bytes** (**32** bits) in length.

Let's take a look at this IP address (**192.168.1.254**):
An IP address is 32 bits long, so each of these four groups of numbers represents **8** bits.
192 represents **8** bits, **168** represents **8** bits, **1** represents **8** bits, **254** represents **8** bits.
If we write these **8** bits out as **1**'s and **0**'s, **192** → **11000000**, **168** → **10101000**, **1** → **00000001**, **254** → **11111110**.
This way of writing with just **0** and **1** is called ***binary***, and it is difficult to read and understand!
So, IP address are written with what's called *dotted decimal*, because there are for decimal number, **192**, **168**, **1** and **254** separated by dots or *periods.*

→ Let's learn about *binary:*
A quick review about *Decimal* and *Hexadecimal* from Minute **8:26** to Minute **10:20**.

Binary System: Binary is (***base 2***), meaning each digit increases by a factor of **2**, it doubles.
So, that means that **192** which is equal in binary **11000000** is really:

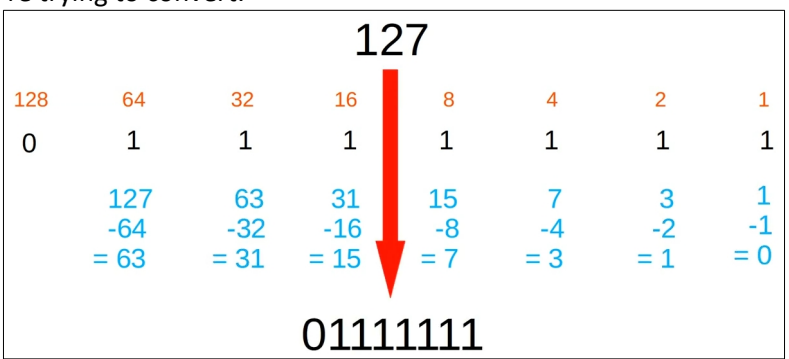
1	1	0	0	0	0	0	0
1 * 128	**1 * 64**	0 * 32	0 * 16	0 * 8	0 * 4	0 * 2	0 * 1

128 + 64 = 192

And so on for the rest of the 3 group numbers, **168**, **1** and **254.**
This is the preferred way to convert the binary ***octets*** to *decimal*.

Note: You'll often hear each of these **8** bits groups referred to as '***octets***'.

→ Let's try converting in the opposite, from *decimal* to *binary*:
To do so, it is recommended, to write out the values of each bit in a binary octet, like the picture.
Then try to subtract each number from the decimal number you're trying to convert.
And if it subtracts well, then we add a **1** under that unit,
f we can't subtract we write a **0** under that unit.

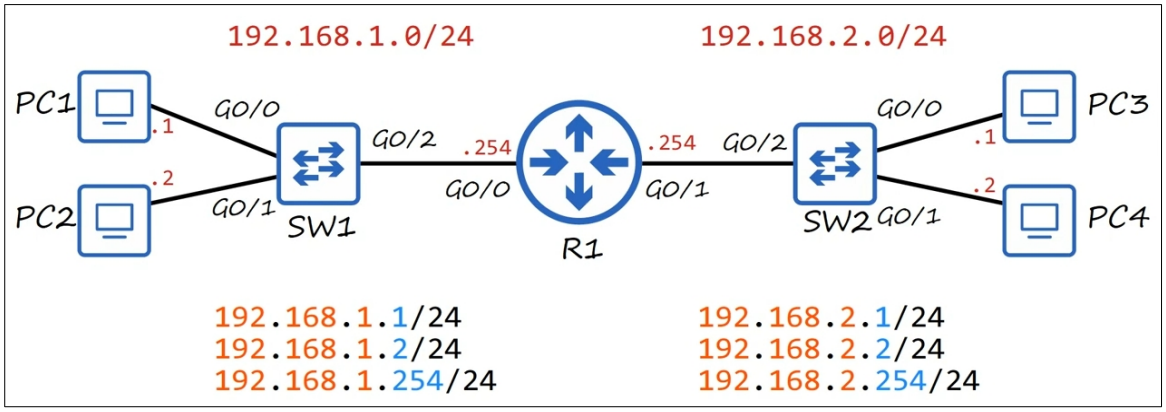


NOTE: The example of converting **221** from decimal to binary in the video is ***incorrect***!! The right value is: **11011101**.

The range of possible numbers that can be represented with **8** binary bits, ranges from **0**, if all bits are zero, to **255**, if all bits are ones. Because **128** + **64** + **32** + **16** + **8** + **4** + **2** + **1** = **255**.

So, and **IPv4** address is really a series of **32** bits. It is split up into ***4 octets***, and then written in ***dotted decimal*** format to make it simpler for us to read and understand.
You also remember there was **/24**, that is used to *identify* which part of the IP address represents the network and which represents the end host.
Since an IP address is **32** bits, you can guess what this **/24** means?
It means that the first **24** bits of this IP address represents the network portion of the address, and the remaining **8** bits represent the end host.
So, the first **24** bits is equal to = the first **3** octets, because 8 + 8 + 8 = 24.
So, **192.168.1** is the network portion of the address, and **254** is the host portion.

If we look at the network from the previous slide, we will see that **192.168.1** and **192.168.2** are the network portion of both the two separate LANs, and the host portion is what changed. You will notice that the host portion changed to the hosts connected together in the LAN.



Another example: **154.78.111.32/16**, now which 16 bits is the network portion and which is the host portion?
Well, what comes after the Slash **/**, we start it from the beginning, which means here the first half of the IP address.
So, **154.78** is the network potion and **111.32** is the host portion.

Another example: **12.128.251.23/8**, now the first **8** bits (**12**) is the network portion, and the last **24** is the host portion (**128.251.23**).

→ IPv4 Address Classes:

→ **IPv4** addresses are split up into **5** different '***classes***'.
→ The class of an **IPv4** address is determined by the ***first octet*** of the address.
→ However the slide, the classes of address we will be focusing on are **A**, **B** and **C**.
→ Addresses in class **D** are reserved for '***multicast***' addresses.
→ Multicast is another type of address, separate from unicast and broadcast
→ Class **E** addresses are reserved for experimental uses.
→ The end of the class **A** range is usually considered to be **126**, not **127**.

Class	First octet	First octet numeric range
A	0xxxxxxx	0-127
B	10xxxxxx	128-191
C	110xxxxx	192-223
D	1110xxxx	224-239
E	1111xxxx	240-255

→ Loopback Addresses:
→ The **127** range is reserved for '***loopback addresses***'.
→ The range is anything with a first octet of **127**, so **127.0.0.0** to **127.255.255.255**.
→ These addresses are used to test the '***network stack***' (think **OSI**, **TCP/IP model**) on the local device.
→ If a device sends any network traffic to an address in this range, it is simply processed back up the **TCP/IP stack** as if there were traffic received from another device.

In this image, I pinged **127.0.0.1** on my Windows PC, and you can see that your own PC is responding to its own pings:

```
C:\Users\hp>ping 127.0.0.1

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

In this image, I sent a ping to a random IP address in the **127** range, **127.214.15.10**, and again My PC responds back to its pings:

```
C:\Users\hp>ping 127.214.15.10

Pinging 127.214.15.10 with 32 bytes of data:
Reply from 127.214.15.10: bytes=32 time<1ms TTL=128
Reply from 127.214.15.10: bytes=32 time<1ms TTL=128
Reply from 127.214.15.10: bytes=32 time<1ms TTL=128
Reply from 127.214.15.10: bytes=32 time<1ms TTL=128

Ping statistics for 127.214.15.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Notice the round trip times, all **0ms** milliseconds.
That's because the traffic isn't going anywhere, my PC just sending and receiving these pings to and from itself.

Now, here's the chart again, with a column added, **Prefix Length**.
Prefix Length is used to identify the length of the network portion of the address.

Class	First octet	First octet numeric range	Prefix Length
A	0xxxxxxx	0-127	/8
B	10xxxxxx	128-191	/16
C	110xxxxx	192-223	/24

Actually, if you look back at these three addresses we used in our examples, you can see that there Prefix Length matches the table:

Class A: 12.128.251.23/8

Class B: 154.78.111.32/16

Class C: 192.168.1.254/24

→ In class **A** there fewer possible network addresses, however, because the host portion is very long, there can be many hosts on each network.
→ Class **C** is the opposite, there are many more possible networks, but because the host portion is smaller, there are fewer hosts on each network.

Class	Leading bits	Size of *network number* bit field	Size of *rest* bit field	Number of networks	Addresses per network
Class A	0	8	24	128 (2^7)	16,777,216 (2^{24})
Class B	10	16	16	16,384 (2^{14})	65,536 (2^{16})
Class C	110	24	8	2,097,152 (2^{21})	256 (2^8)

The 'Leading bits' column refers to the first bits of the first octet.
The 'Size of network number bit' field displays the number of bits in the network portion of the IP address.
'Number of networks' displays the number of possible networks in each class.

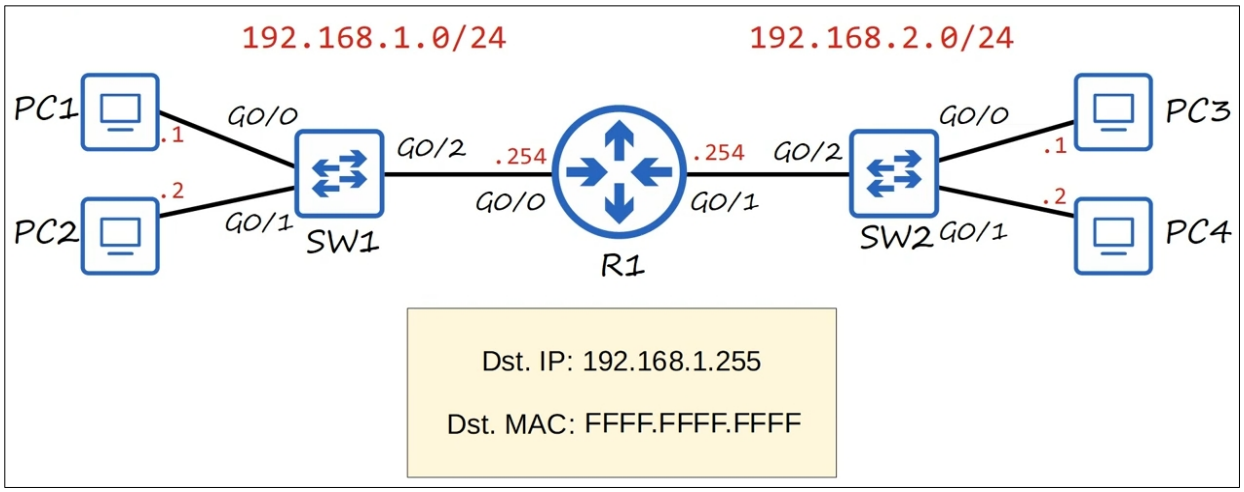
However, because the first address in each network is the network address, it can't be assigned to hosts!
Also, the last address of the network is the ***broadcast*** address, the Layer3 used when you want to send traffic to all hosts, and it also can't be assigned to the hosts.
So, really the host count is two less!

→ Netmask:

There is another way of writing these prefix length. Using a slash, followed by the length of the prefix, is a newer and easier way of writing the prefix length.
On *Juniper* network devices, you write prefix length using this slash notation.
However, *Cisco* devices still use an older, slightly more complicated way of writing the prefix length.
That is using a *dotted decimal netmask*.
A ***netmask*** is written in dotted decimal like an IP address, where the network portion is all **1**s and the host portion is all **0**s
For example, the ***netmask*** of class **A** address is **255.0.0.0**. That is the dotted decimal version of **11111111** followed by **24** zeros.
So, the ***netmask*** of class **B** address is **255.255.0.0**, it's the dotted decimal version of **11111111 11111111** followed by **16** zeros.
Netmask of class **C** address is **255.255.255.0**, it's the dotted decimal version of **11111111 11111111 11111111** followed by **8** zeros.
So these ***prefix length*** and the ***netmasks*** are the same thing, just written in different ways.

→ Two more types of IP addresses: The *network* and *broadcast* addresses:

→ If the host portion of an IP address is all **0**s, it means it is the ***Network address***.
→ The ***Network address*** is the *identifier* of the network itself.
→ In the last network diagram, you can see **192.168.1.0/24**. We know that **/24** means the first three octets are the network portion, and the last octet is the host portion.
→ Since the host portion is ***zero***, it means the last octet, the host portion, is all ***zero***s. Therefore, this address **192.168.1.0/24**, is the network address.
→ *Keep in mind*, The ***Network address*** can not be assigned to a host!
→ The ***Network address*** is the first address of the network, but the first ***usable*** address is *one* above the ***Network address***.
In this case, it is **192.168.1.1**, which is assigned to **PC1**.
→ However the last address in the network, with a host portion of all **1**s, is the ***broadcast address*** for the network.
→ Like the *network address*, the ***broadcast address*** can not be assigned to a host.
→ Although it is the last address in the network, the last ***usable*** address is actually *one* under the broadcast address.
In this case, that's **192.168.1.254**, the address assigned to **R1**'s **G0/0** interface.
→ So, the ***broadcast address*** is the Layer3 address used to send a packet to all hosts on the local network.
→ If a packet is sent with this destination IP address, **192.168.1.255**, what do you think will be the destination MAC address of the frame it is encapsulated in? → My answer is, the destination MAC address will be **FFFF.FFFF.FFFF**.
→ I am right! It will be all **FFFF**'s, the ***broadcast MAC address.***
→ If **PC1**,for example, sent a ping to **192.168.1.255**, It would be received by **PC2** and **R1**'s **G0/0** interface!


192.168.1.0/24
192.168.2.0/24
PC1
.1
Gi0/0
.2
PC2
Gi0/1
SW1
Gi0/2
.254
Gi0/0
R1
.254
Gi0/1
Gi0/2
SW2
Gi0/0
.1
Gi0/1
.2
PC3
PC4
Dst. IP: 192.168.1.255
Dst. MAC: FFFF.FFFF.FFFF

Day 8: IPv4 addressing Pt.2 :-

→ Some Review and Clarification:

→ We know that **127** range is reserved for the loopback, so it not considered a part of Class **A** range.
→ However, the **0** range is also reserved in Class **A**, so some might say class **A** really begins at **1**, making the range **1 – 126**. Different sources say different things, so it is recommended to remember the Class **A** as **0 – 127**.

→ **How to calculate the maximum number of *usable* addresses:-** that can be assigned to hosts:
Let's take this Class **C** network **192.168.1.0/24**, it uses a **/24** prefix length as it's Class **C**. and therefore the last octet, last **8** bits, are the host portion.
→ That means that the host portion can be **0** to **255**. Which gives **256** addresses as a total.

Host portion = **8** bits = $\mathbf{2^8}$ = **256**.
There are 2 types of addresses, the first is when the host portion is all *zero*s, it will be the *network* address (*Network ID*).
The second is when all are *one*s, it will be the *broadcast* address. Both of them can not be assigned to the host.

→ The maximum number of hosts per network is = $2^8 - 2 = 254$ hosts.

Let's take a look at this Class **B** network, **172.16.0.0/16**
→ The *broadcast* address is **172.16.255.255**, and the *network* address is **172.16.0.0**
→ The host portion is = $\mathbf{2^{16}}$ = **65**,**536**
→ Maximum hosts per network = $\mathbf{2^{16} - 2}$ = **65**,**534** hosts.

Let's take one more example on this Class **A** network, **10.0.0.0/8**:
→ The *broadcast* address is **10.255.255.255**, and the *network* address is **10.0.0.0**.
→ The host portion is = $\mathbf{2^{24}}$ **= 16**,**777**,**216**
→ Maximum hosts per network = $\mathbf{2^{24} - 2}$ = **16**,**777**,**214**.

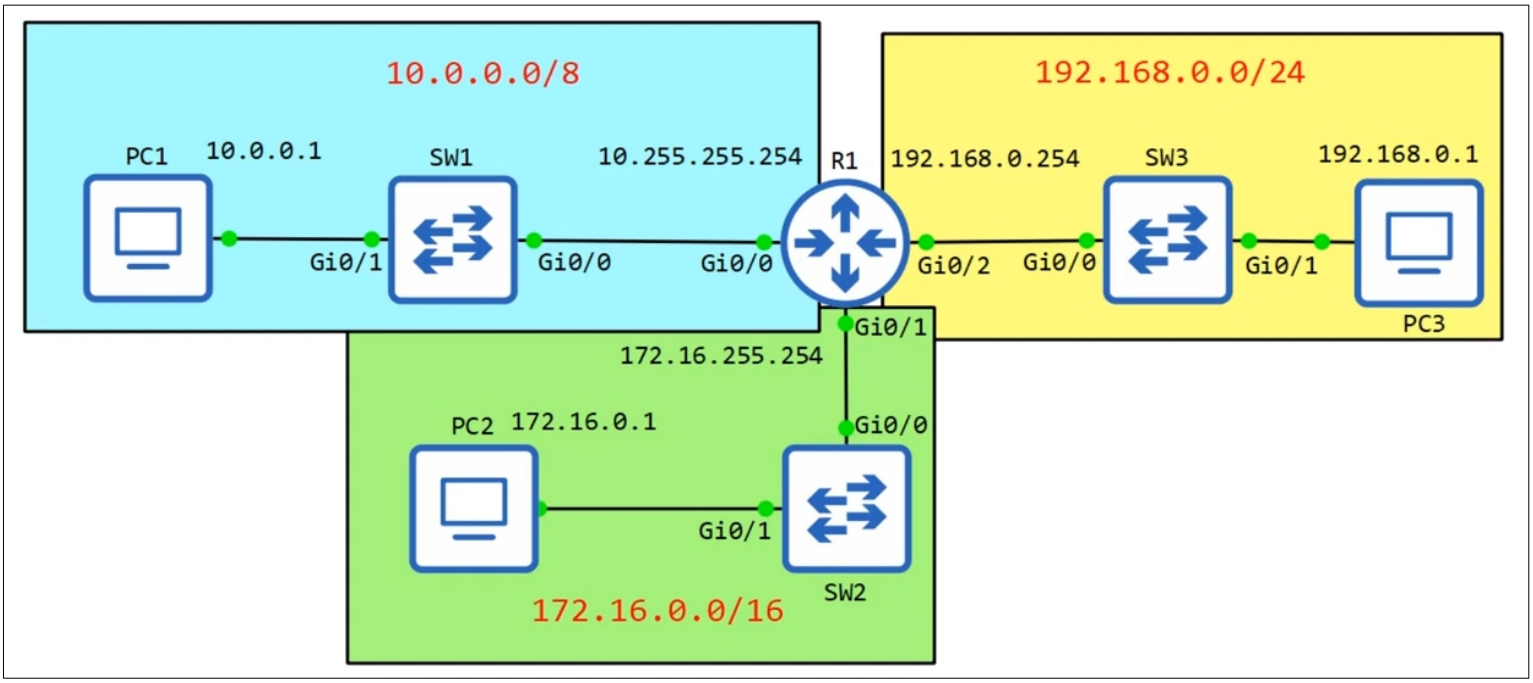
→ The formula for determining the number of hosts on a network is :
$\mathbf{2^n - 2}$, while the **n** = number of host bits.

→ **First/Last Usable Address:**
First Usable address, is the one after the *network* address!
Last Usable address, is the one before the *broadcast* address!
→ So, let's get them to the network, **192.168.1.0/24**:
The *network* address is the address with the host portion of all *zeros*, which is **192.168.1.0**, and the *broadcast* address is the address with the host portion of all *ones*, which is **192.168.1.255**,
As the previous, ***first usable*** address is **192.168.1.1**, and ***last usable*** address is **192.168.1.254**.

→ So, let's get them to the network, **172.16.0.0/16:**
The *network* address is the address with the host portion of all *zeros*, which is **172.16.0.0**, and the *broadcast* address is the address with the host portion of all *ones*, which is **172.16.255.255**,
As the previous, ***first usable*** address is **172.16.0.1**, and ***last usable*** address is **172.16.255.254**.

→ So, let's get them to the network, **10.0.0.0/8:**
The *network* address is the address with the host portion of all *zeros*, which is **10.0.0.0**, and the *broadcast* address is the address with the host portion of all *ones*, which is **10.255.255.255**,
As the previous, ***first usable*** address is **10.0.0.1**, and ***last usable*** address is **10.255.255.254**.

→ IPv4 Addressing:



→ Let's go into the CLI of **R1** and make the configurations:

```
R1>en
R1#show ip interface brief
Interface                  IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0         unassigned      YES unset  administratively down down
GigabitEthernet0/1         unassigned      YES unset  administratively down down
GigabitEthernet0/2         unassigned      YES unset  administratively down down
GigabitEthernet0/3         unassigned      YES unset  administratively down down
R1#
```

→ We used **en**, the shortcut of the **`enable`** command, to enter privileged exec mode.
The second command (**`show ip interface brief`**), we use it to confirm the status of each interface on the devices, as well as their IP addresses.
→ The **Interface** column, lists the interfaces on the device.
→ The **IP-Address** column, lists the IP of each interface. They're all unassigned yet.
→ **OK?** column, a *legacy* feature of the command, It's not relevant any more. Basically, It says whether or not the IP address is valid or not. On modern devices, the device won't let you assign invalid IP addresses, So, You should never see **NO** in the column.
→ The **Method** column, indicates the method by which the interface was assigned an IP address. Currently it Is *unset*.
→ The **Status** column, You can consider this the Layer1 status of the interface. If the interface is enabled, there is a cable connected, and the other end of the cable is properly connected to another device, you should see **up** here, not **down**. However, here it displays **administratively down**. This means the interface has been disabled with the '**shutdown**' command. AS we haven't done any configurations yet, this is the *default status* of Cisco router interfaces!
NOTE1: Cisco Switch interfaces are *not* ***administratively down*** by default.
They will be **up** if connected to another device or **down** if not.
NOTE2: Even though **GigabitEthernet0/0**, **0/1** and **0/2** on this router are connected to Switches, the Interfaces remain ***administratively down***, because the '**shutdown**' command is applied to them by default.
→ The final field is '***Protocol***', while the **Status** column refers to the Layer1 status of the interface, this '***Protocol***', is the Layer2 status. Because the interfaces are down at Layer1, Layer2 can not operate, So all of these interfaces are down at Layer2.
You' ll never see an interface are *down* at the **Status** column and *up* in the **Protocol** column, although the reverse is *possible*.
-- Remember: the **Status** refers to Layer1 Status, for example, is the interface is shutdown, is there a cable attached, etc.
Protocol refers to Layer2 Status, for example, is Ethernet functioning properly between this device and the one connected to it.

→ Configurations:

→ Let's configure the **GigabitEthernet0/0** first:

```
R1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#interface gigabitethernet 0/0
R1(config-if)#
```

We use the command (**`conf t`**), the shortcut of the 'configure terminal' to enter *Global Config Mode*.
Next, to configure the interface itself, We have to enter *Interface Config Mode.* So we use the command (**interface**) followed by the name of interface, *gigabitethernet 0/0.*
And now, as you can see, it displays (**config-if**) beside the hostname of the device.
→
Some ways to enter the **Configuration Mode:**
Minute **13:55** to Minute **15:32**.
→
→ Let's set the IP Address:
That is done with the command (**`ip address`**) and followed by the IP address you wanna set.

```
R1(config-if)#ip address 10.255.255.254 ?
  A.B.C.D  IP subnet mask

R1(config-if)#ip address 10.255.255.254 255.0.0.0
R1(config-if)#no shutdown
R1(config-if)#
*Dec  7 08:29:08.937: %LINK-3-UPDOWN: Interface GigabitEthernet0/0, changed state to up
*Dec  7 08:29:09.938: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/
0, changed state to up
R1(config-if)#
```

We first used the *context-sensitive* help, the question mark **?**, to display the next option, and it is the '***subnet mask***'.
This is another name for the '***netmask***'.
As opposed to writing **/8** for this class **A** address, we will have to write out the *subnet mask* in dotted decimal.
-- Remember: The **/8** is equivalent to **255.0.0.0**.
Next, We entered the command (**`no shutdown`**), The command we use to *enable* the interface. As they are being shutdown by the default!
Now, that we enter the command to enable the interface, we got 2 messages on the device.
The first one says '*Interface GigabitEthernet0/0, changed state to up*'. This refers to the Physical Layer status, the '**Status**' column of the (**`show ip interface brief`**).
The second message says '*Line protocol on Interface GigabitEthernet0/0, changed state to up*'. This is the Layer2 status of the interface, the '**Protocol**' column of the (**`show ip interface brief`**) command.
Now, if we gave a look at the (**`show ip interface brief`**) command, both of those columns should display up.

```
R1(config-if)#do sh ip int br
Interface                  IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0         10.255.255.254  YES manual 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
R1(config-if)#
```

We used (**`do`**) to let us execute this privileged exec mode command from interface config mode.
We also used the shortcut instead of using the full command.

We can now see the IP address, the method is displayed as **manual** instead of **unset**, and both the **Status** and **Protocol** display up.
Seems Like our interface configuration was a success!

→ Configurations:

→ Let's configure the **GigabitEthernet0/1** second:
We will give it an Address of **172.16.0.0** and the prefix length will be **/16**, and the ***subnet mask*** will be **255.255.0.0**.
First we used the (**`int g0/1`**) command to enter the *Interface Configuration Mode* for the interface.
Followed by the command (**`ip add 172.16.0.0 255.255.0.0`**) and then with the command (**`no shutdown`**) or just (**`no shut`**).
And then we (**`do sh ip int br`**) to show the IP addresses of the Interfaces to be sure that every thing works fine!

```
R1(config-if)#int g0/1
R1(config-if)#ip add 172.16.255.254 255.255.0.0
R1(config-if)#no shut
R1(config-if)#
*Dec  7 08:51:42.648: %LINK-3-UPDOWN: Interface GigabitEthernet0/1, changed state to up
*Dec  7 08:51:43.649: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/
1, changed state to up
R1(config-if)#do sh ip int br
Interface                  IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0         10.255.255.254  YES manual up                    up
GigabitEthernet0/1         172.16.255.254  YES manual up                    up
GigabitEthernet0/2         unassigned      YES unset  administratively down down
GigabitEthernet0/3         unassigned      YES unset  administratively down down
R1(config-if)#
```

→ You can directly switch from one interface to the other by the command (**`int`** + interface name).

→ Configurations:

→ Let's configure the **GigabitEthernet0/2** finally:
Same Like the others!

```
R1(config-if)#int g0/2
R1(config-if)#ip add 192.168.0.254 255.255.255.0
R1(config-if)#no shut
R1(config-if)#
*Dec  7 09:05:41.505: %LINK-3-UPDOWN: Interface GigabitEthernet0/2, changed state to up
R1(config-if)#
*Dec  7 09:05:42.505: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/
2, changed state to up
R1(config-if)#do sh ip int br
Interface                  IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0         10.255.255.254  YES manual up                    up
GigabitEthernet0/1         172.16.255.254  YES manual up                    up
GigabitEthernet0/2         192.168.0.254   YES manual up                    up
GigabitEthernet0/3         unassigned      YES unset  administratively down down
R1(config-if)#
```

→ Some more '`show`' commands:

They can be used to check out the interfaces on a Cisco device:

→ (**`show interfaces`** + interface): It shows a lot of information for each interface, This command show primarily Layer1 and Layer2 information about the interface, but also some Layer3.

```
R1#show interfaces g0/0
GigabitEthernet0/0 is up, line protocol is up
  Hardware is iGbE, address is 0c1b.8444.f000 (bia 0c1b.8444.f000)
  Internet address is 10.255.255.254/8
  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 RJ45
  output flow-control is unsupported, input flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:06, output 00:00:05, 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: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     167 packets input, 30159 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog, 0 multicast, 0 pause input
     350 packets output, 39097 bytes, 0 underruns
     0 output errors, 0 collisions, 2 interface resets
     105 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     1 lost carrier, 0 no carrier, 0 pause output
     0 output buffer failures, 0 output buffers swapped out
```

--NOTE: That the MAC Address is mentioned twice, that's because the **B.I.A** is the actual Physical Address of the device, but you can also configure a different MAC Address in the **CLI**, although usually you won't configure a different MAC Address.

→ (**`show interfaces description`**): like the (**`do sh ip int br`**) command, But it also has this '**Description**' column. Interface description are *optional*, but can be very helpful in identifying the purpose of each interface.

→ Let's configure descriptions on each of these interfaces:

From *Global Configuration Mode*, We enter the *Interface Configuration Mode* for **G0/0**.

Now we use the command (**`description`** + description)! Or just (**`desc`** + description)

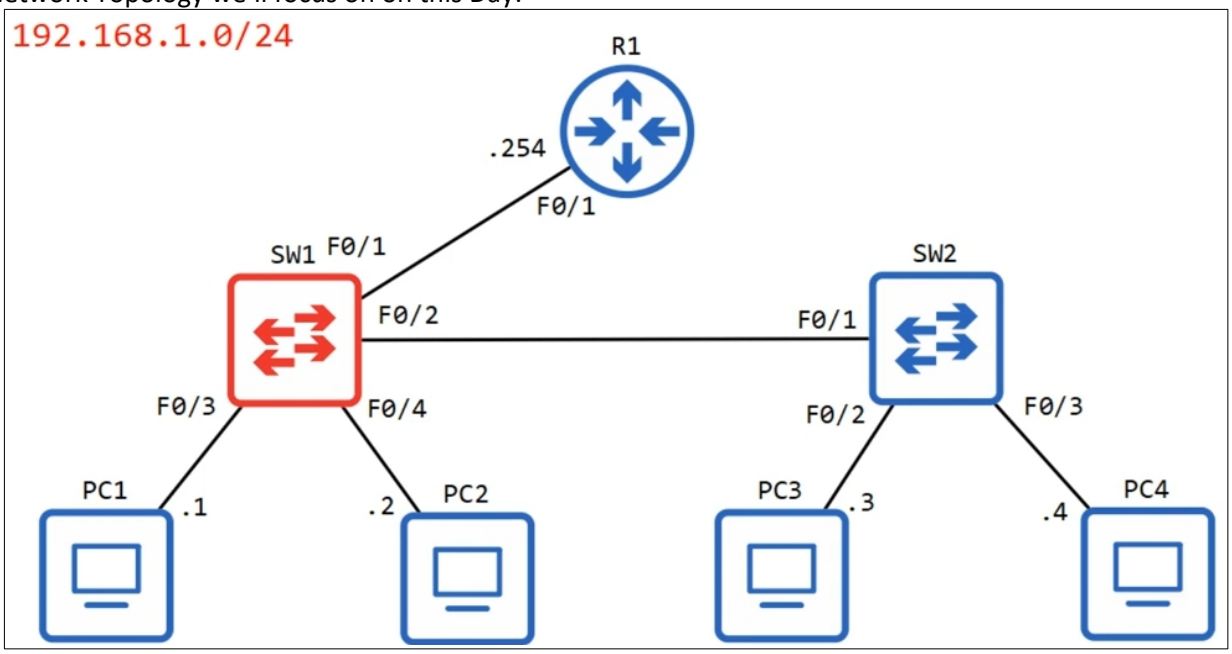
```
R1(config)#int g0/0
R1(config-if)#description ## to SW1 ##
R1(config-if)#int g0/1
R1(config-if)#desc ## to SW2 ##
R1(config-if)#int g0/2
R1(config-if)#desc ## to SW3 ##
R1(config-if)#do sh int desc
Interface                  Status         Protocol Description
Gi0/0                      up             up       ## to SW1 ##
Gi0/1                      up             up       ## to SW2 ##
Gi0/2                      up             up       ## to SW3 ##
Gi0/3                      admin down     down
```

-- Remember: There are many **show** commands that can be used, but for now just remember these three:

- **`show ip interface brief`**
- **`show interfaces`**
- **`show interfaces description`**

Day 9: Switches Interfaces :-

This is the Network Topology we'll focus on on this Day:



This is a single LAN, **192.168.1.0/24**, We are going to focus on **Switch1**, configuring its network interfaces.

→ CLI of SW1:

First, We enter the Privileged Exec Mode with the (**en**) command, and then (**sh ip int br**), we will note that there are some differences here:

```
SW1>en
SW1#sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
Vlan 1                 unassigned      YES unset  up                    up
FastEthernet0/1        unassigned      YES unset  up                    up
FastEthernet0/2        unassigned      YES unset  up                    up
FastEthernet0/3        unassigned      YES unset  up                    up
FastEthernet0/4        unassigned      YES unset  up                    up
FastEthernet0/5        unassigned      YES unset  down                  down
FastEthernet0/6        unassigned      YES unset  down                  down
FastEthernet0/7        unassigned      YES unset  down                  down
FastEthernet0/8        unassigned      YES unset  down                  down
FastEthernet0/9        unassigned      YES unset  down                  down
FastEthernet0/10       unassigned      YES unset  down                  down
FastEthernet0/11       unassigned      YES unset  down                  down
FastEthernet0/12       unassigned      YES unset  down                  down
```

As you can see, the Four Interfaces which are connected to devices have a **Status** column, which is the Layer1 status, and a **Protocol** column, which is the Layer2 status of **up**/**up**.

Without any done configurations on **SW1**, Now we can see a difference between Cisco Routers and Switches!

-- Router interfaces are in an administratively disabled state by default meaning they have the '**shutdown**' command applied.

-- Switch interfaces, however, don't have the shutdown command applied, so if you connect them to another device they will usually be in the **up**/**up** state with no configurations required.

Now, the IP address is *unassigned*, and it will remain that way because these are *Layer2 Switch-Port*, they don't need an IP address.

** → The concept of MultiLayer Switching, where you actually *do* assign IP addresses to Switches, will be for a future Lesson ← **

** → Also the VLAN interface, will be a topic for another Lesson ← **

The other interfaces aren't connected to any other devices, so their status is **down/down**.
-- The difference between **administratively down** and **down**, is because of the ***shutdown*** command!

To summarize:

> **Router** interfaces have the `shutdown` command applied by default
> =will be in the `administratively down/down` state by default
>
> **Switch** interfaces do NOT have the '`shutdown`' command applied by default
> =will be in the `up/up` state if connected to another device
> OR
> in the `down/down` state if not connected to another device

Let's Look an another useful command to check on the Switch interfaces:
That is '**`show interfaces status`**':

```
SW1#show interfaces status
Port      Name               Status       Vlan       Duplex  Speed Type
Fa0/1                        connected    1          a-full  a-100 10/100BaseTX
Fa0/2                        connected    trunk      a-full  a-100 10/100BaseTX
Fa0/3                        connected    1          a-full  a-100 10/100BaseTX
Fa0/4                        connected    1          a-full  a-100 10/100BaseTX
Fa0/5                        notconnect   1            auto   auto 10/100BaseTX
Fa0/6                        notconnect   1            auto   auto 10/100BaseTX
Fa0/7                        notconnect   1            auto   auto 10/100BaseTX
Fa0/8                        notconnect   1            auto   auto 10/100BaseTX
Fa0/9                        notconnect   1            auto   auto 10/100BaseTX
Fa0/10                       notconnect   1            auto   auto 10/100BaseTX
Fa0/11                       notconnect   1            auto   auto 10/100BaseTX
Fa0/12                       notconnect   1            auto   auto 10/100BaseTX
```

→ Let's Check every field:

- The **Port** field, simply lists each interfaces.
- The **Name** field, it's the description of the interface.
- The **Status** field, different than the '**`sh ip int br`**', The connected interfaces show a status of *connected* and the unconnected interfaces show a status of *notconnect*.
- The **Vlan** field, it can be used to divide LANs into smaller LANs. The *default* Vlan is **1**, The only one that doesn't show a **Vlan** of **1** is **F0/2**, which shows ***trunk***. Just take note of the fact that the interface connected to the other Switch, **SW2**, is a ***trunk*** interface.
- The **Duplex** field, indicates whether the device is capable of both sending and receiving data at the same time, which is known as *full-duplex*, or it is not which is called *half-duplex*. **Duplex** is ***auto*** be default on Cisco Switches, meaning it will negotiate with the neighboring device and use *full-duplex* if possible. Note that all of the unconnected interfaces have a duplex of **auto**, and the connected interfaces have a duplex of '**a-full**', **a** stands for auto, and it means that it automatically negotiated a duplex of *full* with the neighboring device
- The **Speed** field, which is also **auto** by default. These are **FastEthernet** interfaces, so they are capable of speeds up to **100** megabits per second. However, they are also capable of operating at **10** megabits per second. **Auto** means they're able to negotiate with the device they are connected to and use the fastest speed both devices are capable of. '**a-100**' means a speed of **100** megabits per second was '*auto-negotiated*' with the neighboring device.
- The **Type** field, These are all **RJ45** interfaces for copper **UTP** cables, but if they were small form-factor pluggable, or **SFP** modules, you'd see that here instead. In this case, we see **10/100BaseTX**, the **10/100** referring to the speeds at which these interfaces can operate.

→ Configuring Interface Speed and Duplex:
Auto-negotiation works well, so usually you'll leave it be, but let's go and manually configure the **Speed** and **Duplex** of an interface, **F0/1** which is connected to **R1**:

```
SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#int f0/1
SW1(config-if)#speed ?
10                        Force 10 Mbps operation
100                       Force 100 Mbps operation
auto                      Enable AUTO speed configuration
SW1(config-if)#speed 100
SW1(config-if)#duplex ?
auto                      Enable AUTO duplex configuration
full                      Force full duplex operation
half                      Force half-duplex operation
SW1(config-if)#duplex full
SW1(config-if)#description ## to R1 ##
```

-- First, we entered the *Configuration Mode* and then used the command (**`speed ?`**) to allow options and then set it to **100** since **R1**'s interface is a FastEthernet also.
-- Then the command (**`duplex ?`**) to get the options, both **SW1** and **R1** support *full-duplex*, so we choose the *full* option.
-- Finally, we configured the description of the interface.

Okay, so when we use the command (**`show interfaces status`**) you can see both the configured **Speed** and **Duplex**:

```
SW1#sh int status
Port      Name               Status       Vlan       Duplex  Speed Type
Fa0/1     ## to R1 ##        connected    1            full    100 10/100BaseTX
Fa0/2                        connected    trunk      a-full  a-100 10/100BaseTX
Fa0/3                        connected    1          a-full  a-100 10/100BaseTX
```

The **Speed** is **100** rather than **a-100** and the **Duplex** is **full** rather than **a-full**, because they are not being *auto-negotiated* any more. Normally, You'll keep *auto-negotiation* **on**, but if perhaps there is some problem and it's not working, you should know how to manually configure the speed and the duplex of an interface.

Okay, Now What about the *unconnected* interfaces??
-- Although the fact that Switch interfaces are enabled by default is convenient, as you can just plug a device in and use it straight away, it can be a *Security Concern*! Really, You should disable the interfaces!
-- Fortunately, instead of having to configure each of the **8** interfaces one by one, there is a way to configure **8** interfaces at once. Here's a command that can save you a lot of time (**`interface range f0/5 – 12`**) and we then (**`shutdown`**) the interfaces.

```
SW1(config)#interface range f0/5 - 12
SW1(config-if-range)#description ## not in use ##
SW1(config-if-range)#shutdown
00:42:36: %LINK-5-CHANGED: Interface FastEthernet0/5, changed state to administratively down
00:42:36: %LINK-5-CHANGED: Interface FastEthernet0/6, changed state to administratively down
00:42:36: %LINK-5-CHANGED: Interface FastEthernet0/7, changed state to administratively down
00:42:36: %LINK-5-CHANGED: Interface FastEthernet0/8, changed state to administratively down
00:42:36: %LINK-5-CHANGED: Interface FastEthernet0/9, changed state to administratively down
00:42:36: %LINK-5-CHANGED: Interface FastEthernet0/10, changed state to administratively down
00:42:36: %LINK-5-CHANGED: Interface FastEthernet0/11, changed state to administratively down
00:42:36: %LINK-5-CHANGED: Interface FastEthernet0/12, changed state to administratively down
SW1(config-if-range)#
```

From *Global Config Mode*, type the command and then you entered the range of the interfaces range config mode and set them a uni-description that they are all not working and then make all of them shutdown!

The interfaces in the range don't all have to be consecutive, you can choose your own like this:

```
SW1(config)#int range f0/5 - 6, f0/9 - 12
SW1(config-if-range)#no shut
00:57:07: %LINK-3-UPDOWN: Interface FastEthernet0/5, changed state to up
00:57:07: %LINK-3-UPDOWN: Interface FastEthernet0/6, changed state to up
00:57:07: %LINK-3-UPDOWN: Interface FastEthernet0/9, changed state to up
00:57:07: %LINK-3-UPDOWN: Interface FastEthernet0/10, changed state to up
00:57:07: %LINK-3-UPDOWN: Interface FastEthernet0/11, changed state to up
00:57:07: %LINK-3-UPDOWN: Interface FastEthernet0/12, changed state to up
```

Now you can see that only the interfaces in the range are enabled.

So, we shutdown the interfaces from **f0/5** to **f0/12** again, and here's the output of the show command:
We can see the description and the status is now *disabled* rather than *notconnect*.

```
SW1(config-if-range)#do sh int status
Port    Name               Status       Vlan       Duplex  Speed Type
Fa0/1   ## to R1 ##        connected    1            full    100 10/100BaseTX
Fa0/2   ## to SW2 ##       connected    trunk      a-full  a-100 10/100BaseTX
Fa0/3   ## to end hosts ## connected    1          a-full  a-100 10/100BaseTX
Fa0/4   ## to end hosts ## connected    1          a-full  a-100 10/100BaseTX
Fa0/5   ## not in use ##   disabled     1            auto   auto 10/100BaseTX
Fa0/6   ## not in use ##   disabled     1            auto   auto 10/100BaseTX
Fa0/7   ## not in use ##   disabled     1            auto   auto 10/100BaseTX
Fa0/8   ## not in use ##   disabled     1            auto   auto 10/100BaseTX
Fa0/9   ## not in use ##   disabled     1            auto   auto 10/100BaseTX
Fa0/10  ## not in use ##   disabled     1            auto   auto 10/100BaseTX
Fa0/11  ## not in use ##   disabled     1            auto   auto 10/100BaseTX
Fa0/12  ## not in use ##   disabled     1            auto   auto 10/100BaseTX
```

→ Full/Half Duplex:

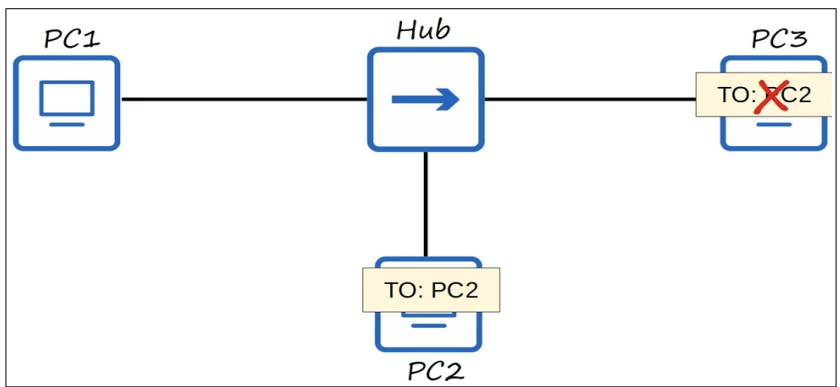
→ **Half Duplex:** means that The device can not send and receive data at the same time. If it is receiving a frame, it must wait before sending a frame.

→ **Full Duplex:** The device can send and receive data at the same time. It doesn't have to wait. It is the *preferred* way to go!
In modern networks that use Switches, all devices can use *full-duplex* on their interfaces.

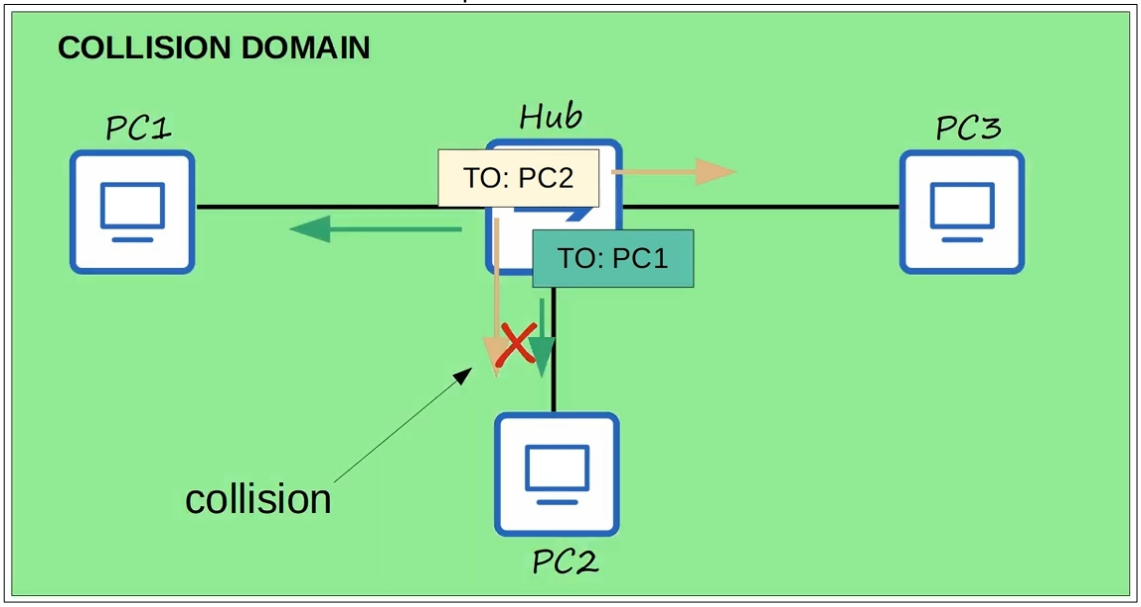
→ LAN Hubs:

So, where is *half-duplex* used? Well, in modern day networks, almost nowhere!
To understand, let's introduce an old type of network device which was around before the network Switch:

The **Hub** is much simpler than a Switch, in fact it is simply a *repeater*. Any frame it receives it *floods* like a Switch does with a *broadcast* or *unknown unicast* frame.
For example, If **PC1** wants to send a frame to **PC2**, it will send the frame out of its network interface, after the **Hub** receives it, it will *repeat* the frame out of its other interfaces, to **PC2** and **PC3**. **PC3** will recognize that the destination MAC address doesn't match with its own and ignore the frame, and **PC2** will receive it normally.



-- Now, what if two PCs tried to send a frame at the same time? In this case, **PC1** is trying to send a frame to **PC2**, and **PC3** trying to send a frame to **PC1**, They both send the frame out of their network interfaces, and this is where the problem occurs!
-- The **Hub** won't send one first and the other after, it simply tries to *flood* both at the same time, and this will result a ***collision*** on the interface, and **PC2** won't receive either frame intact.
-- All devices connected to a **Hub** are part of what's called a ***Collision Domain***.

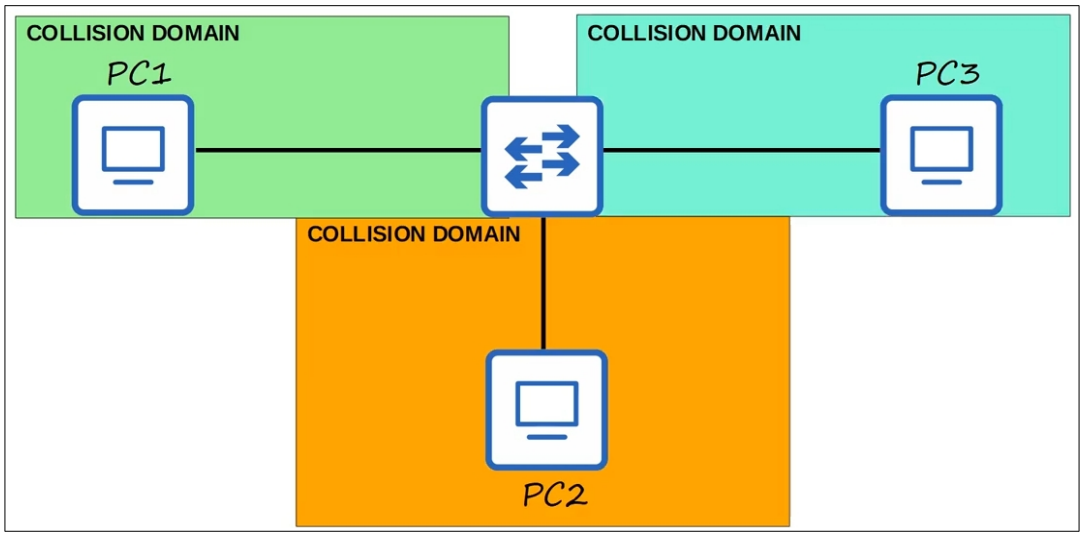


-- The frame they sent could *collide* with frames any of the other devices connected to the **Hub** send.
-- To deal with *collision* in a *half-duplex* situation like this, Ethernet devices use a mechanism called '**CSMA/CD**'.

→ CSMA/CD
-- stands for **Carrier Sense Multiple Access with Collision Detection**.
-- Describes how devices avoid collisions in a *half-duplex* situation, and how they react if collisions do occur.
-- Describes how deices using *half-duplex* listen for activity on a network segment, and then send data only when other devices aren't sending
-- Also describes how a device will react when a collision does occur.
-- How it works: Before sending frames, devices ***listen*** to the collision domain until they detect that other devices are not sending.
If a collision does occur, which can still happen because of bad timing and such, the device sends a jamming signal to inform the other devices that a collision happened!
Each device then waits a random period of time before sending frames again.
The process then *repeats*, with each device listening to check if other devices are sending frames before sending their own frames.
-- That Process works, and it was how networks operated for a long time.

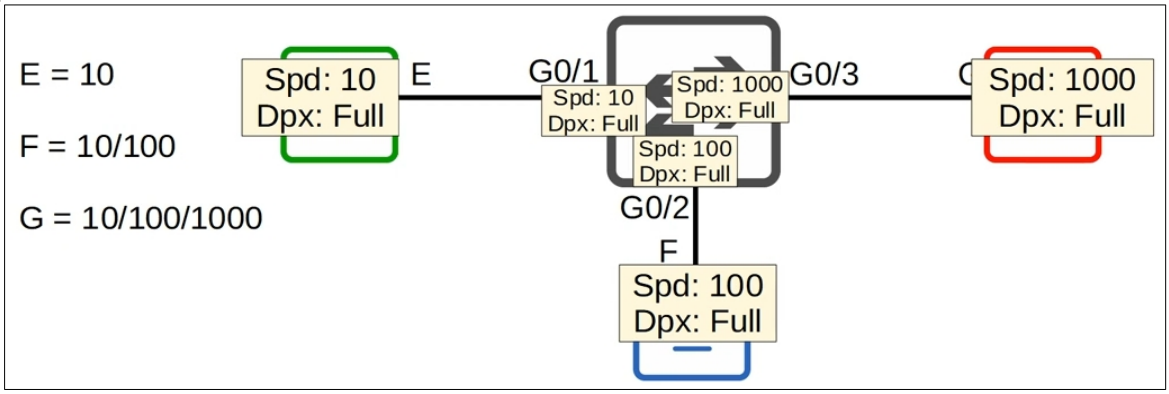
→ Switches and Hubs:
-- But Switches are more *sophisticated* than Hubs.
-- Hubs are simple repeaters which operate at Layer1, repeating whatever signals they receive. Switches operate at Layer2, using Layer2 addressing, MAC addresses, to send frames to specific hosts. They also won't send 2 frames to the host at once.
-- Devices attached to a Hub must operate in *half-duplex*.
-- Devices attached to a Switch can operate in *full-duplex*.

So the previous network, which was *one* collision domain when connected to a Hub, is now *three* collision domains.



-- Because of the improved functionality of Switches over Hubs, these devices can now operate in *full-duplex*, meaning they don't have to worry about whether or not other devices are sending data at the same time, they can send data freely.
-- Although problems like collision still **do** occur, they are usually rare and usually are a sign of a problem, like a *misconfiguration*, rather than a regular occurrence like in a *half-duplex* network.

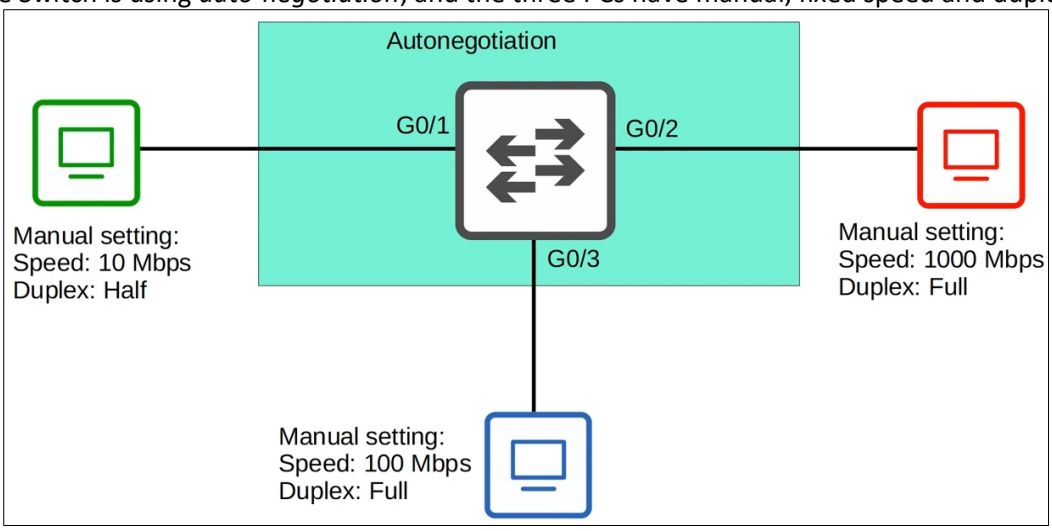
→ **Speed/Duplex *Auto-negotiation:*** This apply to both Routers and Switches:
-- Interfaces that can run at different speeds (**10**/**100** or **10**/**100**/**1000**), have default settings of **Speed** *auto* and **Duplex** *auto*.
-- Interfaces '*advertise*' their capabilities to the neighboring device, and they neogtiate the best **speed** and **duplex** settings they are both capable of.
-- <u>For example</u>:



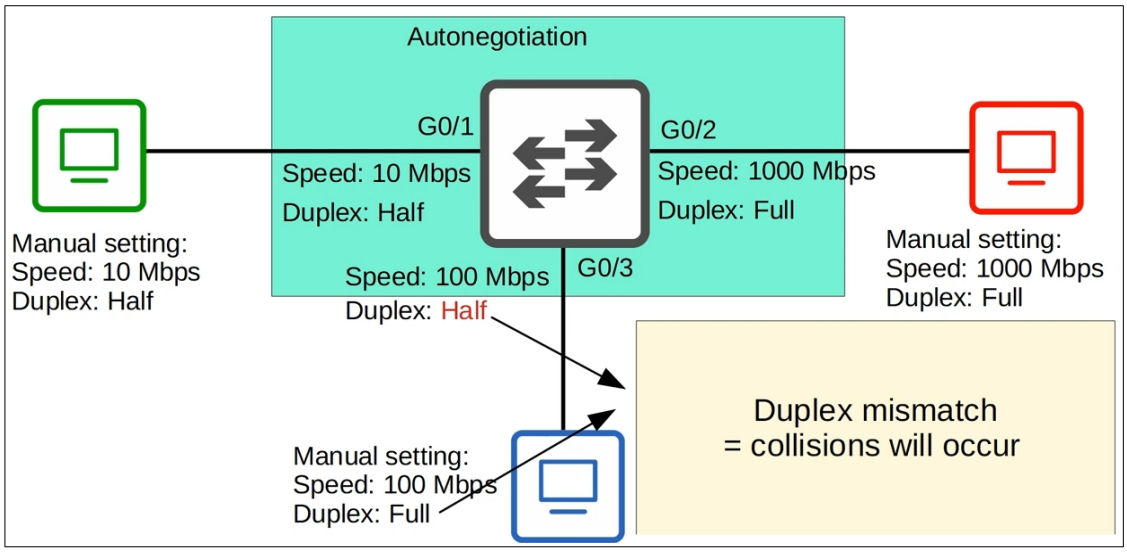
G0/1 and the PC will negotiate to a speed of **10** megabits per second and *full-duplex*.
G0/2 and the PC will negotiate to a speed of **100** megabits per second and *full-duplex*.
G0/3 and the PC will negotiate to a speed of **1000** megabits per second and *full-duplex*.
-- The PCs are all able to use the Max speed of the network interfaces, and the Switch adjusts the speed of its interfaces to match.
-- In a network like this with all PCs and Switches, there is no reason to use *half-duplex*, so they all negotiate to use *full-duplex*.

-- <u>Another situation</u>: What if *auto-negotiation* is disabled on the device connected to the Switch?
So, the Switch is trying to auto-negotiate, but the other devices don't respond. This is how the Switch reply:-
For **Speed**: The Switch will try the sense the speed that the other device is operating at. If it fails to sense, it will use the slowest supported speed (ie. **10Mbps** on a **10**/**100**/**1000** interface)
For **Duplex**: If the speed is **10** or **100 Mbps**, the Switch will use *half-duplex*.
If the speed is **1000 Mbps** or grater, it will use the *full-duplex*.

-- Let's see how this works:
In this case, only the Switch is using *auto-negotiation*, and the three PCs have manual, fixed speed and duplex settings.



We'll also assume that the Switch successfully detects the speed that the PCs are using.
-- The Switch detects the speed of Green PC and set speed to the same **10 Mbps**, and because of that it sets duplex to *half-duplex*.
-- The Switch senses the Red PC using **1000 Mbps**, so it uses the same. And because of that it uses *full-duplex*.-
-- How about the Blue PC connected to **G0/3**? The Switch senses the speed of **100 Mbps**, but then what about the duplex??
The Blue PC is using *full-duplex*, but without *auto-negotiation* the Switch can't sense that!
So, because the speed is **100 Mbps**, the Switch uses *half-duplex*.
And this results in a *duplex mismatch*!, which will cause collisions to occur, resulting in poor network performance.
So, really you should be using *auto-negotiation* on all devices in the network.



Quiz-Note: The *full-duplex* is unaware of that the *half-duplex* is unable to send and receive data ate the same time, and will send data even if the *half-duplex* side isn't ready to receive it, causing *collisions*!

→ Interface Errors:

Let's take a look at some of the errors that can show up on interfaces that otherwise seem to be working:
The Switch will take count of some of these things and you can view them with the command (**show interfaces** + interface).
We took a look at the command when we were talking about Routers, so these things aren't specific of Switch interfaces!
-- This time, let's focus on some of these statistics at the bottom:

```
SW1#show interfaces f0/2
FastEthernet0/2 is up, line protocol is up
  Hardware is Fast Ethernet, address is 000C.3168.8461 (bia 000C.3168.8461)
  Description: ## to SW2 ##
  MTU 1500 bytes, BW 100000 Kbit, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Auto-duplex, Auto-speed
  Encapsulation ARPA, loopback not set
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 02:29:44, output never, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queuing strategy: fifo
  Output queue :0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     269 packets input, 71059 bytes, 0 no buffer
     Received 6 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     7290 packets output, 429075 bytes, 0 underruns
     0 output errors, 3 interface resets
     0 output buffer failures, 0 output buffers swapped out
```

There is a lot of different kinds of counters shown here and you don't have to know all of them at this point.
So, let's just focus on some:

- First up, not errors, but you can see the total number of packets received on the interface and the total number of bytes in those packets.
- **Runts**, Which is frames that are smaller than the minimum Ethernet frame size (**64** bytes).
- **Giants**, Frames that are larger than the maximum Ethernet frame size (**1518** bytes).
- **CRC**, which counts frames that failed their CRC Check (in the Ethernet **FCS trailer**).
- **Frame**, Which counts frames that have an incorrect or illegal format (due to an error).
- **Input errors**, Total of various counters, including the above four counters.
- **Output errors**, counts frames the Switch tried to send, but failed due to an error.

```
269 packets input, 71059 bytes, 0 no buffer
Received 6 broadcasts, 0 runts, 0 giants, 0 throttles
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
7290 packets output, 429075 bytes, 0 underruns
0 output errors, 3 interface resets
0 output buffer failures, 0 output buffers swapped out
```

Keep in mind, these counters are shown on the Switch, but they are the same on a Router.

Day 10: IPv4 Header :-

-- The IP is used in the Layer3, to help send the data between devices in separate networks, even on other sides of the world over the Internet, this is known as ***Routing***.

We will focus exclusively on the fields in the **IPv4 Header**.
Or just the Layer3 Header, as it contains the information that is needed to route this packet to its destination.

Since we'll be focusing on Layer3, We will usually be using the term ***Packet***, talking about ***routing packets***, rather than ***frames***.

→ IPv4 Header:

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

To read this chart, it's read *left-to-right*, *top-to-bottom*.

→ **Version** field:

- It is **4** bits in length, half of one *octet*.
- It's purpose is *straight forward*. It identifies the version of IP used. There are only **2** versions of IP in use, **IPv4** and **IPv6**.
- **IPv4**, which is simply identified with a value of **4** in this field, or **0100** in binary.
- **IPv6**, which is identified with a value of **6**, or **0110** in binary.
- Since we're focusing on the **IPv4 Header**, this value will be always **4**, since the **IPv6** has a different structure.

→ **Internet Header Length**, or **IHL** field:

- It is also **4** bits in length, half of one *octet*.
- The final field of the **IPv4 Header**, **Options**, is *variable* in length, so this field is necessary to indicate the total length of the header.
- This field specifies the length of the Header ***in 4-byte increments***. For example, if the value in this field is **5**, *Value of **5** = **5 x 4**-byte = **20** bytes*, meaning the length of the Header is ***20 bytes***.
- Minimum value in this field is **5**, which is equal to **= 20** bytes. That's the length of the **IP** Packet without any **IP** options at the end, so an empty options field.
- Maximum value is **15**, which is the maximum value of **4** bits. **15** (**15** x **4**-bytes = **60** bytes).
- As you can see the value of the **4** bits are **1**, **2**, **4** and **8**, adding them up results in **15**.
- That means the maximum length of the **IP Options** field is, **40 bytes**.
- AN **IPv4 Header** with no **Options** field, is **20 bytes** in length, and that's the minimum length of an **IPv4 Header**.
- AN **IPv4 Header** with a maximum length **Options** field, **40 bytes**, has a length of **60 bytes**. And that's the maximum length of an **IPv4 Header**.
- It ***Only*** indicates the length of the **IPv4 Header** itself.

1	1	1	1	
8	+ 4	+ 2	+ 1	= 15

→ **DSCP** field:

- **DSCP**, stands for '***Differentiated Services Code Point***'.
- It's length is **6** bits.
- This field is used for **Qos** (**Quality of Service**).
- It is used to prioritize delay-sensitive data, such as streaming voice and video etc. If you're loading a web page and the Internet is a little slow, it is not a big deal. However, if you're having a Skype call and there's terrible delay, or the audio and video keep freezing, It can totally ruin the experience.
- This field is used to identify which traffic should receive priority treatment.

→ **ECN** field:

- stands for '**Explicit Congestion Notification**'.
- **2** bits in length.
- Provide *end-to-end (*between two endpoints*)* notification of network congestion without dropping **Packets**.
- Normally in a network, if the network is super busy, if there is congestion, this is *signalled* by dropping packets.
- The **ECN** field provides a way to signal a congested network without dropping packets.
- However, this is an *optional* field that requires both endpoints, as well as the underlying network infrastructure, to support it.

→ **Total Length** field:

- Its length is **16** bits, or **2** octets, or **2** bytes.
- Indicates the Total Length of the Packet (**L3 header** + **L4 segment**). → Layer4 segment includes the L4 header + Data.
- Indicates the length in **bytes** (not **4-byte** increments like the **IHL** field). So, a value of **20** in his field simply means **20** bytes.
- Minimum value of this field is **20**, meaning **20** bytes, which is equal to a minimum-size **IPv4** header with no encapsulated data.
- The maximum value is **65**,**535**, which is the maximum value of **16** binary bits, all set to **1**.

→ **Identification** field:

- Length is **16** bits.
- If a packet is fragmented due to being too large, this field is used to identify which *packet* the fragment belongs to. So it can be re-assembled again to make the original packet.
- All fragments of the same packet will have their own **IPv4 header** with the same value in this field. So they can be re-assembled later.
- Packets are fragmented if larger than the **MTU**, which stands for, **Maximum Transmission Unit**.
- **MTU** is usually **1500** bytes.
- The Maximum Payload size of an Ethernet frame is **1500** bytes, so these are related.
- Fragments are re-assembled by the receiving host.

→ **Flags** field:

- Length is **3** bits.
- It is used to control/identify fragments.
- The **3** bits are functioning like this:
    - **Bit 0**: Reserved, always set to **0** or just '**Not set**'.
    - **Bit 1**: Don't fragment (**DF** bit), If it is set to **1,** it is used to indicate a packet that should not be fragmented or you may see it '**Set**' if it is set rather than **1**.
    - **Bit 2**: More fragments (**MF** bit), It is set to **1** if there are fragments in the packet, and then set to **0** for the last fragment.
    - Also, if the packet is a whole, unfragmented packet, the **MF** bit is set to **0**, since there are no fragments.

→ **Fragment Offset** field:

- It is **13** bits in length.
- Indicates the position of the fragment within the original, unfragmented IP packet.
- Allows fragmented packets to be re-assembled even if the fragments arrive out of order. Since this field lets the receiver know the original order of the fragments.

→ **TTL, Time To Live** field:

- Length is **8** bits. You may see it have a value of **255**, the maximum **8**-bit value.
- A Router will drop a packet with a **TTL** of **0**.
- This field is used to *prevent* infinite loops. If a poor routing configuration causes a packet to be continually sent around in a loop, never reaching its intended destination, if enough traffic like that accumulates, it could causes network congestion, and eventually failure.
- This field prevents that from happening, causing looped traffic to be dropped when the **TTL** reaches **0**.
- This field was originally designed to indicate the packet's maximum lifetime in seconds.
- In practice, However, this actually indicates a '**hop count**'. Each time the packet arrives at a Router on the way to its destination, the router decreases the **TTL** by **1**, until the packet reaches its destination, or the **TTL** reaches **0** and the packet is dropped.
- The current recommend default **TTL** is **64**.
- To summarize, It is reduced by **1** at each each Router the packet passes through. If it reaches **0**, the packet is dropped.

→ **Protocol** field:

- It is **8** bits in length.
- Indicates the protocol of the encapsulated Layer4 PDU.
- Typically, this will be one of the following:
  - **TCP**, which is indicated by a value of **6**.
  - **UDP**, which is indicated by a value of **17**.
  - **ICMP**, which is indicated by a value of **1**.
  - **OSPF**, ***Open Shortest Path First***, which is indicated by a value of **89**.
  - **OSPF**, the Dynamic Routing Protocol, which allows routers to learn routes to the destinations from their neighbors, without us having to manually configure the routes.

→ **Header Checksum** field:

- **16** bits in length.
- A calculated checksum, used to check for errors in the **IPv4 Header**.
- When a Router receives a packet, it calculates the checksum of the header and compares it to the one in this field of the header.
- If the newly calculated checksum and the checksum in the **IPv4 header** do not match, it means that an error has occurred in transmission, so the Router drops the packet.
- This is used to check for errors <u>only</u> in the **IPv4 header**, not in the encapsulated data.
- IP relies on the encapsulated protocol to detect errors in the encapsulated data.
- Both **TCP** and **UDP**, the two Layer4 protocols most likely to be encapsulated, have their own checksum fields to check for/detect errors in the encapsulated data.

→ **Source** and **Destination** fields:

- Both of them is **32** bits in length, as that is the length of an **IPv4** address.
- **Source**: indicates **IPv4** address of the sender of the packet.
- **Destination**: indicates **IPv4** address of the intended receiver of the packet.

→ **Options** field:

- Is an optional field, and can be **0** bits in length if not used, or up to **320** bits, **40** bytes, in length.
- This field is rarely used, however, if the **IHL**, **Internet Header Length**, field is greater than **5**, it means that **Options** are present.
- Here is a chart showing the structure of the **Options** field:

Field	Size (bits)	Description
Copied	1	Set to 1 if the options need to be copied into all fragments of a fragmented packet.
Option Class	2	A general options category. 0 is for "control" options, and 2 is for "debugging and measurement". 1 and 3 are reserved.
Option Number	5	Specifies an option.
Option Length	8	Indicates the size of the entire option (including this field). This field may not exist for simple options.
Option Data	Variable	Option-specific data. This field may not exist for simple options.

NOTE: Just Google and read more about the **IPv4 Header.**

→ Wireshark Packet Capture:

From Minute **17:50** to Minute **24:20**.

Quick Tips:

→ The **Options** field can *vary* in length from **0** bits to **320** bits.
→ The other field are *fixed-length*.
→ Although the **Total Length** and **IHL** fields are used to represent the variable length of the **IPv4 header** and packet, the fields themselves are *fixed* in length.
→ The **More Fragments** bit, part of the **Flags** field of the **IPv4 header**, is used to indicate that the current fragment is *not* the last fragment of a fragmented packet. It is set to **1** on all fragments except the last, which will set it to **0**.
→ **Fragment Offset**, is a **13**-bit field in the header, not a single bit.
→ **Don't Fragment** bit, is used to *prevent* a packet fro being fragmented.

Day 11: Static Routing :-