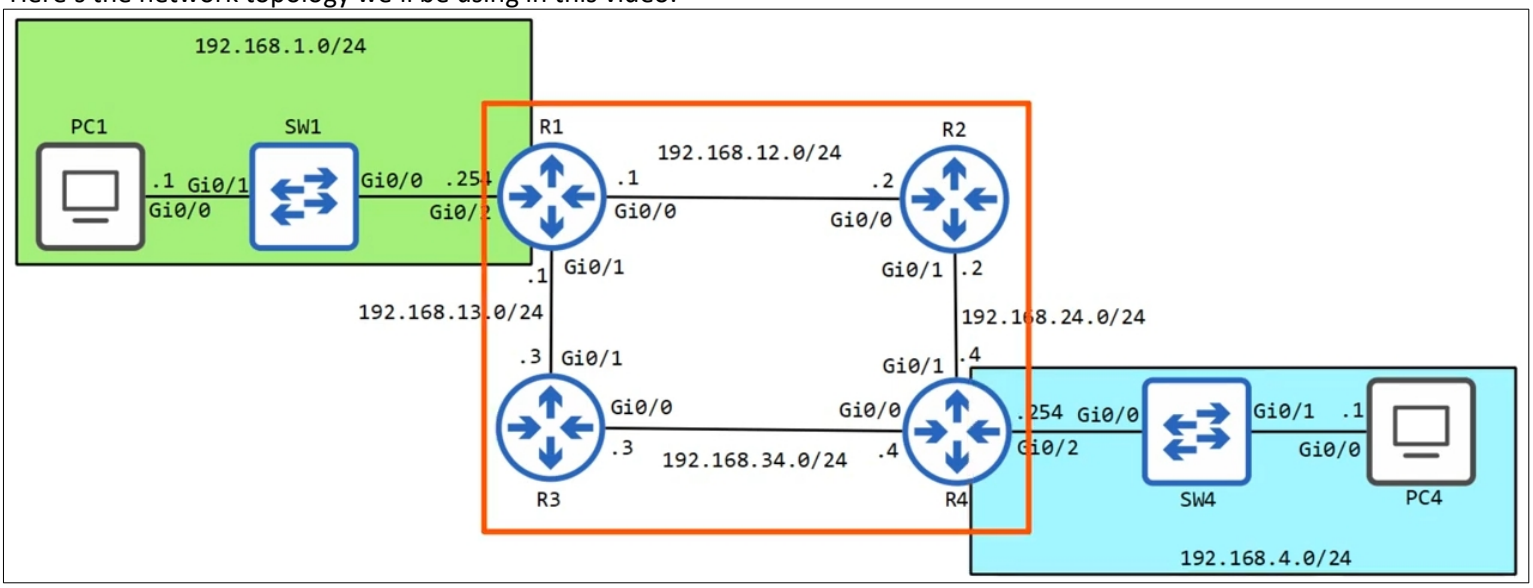
Day 11: Static Routing :-

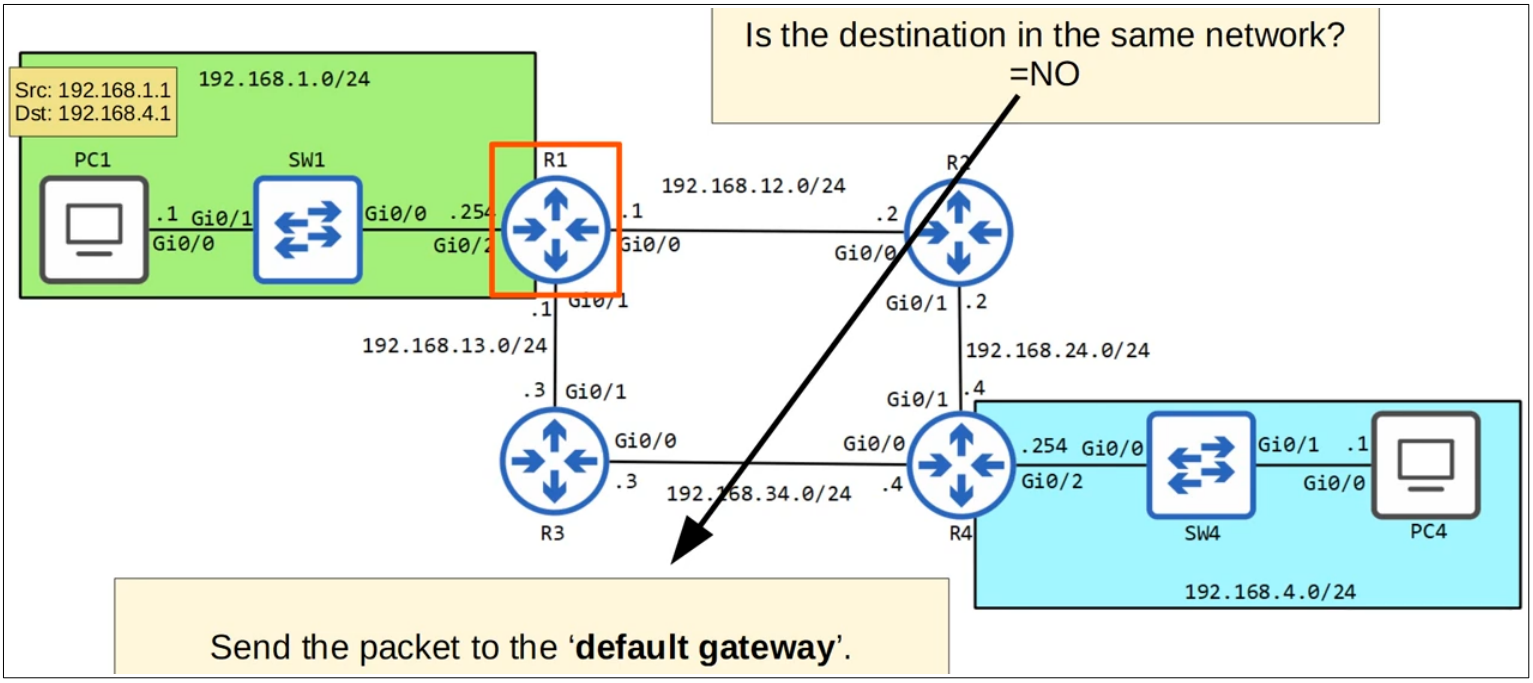
→ IP routing process:

Here's the network topology we'll be using in this video:



Here we have 2 LANs: **192.168.1.0/24** and **192.168.4.0/24**
These 4 Routers represents not a LAN, but rather a **WAN**, which stands for **Wide Area Network**.
→ **WAN** is a network that spreads/extends over a large geographical area.
These Routers could be Kilometers apart, connected by Single-Mode Fiber cable.

So, If **PC1** wants to send a packet to **PC4**, First, **PC1** asks itself "Is the destination IP address in the same network?"
Well, **PC1**'s IP address is **192.168.1.1/24**, and **/24** means that the first **3** octets are the network portion, so **192.168.1** is the network address. In the same boat, **PC4**'s IP address is **192.168.4.1/24** and so on.
The destination is in a different network, a different LAN.
So, the routing logic tells **PC1** to send the packet to the '***default gateway***'.



→ The '***default gateway***' is the device to which the host -**PC1** in this case- will forward data that is destined for another network.
R1 is the *default gateway* for **PC1**.

Routers are the devices used to connect separate networks, for example:
R1 connects the **192.168.1.0/24**, **192.168.12.0/24** and **192.168.13.0/24** networks.
→ So, to reach other networks, an end host will send the packet to its network's Router, the *default gateway*.
Now **R1** has the packet, and it is **R1**'s responsibility to forward it to the next device.
After receiving the packet, **R1** will compare the packet's destination IP address to the ***routing table***.

→ Each Router keeps a *routing table* which stores a list of destinations and how to reach them

Assuming **R1** already has an entry in its *routing table* for the **192.168.4.0/24** network, the entry will look something like this:

```
192.168.4.0/24 via 192.168.12.2, Gi0/0
```

First is the destination and the next is the '***next hop***', the next destination in the path to the **192.168.4.0/24** network, the final destination, and also the interface out of which **R1** will send the packet.
In this case, **R1** could reach the network either by sending it to **R2** or **R3**, but let's say we configured it to send it via **R2**, as you can tell by both the *next-hop* address of **192.168.12.2**, and the exit interface on **R1**, **G0/0**.

→ So, **R1** will forward the packet to the next Router on the route to the destination, which is **R2**.
Now **R2** has the packet, and it is **R2**'s responsibility to forward the packet to the next device.
R2 will follow the same process as **R1**, it will compare the packet's destination IP address to the *routing table* and then, assuming it already has an entry in its *routing table* for the destination, It might look like this:

```
192.168.4.0/24 via 192.168.14.4, Gi0/1
```

The destination is **192.168.4.0/24**, the *next-hop* address is **192.168.24.4** -the **R4**'s IP address- and the exit interface is **G0/1**, which is connected to **R4**.
After looking for the route in the *routing table*, **R2** forwards the packet to **R4**.

→ Now **R4** has the packet, and it is **R4**'s responsibility to forward the packet to the next device.
R4 will follow the same process as both **R1** and **R2**, it will compare the packet's destination IP address to the *routing table*.
In **R4**'s case, the entry in the table will look something like this:

```
192.168.4.0/24 is directly connected, Gi0/2
```

That's because **R4** has an IP address of **192.168.4.254/24** configured on its **G0/2** interface, so it knows the **192.168.4.0/24** network is directly connected to that interface.
It forwards the packet out of that interface, and **SW4** passes it on to **PC4**, the final destination.

→ Notice that we didn't mention MAC addresses, or Layer2 at all. This is just a Layer3 overview of this process.
We'll go in depth later in a video '***Life of a packet***' not just about Layer3, but Layer2 also.

→ The *routing table* on a Cisco Router:

we use the command '**`show ip route`**' from privileged exec mode, and it displays the routing table:

```
PC1#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set

      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, GigabitEthernet0/0
L        192.168.1.1/32 is directly connected, GigabitEthernet0/0
PC1#
```

Down here under the codes at the top, two routes are displayed, this may seem like 3 routes, but this line at the top, **192.168.1.0/24 is variably subntted, 2 subnets, 2 masks**, isn't a route.
This lists the classful address, and what class is **192.168.1.0/24**? It's a Class **C** address.
Underneath are the two routes that fit within that class.

Ok, so we've got 2 routes, and these two letters on the left, **C** and **L**, display the type of route. We can find them in the list up top: **L** stands for **Local**, and **C** stands for **Connected**.

→ Let's look for the **Connected** route first:
It's a route to **192.168.1.0/24**.
The IP address configured on **PC1**'s interface is **192.168.1.1/24**, so **192.168.1.0/24** is the network address, the address with a host portion of all **0**'s. **/24** means that the first three octets are the network portion and the last octet is the host portion.
Set the host portion to **0** and that's the network address. There are probably lots of other hosts on this network, maybe with addresses like **192.168.1.2/24**, **192.168.1.3/24**, **192.168.1.4/24**, etc.
All of them are part of the **192.168.1.0/24** network, because the network portion of their address is the same.
So, **PC1** knows it can reach any host on the **192.168.1.0/24** network via its network interface which is **GigabitEthernet0/0** on this Cisco Router which we're using to simulate a PC.

→ Let's look for the **Local** route second:
Look at this one, **192.168.1.1/32**, which is the exact address on **PC1**'s interface, but this time with a **/32** mask.
/32 would mean that all four octets are the network portion, and there is no host portion.
You could say this is a route to the **192.168.1.1/32** network, and there is only one address in the network, **192.168.1.1**.
This is how you identify a single, specific address, by using a **/32** mask.

→ So, to recap, the **Connected** route indicates the network to which the interface is attached.
The **Local** route indicates the actual address configured on the interface, with a **/32** mask.

Connected route = the network the interface is connected to.
Local route = the actual IP address on the interface (with a /32 mask)

So, that means you already know how to configure two types of routes on a Cisco Router. If you configure an IP address on an interface, the **Connected** and **Local** routes of that interface are added!

→ Configuring a *Default Route*:

However, this **PC1** doesn't have a route to the intended destination, **192.168.4.1**.
Actually, this is not a problem, An end-host like a PC doesn't need to know the route to every destination.
All it needs is a **default-gateway** to which it can send any traffic destined for a location outside of the local network.

On a Cisco Router, the *default-gateway* is known as the '***Gateway of last resort***', and as it says here, it is not set yet.

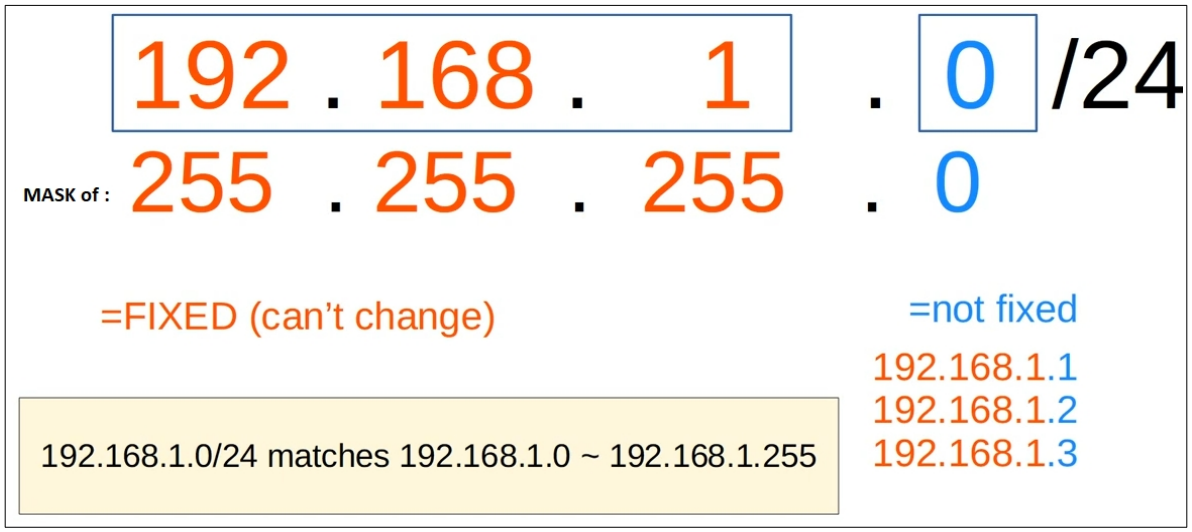
```
Gateway of last resort is not set
```

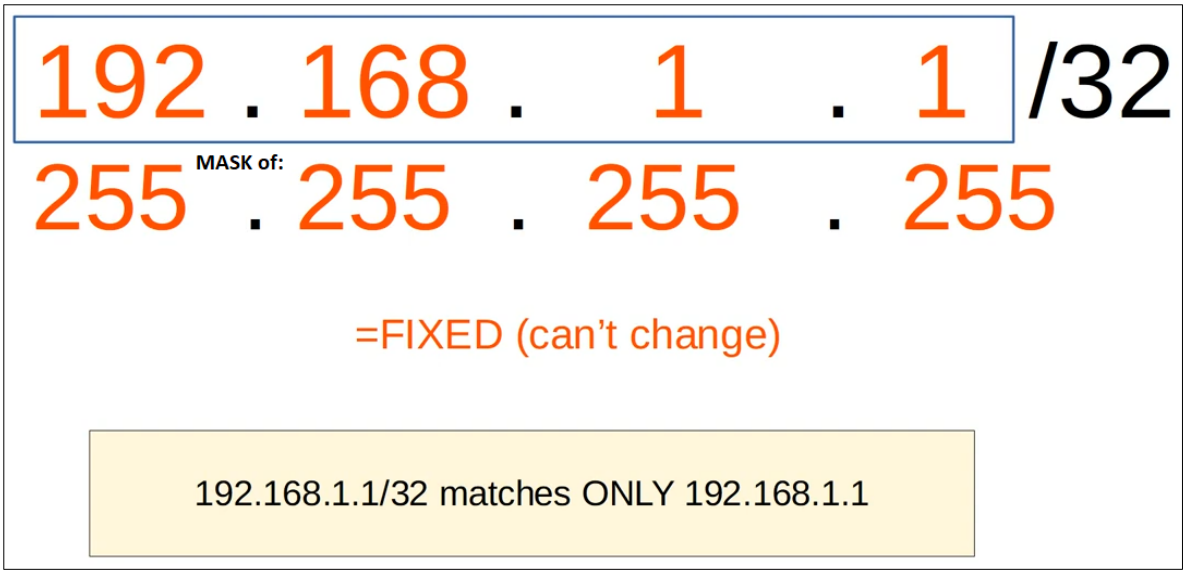
→ So, let's configure the *default-gateway* on our PC, which is actually a Cisco Router.
To configure the G*ateway of last resort*, we must configure a '***default route***'.

→ A '***default route***' is a route that matches **ALL** possible destinations.
It is used only if a more specific route match isn't found in the routing table.
When a Router looks up a route to a destination in its routing table, it looks for the most specific match, and chooses that route.
The *default route* is the LEAST SPECIFIC possible route.
It uses an IP address of: **0.0.0.0**
Mask of: **0.0.0.0**
* To set the *default route/gateway* of **last resort**, configure a route to **0.0.0.0/0**.
* The **/0** means that there is no network portion of the address, no fixed part of the address that can't change.

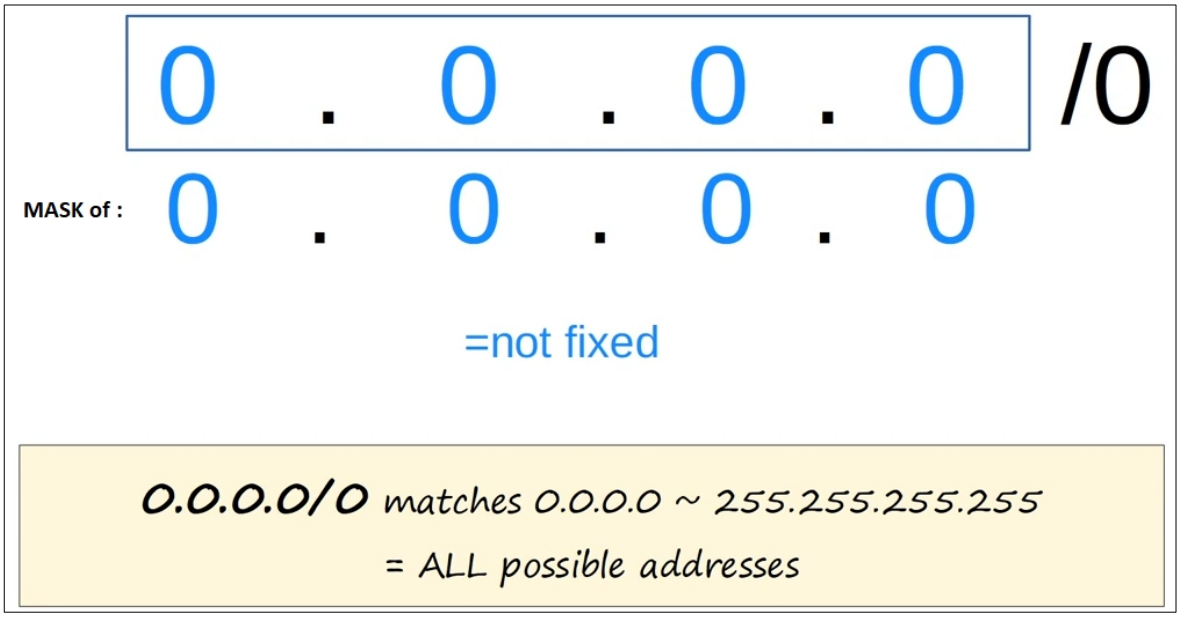
The **0.0.0.0/0** range includes **0.0.0.0** ~ **255.255.255.255 which is = ALL possible addresses**.

→ **Explanation:**





That's why the **/32** mask is used to specify the exact address configured on the interface in the **Local** route.



This is called the LEAST SPECIFIC route. AS It doesn't specify a single address, it includes ALL possible addresses.

→ Configuring a *Static Route*:

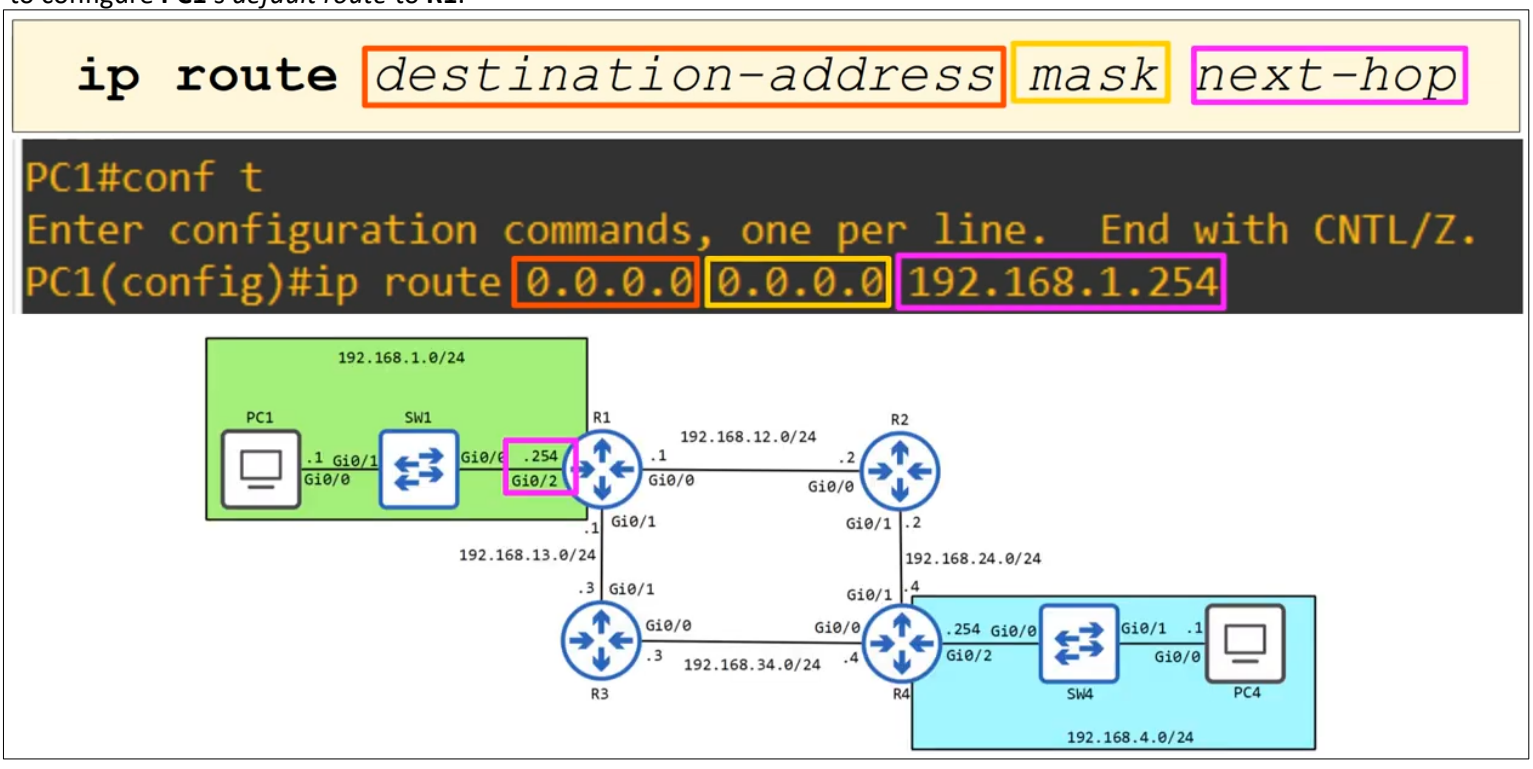
Let's go to actually configuring this *default route*.
To do so, we have to learn how to configure a '***Static Route***'.
A '***Static Route***': is a route you manually configure yourself.
This is different than **Local** and **Connected** routes we saw before, which are automatically added to the routing table when we configure an IP address on an interface and enable it.

Here's the command to configure a *static route:*

```
ip route destination-address mask next-hop
```

```
ip route destination-address mask exit-interface
```

→ So here, we use the configure terminal command to enter Global configuration mode, and then used **`ip route`** command to configure **PC1**'s *default route* to **R1**.



→ So, from global configuration mode we use '**`do show ip route`**' to check the routing table once more:
And first off, you can see it says '***Gateway of last resort is 192.168.1.254 to network 0.0.0.0***'
The '***default route***' also appears down here with the other routes.
NOTE the code on the left is different, we have an **S** and an asterisk *****, If we look up we will see that
S means **Static**, which we just explained before is a route we manually configure on the device.
***** The asterisk means it is a '***candidate default***' route.
The fact that we can see it is added as the gateway of last resort means it isn't just a ***candidate***, it was selected as the *default route*, but it's still labeled as a ***candidate default route***.

→ Looking back at our Topology, Since **PC1** now has a *default gateway* configured, known as a *Gateway of last resort* on a Cisco Router, it sends the packet to the *default gateway*, **R1**.
Since **R1** has the packet now, let's take a look at its routing table to see what it will do with the packet:

→ R1 routing table:

First we use '**`show ip route`**' command:
Because **R1** is connected to three networks, **192.168.1.0/24**, **192.168.12.0/24** and **192.168.13.0/24**, it has **Connected** routes for each of those networks, as well as the **Local** routes for the IP addresses configured on its interfaces.

```
Gateway of last resort is not set

      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, GigabitEthernet0/2
L        192.168.1.254/32 is directly connected, GigabitEthernet0/2
      192.168.12.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.12.0/24 is directly connected, GigabitEthernet0/0
L        192.168.12.1/32 is directly connected, GigabitEthernet0/0
      192.168.13.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.13.0/24 is directly connected, GigabitEthernet0/1
L        192.168.13.1/32 is directly connected, GigabitEthernet0/1
```

However, here's a problem: The packet's destination is **192.168.4.1**, in the **192.168.4.0/24** network, but it is not in the routing table!
If this were a Switch receiving a frame with an unknown destination MAC address, It would flood the frame out of all of its interfaces. How about in this case, what will a Router do when it receives a packet with an unknown destination IP address?
The Router will simply ***drop*** the packet!

Switches **flood** frames with unknown destinations (destinations not in the MAC table).

Routers **drop** packets with unknown destinations.

So, unless we configure a route on **R1**, this ping will not succeed.

So, Let's configure a *Static route* on **R1** aimed for **192.168.4.0/24** network:
This time instead of specifying an IP address as the next hop, we just specify the ***exit-interface***, the interface out of which **R1** should send the packet.

```
ip route destination-address mask exit-interface
R1(config)#ip route 192.168.4.0 255.255.255.0 g0/0
```

Then we use '**`do show ip route`**' command, and here's the route we configured:

```
S       192.168.4.0/24 is directly connected, GigabitEthernet0/0
```

Notice how this static route isn't added as the *Gateway of last resort*, because it is for a specific network, not for **0.0.0.0/0** which matches all networks.

→ Now **R1** has a route in its routing table for the destination.
The destination of this packet is **192.168.4.1** which is included in the **192.168.4.0/24** network.
We don't need a route to the exact destination. Because **192.168.4.1** is included in the **192.168.4.0/24** rage, it is fine, it matches.

→ So, **R1** forwards the packet to **R2**.
And there's a problem here, **R2** also doesn't have a route that matches **192.168.4.1**
So we configure a route on **R2**, by the command '`ip route 192.168.4.0 255.255.255.0 192.168.24.4`'
We specified the next-hop IP address of **192.168.24.4** which is **R4**'s IP address.

```
S       192.168.4.0/24 [1/0] via 192.168.24.4
```

So, **R2** receives the packet with destination **192.168.4.1** and looks up the most specific match in its routing table.
It finds this entry for **192.168.4.0/24** and it says via **192.168.24.4**
R2 then looks up for the most specific match for **192.168.24.4**, that is this route here:

```
C       192.168.24.0/24 is directly connected, GigabitEthernet0/1
```

Which is a directly connected route on **GigabitEthernet0/1**, so it will forward the packet out of that interface.
So, **R2** sends the packet to **R4**.

→ R4 routing table:

We haven't configured any static routes on **R4**, will that be a problem?
Let's check:

```
Gateway of last resort is not set

      192.168.4.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.4.0/24 is directly connected, GigabitEthernet0/2
L        192.168.4.254/32 is directly connected, GigabitEthernet0/2
      192.168.24.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.24.0/24 is directly connected, GigabitEthernet0/1
L        192.168.24.4/32 is directly connected, GigabitEthernet0/1
      192.168.34.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.34.0/24 is directly connected, GigabitEthernet0/0
L        192.168.34.4/32 is directly connected, GigabitEthernet0/0
```

Because **R4**'s **G0/2** interface has an IP address of **192.168.4.254** configured, the **192.168.4.0/24** network has automatically been added to its routing table as a **Connected** route.
→ So, **R4** sends the packet out of its **G0/2** interface, and **SW4** forward it to **PC4**.

→ Ping from PC1 to PC4:

we use the command '**ping 192.168.4.1**' on **PC1**:

```
PC1#ping 192.168.4.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.4.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
PC1#
```

However, as you can see, the ping fails, success rate is 0 percent, 0 out of 5.
PC1 has a *default gateway*, all of the routers along the way know how to reach **PC4**. So, what's the problem??

→ The problem is ***one-way reachability***.
The packet from **PC1** was able to reach **PC4**. **R1**, **R2** and **R4** all have routes to the **192.168.4.0/24** network.
However. To send a reply back, **PC4** sends packets with a source of **192.168.4.1** and a destination of **192.168.1.1**.
The problem is that **R4** and **R2** have no route to reach the **192.168.1.0/24** network.

If we take a look back to **R4** routing table, there is no route to **192.168.1.0/24**. And also **R2**.
Not to mention, we also haven't configured the *default gateway* on **PC4**!

→ So, we configure the *default route* on **PC4** like we did on **PC1**, by using the command
'**`ip route 0.0.0.0 0.0.0.0 192.168.4.254`**'
And it has been selected as the *Gateway of last resort*, the *default gateway*.

```
Gateway of last resort is 192.168.4.254 to network 0.0.0.0

S*    0.0.0.0/0 [1/0] via 192.168.4.254
```

→ So, we also configure a static route to the **192.168.1.0/24** network, and the next-hop is **192.168.24.2**, **R2**'s address.
We use '**`ip route 192.168.1.0 255.255.255.0 192.168.24.2`**'

```
S     192.168.1.0/24 [1/0] via 192.168.24.2
```

→ And finally, **R2**'s configurations:
we use the command '**`ip route 192.168.1.0 255.255.255.0 192.168.12.1`**'

```
S     192.168.1.0/24 [1/0] via 192.168.12.1
```

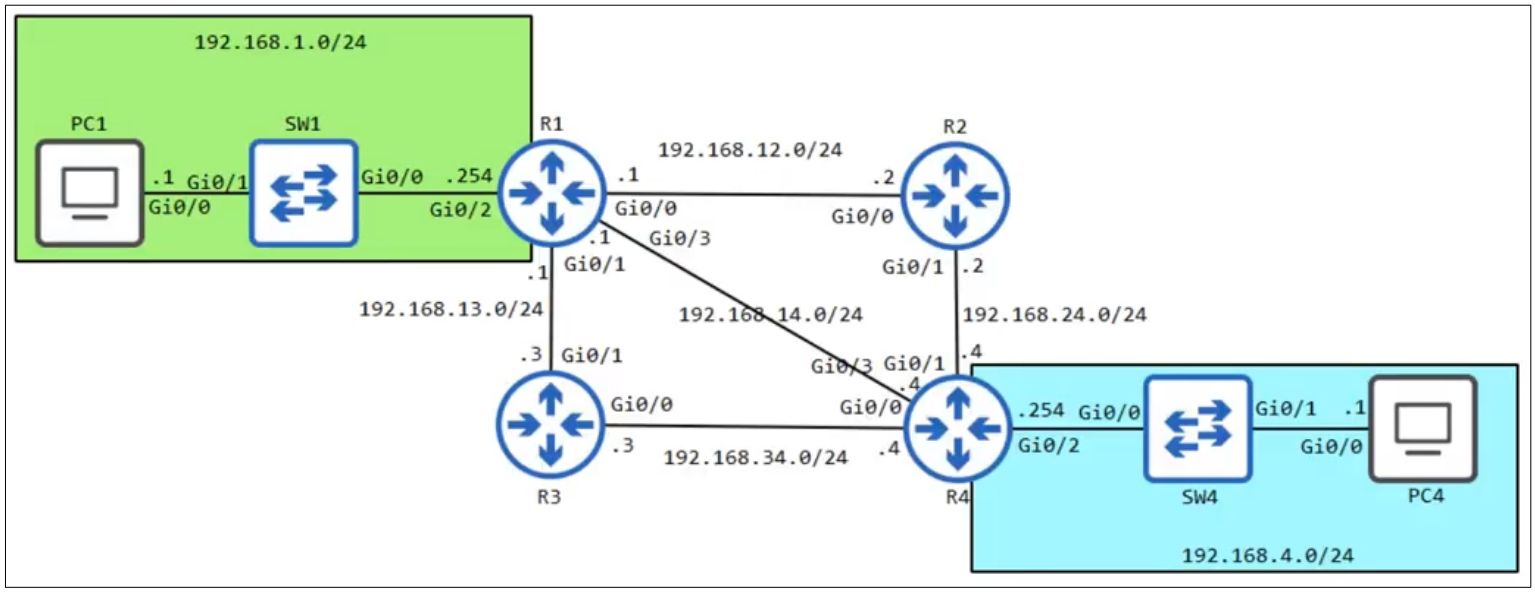
→ We should have ***two-way rachability*** now!

Okay, now we have configured static routes in both directions, you can see that the ping succeeds!

```
PC1#ping 192.168.4.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.4.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 26/31/41 ms
PC1#
```

→ Most Specific Matching Route:
- When a Router looks up a destination address in its routing table, it looks for the ***most specific matching*** route.
- Most Specific = Longest prefix length (**/32** > **/24** > **/16** > **/8** > **/0**).

→ Here We've added one more link between **R1** and **R4** to our Topology:



It's the **192.168.14.0/24** network, with **R1**'s address being **192.168.14.1** and **R4**'s **192.168.14.4**.
We also configured some extra routes, lets check the route table:

On the **R1**'s routing table, let's say we issue the command '**`ping 192.168.4.1`**' on **R1**.
How many routes actually match **192.168.4.1**?

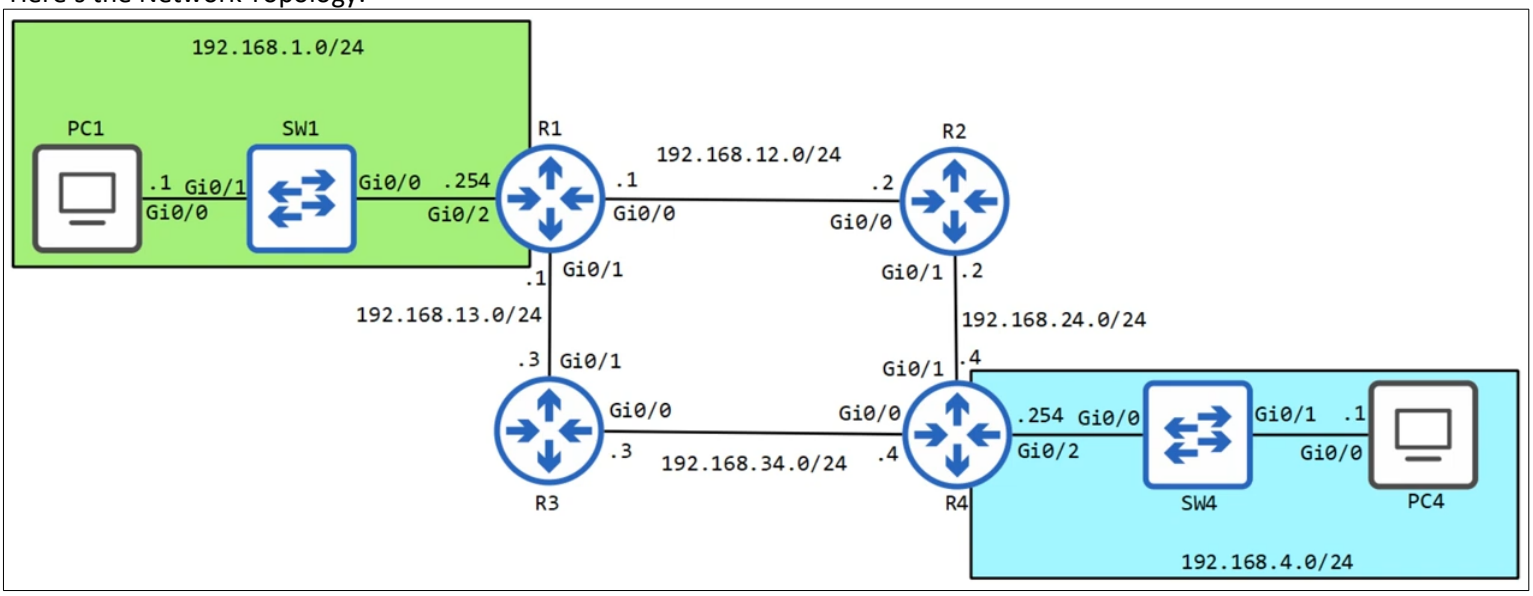
```
S       192.0.0.0/8 [1/0] via 192.168.13.3
        192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C          192.168.1.0/24 is directly connected, GigabitEthernet0/2
L          192.168.1.254/32 is directly connected, GigabitEthernet0/2
        192.168.4.0/24 is variably subnetted, 2 subnets, 2 masks
S          192.168.4.0/24 is directly connected, GigabitEthernet0/0
S          192.168.4.1/32 [1/0] via 192.168.14.4
        192.168.12.0/24 is variably subnetted, 2 subnets, 2 masks
C          192.168.12.0/24 is directly connected, GigabitEthernet0/0
L          192.168.12.1/32 is directly connected, GigabitEthernet0/0
        192.168.13.0/24 is variably subnetted, 2 subnets, 2 masks
C          192.168.13.0/24 is directly connected, GigabitEthernet0/1
L          192.168.13.1/32 is directly connected, GigabitEthernet0/1
        192.168.14.0/24 is variably subnetted, 2 subnets, 2 masks
C          192.168.14.0/24 is directly connected, GigabitEthernet0/3
L          192.168.14.1/32 is directly connected, GigabitEthernet0/3
```

We have three routes that matches: **192.0.0.0/8**, **192.168.4.0/24** and **192.168.4.1/32**
and the Most Specific Matching Route is **192.168.4.1/32**.
- We haven't seen a Class C address like **192.0.0.0** with a **/8** mask, but it follows the same logic.
  /8 means that only the first octet is covered by the mask. The other three octets can vary, and they will still match.
  So, **192.168.4.1** does match **192.0.0.0/8**.
- However, As we said, **192.168.4.1** is the most specific route, since it has the longest prefix length, **/32**.
  So, in this case, the Router will choose that route.

Day 12: Life of a Packet:-

→ The Process of Sending a Packet to Remote Networks:-

Here's the Network Topology:

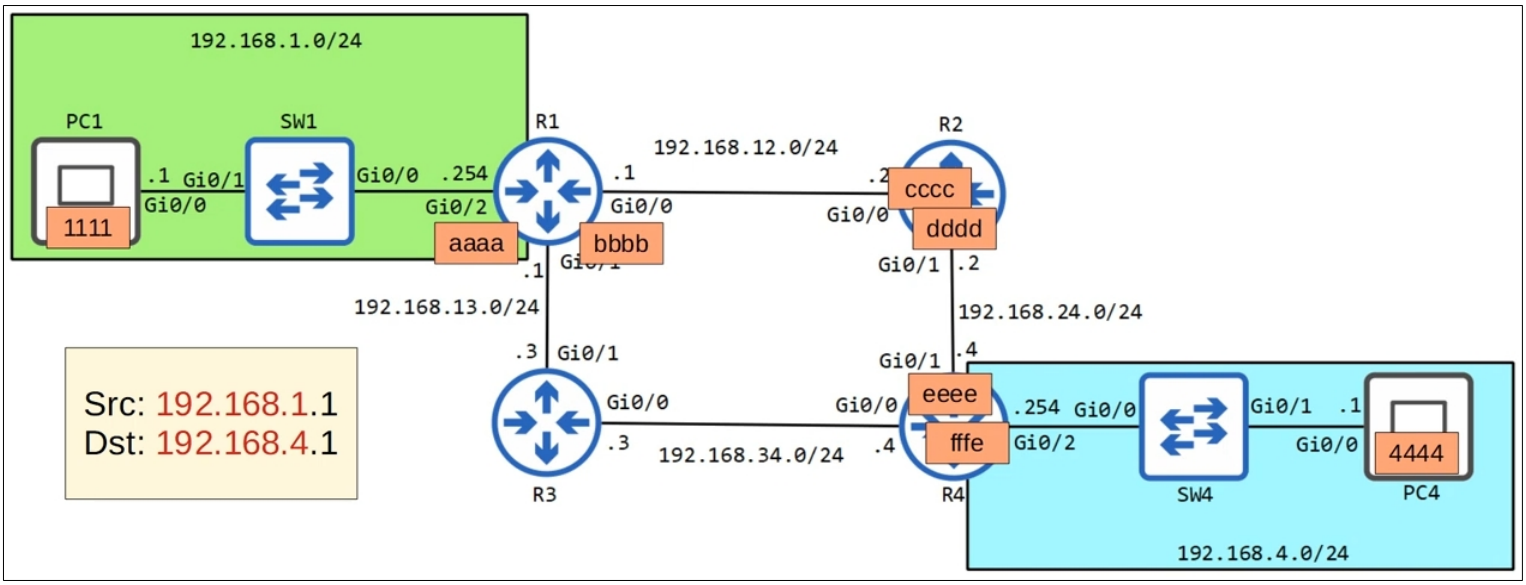


The same Topology we've used to demonstrate Static Routing in Day11 video.

We'll follow a Packet being sent from **PC1** in the **192.168.1.0/24** network, to **PC4** in the **192.168.4.0/24** network.

→ Let's assume we have pre-configured Static Routes on these devices, so that the packet will follow the same path as in the Static Routing video, that is from **PC1** to **R1**, **R2**, **R4**, and then **PC4**.
This doesn't have to be the path the packet takes, the path that goes via **R3** instead of **R2** is valid too, but we'll stick to the last one.

→ Since we're not just looking at Layer3 in this video, let us add MAC addresses for these devices.
As you know MAC address is **12** hexadecimal characters, but just to save space we'll shorten them to **4**:



→ Each interface on a Network device has a ***unique*** MAC address ←

It is not necessary to know the MAC address of the Switches just to avoid clutter.
We didn't make the MAC address of the **G0/2** interface of **R4** with all **FFFF**'s,
because the **FFFF.FFFF.FFFF**, **12 F**s, is the ***Broadcast MAC Address***.
So, just to avoid confusion we added **E** on the end.

Ok, so **PC1** wants to send some data to **PC4**, and it is encapsulated in that IP header. The Source is **192.168.1.1**, **PC1**'s IP address, and the destination is **192.168.4.1**, **PC4**'s IP address.
Now, because **PC1**'s IP address is in the **192.168.1.0/24** network, it notices that the address **192.168.4.1** is in a different network.
→ So it knows that it need to send the packet to its *default gateway*, which is **R1**, something we have already pre-configured.
However, in this example, **PC1** has not sent any traffic yet, so it needs to us **ARP**, Address Resolution Protocol.

→ ARP Process:-

→ So, **PC1** makes this ARP request packet:

ARP Request
Src IP: 192.168.1.1
Dst IP: 192.168.1.254
Dst MAC: ffff.ffff.ffff
Src MAC: 1111

The Source IP address is its own IP address and then The Destination is **R1**'s **G0/2** interface, which is the *default gateway* configured on **PC1**.
The Destination MAC address is the Broadcast MAC address of all **F**s, because it doesn't know the MAC address of **R1**, so it will send the frame to all hosts on the network. Finally, the Source MAC address is its own MAC address.

→ So, it sends the frame, which **SW1** receives and broadcasts out of all its interfaces except the one it received the frame on.
This means that **SW1** will forward the frame out of it's **C0/0** interface.
To translate the meaning of this frame into English, **PC1** is saying **'***Hi 192.168.1.254, What's your MAC address?***'**
NOTE: SW1 learns **PC1**'s MAC address on its **G0/1** interface when the frame arrives on its **G0/1** interface.

→ When this broadcast frame arrives on **R1**, it notices that the destination IP is its own IP, so it creates this ARP reply frame to send back to **PC1**.

ARP Reply
Src IP: 192.168.1.254
Dst IP: 192.168.1.1
Dst MAC: 1111
Src MAC: aaaa

Although, the ARP request message was broadcast, because **R1** learned **PC1**'s IP and MAC addresses from the ARP request message, the ARP reply can be sent unicast directly to **PC1**.
That's what **R1** does.

To translate this ARP reply message into English, basically it means **'***Hi 192.168.1.1, This is 192.168.1.254, My MAC address is AAAA***'**.
NOTE: SW1 will learn **R1**'s MAC address from this message, when the frame arrives on its **G0/0** interface.

→ So, now **PC1** knows the MAC address of its *default gateway*, so it encapsulates the packet with this Ethernet header.
Keep in mind, the original packet is not changed, the destination address remains **PC4**'s IP address, NOT **R1**'s IP address.
Only at Layer2 is the destination set to **R1**'s MAC address.
So, it sends the frame to **R1**.

Src: 192.168.1.1 Dst: 192.168.4.1	Dst: aaaa Src: 1111

→ **R1** receives it, and removes the Ethernet header.
It looks up the destination in its routing table. The most specific match is this entry for the **192.168.4.0/24** network, which specifies a next hop of **192.168.12.2**.
So, **R1** will have to encapsulate this packet with an Ethernet frame with the appropriate MAC address for **192.168.12.2**.
However, **R1** doesn't know **R2**'s MAC address yet.

R1 Routing Table

Destination	Next Hop
192.168.4.0/24	192.168.12.2
...	

→ So, **R1** will send ARP request
The source IP address of this ARP request will be **R1**'s, **192.168.12.1**, and the destination will be **R2**'s, **192.168.12.2**.
The destination MAC address is all **F**s, the broadcast MAC address, because **R1** doesn't know **R2**'s MAC address, and the source is **bbbb**, which is the MAC address of **R1**'s **G0/0** interface.
So, it sends the ARP request, and **R2** receives it

ARP Request
Src IP: 192.168.12.1
Dst IP: 192.168.12.2
Dst MAC: ffff.ffff.ffff
Src MAC: bbbb

ARP Reply
Src IP: 192.168.12.2
Dst IP: 192.168.12.1
Dst MAC: bbbb
Src MAC: cccc

→ **R2** receives the broadcast, and since the destination IP address matches its own IP address, it sends this ARP reply to **R1**.
Because it learned the IP and MAC address of **R1** from the ARP request, It doesn't have to broadcast the frame.
So, **R2** sends this ARP reply back.
Now **R1** knows **R2**'s MAC address, so it can encapsulate the packet with an Ethernet header, inserting **R2**'s MAC address in the destination field, and the MAC address of **R1**'s **G0/0** interface in the source field, and it sends it to **R2**.
After receiving the frame **R2** removes the Ethernet header.
R2 then looks up the destination IP address in its routing table, and the most specific match is this one for **192.168.4.0/24**, with next hop of **192.168.24.4**
Although **192.168.24.0/24** is a connected network to **R2**,
it doesn't know the MAC address of **R4**.

R2 Routing Table

Destination	Next Hop
192.168.4.0/24	192.168.24.4
...	

→ **R2** will now use ARP request to discover **R4**'s MAC address.
R2 prepares this ARP request frame, using its own IP and MAC addresses for the source, **R4**'s IP address as the destination, and the broadcast MAC address, and it forwards it out of its **G0/1** interface.
With this ARP request, **R2** is saying **'***Hi 192.168.24.4, What's your MAC address?***'**

ARP Reply
Src IP: 192.168.24.4
Dst IP: 192.168.24.2
Dst MAC: dddd
Src MAC: eeee

→ **R4** receives the broadcast, and since the destination IP address matches its own IP address, it creates this ARP reply frame to send back to **R2**, once again it already knows **R2**'s IP and MAC addresses because they were used as the source addresses for the ARP request.
It sends the *unicast* frame back to **R2**.
With this reply, **R4** is saying **'***Hi 192.168.24.2, This is 192.168.24.4, My MAC address is EEEE***'**

ARP Reply
Src IP: 192.168.24.4
Dst IP: 192.168.24.2
Dst MAC: dddd
Src MAC: eeee

→ Now that **R2** knows **R4**'s MAC address, it encapsulates **PC1**'s packet with an Ethernet header, with a destination MAC address of *EEEE*, which is **R4**'s **G0/1** interface, and a source MAC address of *DDDD*, which is **R2**'s **G0/1** interface.

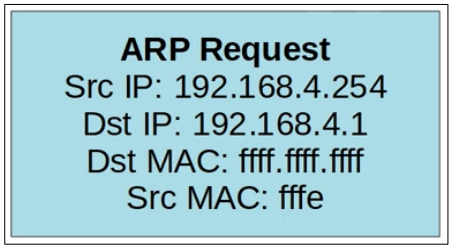
Src: 192.168.1.1 Dst: 192.168.4.1	Dst: eeee Src: dddd

→ **R4** receives the frame and removes the Ethernet header.
It looks up **192.168.4.1** in its routing table, and the most specific match is this entry for **192.168.4.0/24**, which is directly connected via the **G0/2** interface.
But, once again, **R4** doesn't know **PC4**'s MAC address yet.
It will use ARP request to learn **PC4**'s MAC address.
It prepares this ARP request frame, again, the source IP and MAC addresses are its own, the destination IP address is **PC4**'s, and the destination MAC address is the broadcast MAC address, all **F**s.
It sends this message out of its **G0/2** interface

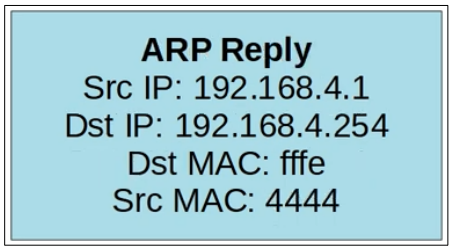
R4 Routing Table

Destination	Next Hop
192.168.4.0/24	directly connected, Gi0/2
...	

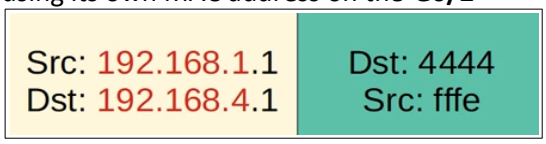
NOTE: SW4 will learn **R4**'s MAC address on its **G0/0** interface from the source MAC address field of his Ethernet frame.



→ After **PC4** receives the frame it checks the destination IP address, since it is its own IP address, it will send and ARP reply.
The ARP reply will be *unicast*, using **PC4**'s IP and MAC addresses for the source and **R4**'s IP and MAC addresses for the destination.
It sends the frame out of its **G0/0** network interface.
NOTE: **SW4** learns **PC4**'s MAC address when it arrives on its **G0/1** interface.



→ Now that **R4** knows **PC4**'s MAC address, it adds an Ethernet header to the packet, using its own MAC address on the **G0/2** interface as the source address, and **PC4**'s MAC address as the destination.
R4 sends the frame to **PC4**, and finally it has reached its destination :)



NOTE: The original Packet hasn't changed throughout the process.
It is always used the same IP header with a source IP address of **192.168.1.1** and a destination IP address of **192.168.4.1**.

NOTE: Switches didn't actually modify the frames at any point. The Switches forwarded the frames and learned the MAC addresses, but they don't actually de-encapsulate and then re-encapsulate the packet with a new Ethernet header.

→ **Minor Point:** In the ARP request , we put the Source IP before the Destination IP, but the Destination MAC before the Source MAC. That's because, in the IPv4 Header the source IP address comes first, but in the Ethernet header the Destination MAC Address comes first.

→ Okay, Now let's say **PC4** sends a reply back to **PC1**, and we've configured static routes on the Routers so that the traffic follows the same path on the way back to **PC1**, going via **SW4**, **R4**, **R2**, **R1**, **SW1**, and then reaching **PC1**, what will be different??
First off, there will be one major difference:
Since these devices have already gone through the ARP process, *there won't be any need for ARP requests and replies.*
The Packet will simply be forwarded from device to device, being de-encapsulated aand then re-encapsulated as it is received by and then forwarded by each Router.

→ Some commands from the Lab video:

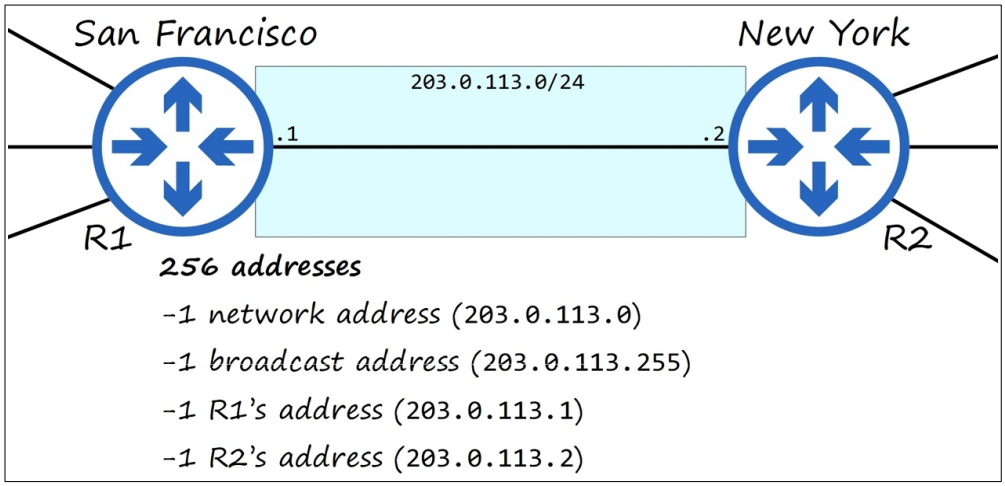
- `ipconfig /all` to know the MAC address and other info on the PC
- `show interfaces [g0/0]` to show the detailed information of the Network interface on a Router
-

→ The **BIA**, Burned-In Address, is the ***actual*** MAC address assigned to the interface by the device Maker, when it was made. You can actually configure a different MAC address in the CLI, and it will use that MAC address!

Day 13: Subnetting pt.1:-

→ IPv4 Address Classes:-

→ How does a company get their own network to use? Well, IP addresses are assigned to companies or organizations by a non-profit American corporation called the **IANA**, the **Internet Assigned Numbers Authority**.
The **IANA** assigns **IPv4** addresses and networks to companies based on their size.
For example, a very large company might receive a **Class A** or **Class B** network,
while a small company might receive a **Class C** network.
This leads to many WASTED IP addresses!
So, multiple methods of improving this system have been created.
An example of how this strict system can waste IP addresses:-



→ We have 2 Routers connected together, **R1** and **R2**. Each Router has *3 separate Layer3 Networks, with different IP networks*.
→ Each of these networks will have a few Switches, with many end-hosts such as PCs and Servers connected to these Switches.
→ There is one more network, The network connecting the two Routers, **R1** and **R2**. It is known as ***point-to-point*** network.
→ Meaning its a network connecting two points.
→ This might be a connection between Offices in different cities, Let's Say San Francisco and New York.
→ As it is a *point-to-point* network, we don't need a large address block, so we can use a Class C network, **203.0.113.0/24**.
→ You will note here that there are only **4** IP addresses in use, and **252** IP addresses are WASTED!
→ This is not an IDEAL system!

Another example:-
→ Company X needs IP addressing for **5000** end hosts. This is a bi problem. Why!?
Because a Class C network doe not provide enough addresses, so a Class B network must be assigned!
Because a Class B network will allows for about **65,000** addresses, this results in about **60,000** addresses being WASTED!

→ CIDR (Classless Inter-Domain Routing):-

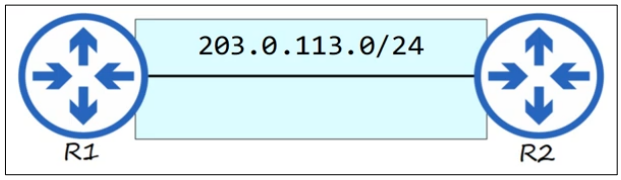
We have introduced **IPv4** adress classes, such as Class A, B and C.
Well, **CIDR** throws all that away and lets us be more flexible with our IPv4 networks.

→ When the Internet was firs being created, the creators did not predict that the Internet would become as large as it is today.
This results in wasted address space like the examples we showed * (And there are many more examples!).

→ The total **IPv4** *address space* includes over **4B billion** of addresses, and that seemed like a huge number of addresses when **IPv4** was created, but now address space exhaustion is a big problem, there's not enough addresses.
→ One way to solve, or remedy this problem is **CIDR**. The **IETF**, (**Internet Engineering Task Force**) introduced **CIDR** in **1993** to replace the '*classful*' addressing system.

→ With **CIDR**, the requirements of (Class **A** addresses must use a **/8** network mask, Class **B /16** mask and Class **C /24** mask) were removed! This allowed larger networks to be split into smaller networks, allowing greater efficiency.
These smaller networks are called '***Subnetworks***' or '***subnets***'.

→ Let's splitt a larger network into a smaller network:



→ Here's the same point-to-point network we looked at before.
Previously, it was assigned the **203.0.113.0/24** network space, but that resulted in lots of wasted addresses!
→ Let's write this out in binary:-
OK, so here's the binary with the dotted decimal underneath:

```
1 1 0 0 1 0 1 1 . 0 0 0 0 0 0 0 0 . 0 1 1 1 0 0 0 1 . 0 0 0 0 0 0 0 0
  203     .     0     .     113     .     0
```

→ The prefix length is **/24**, so here's the network mask, aka, the subnet mask, **255.255.255.0**:

```
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 0 0 0 0 0 0 0 0
  255     .     255     .     255     .     0
```

REMEMBER: All **1**'s in the subnet mask indicate that the same bit in the address is the network portion
In this case, I made the network portion blue, and the host portion is red.

→ Well, how many host bits are there? **8**, because it's one octet! So, how many potential hosts, or how many usable addresses are there?
Well, the formula is this:

$$2^8 - 2 = 254 \text{ usable addresses}$$

8, the power of **2**, is the number of host bits.
And minus two (**-2**), the network address and the broadcast address.
So, we have **254** usable address, but we only need two, one for **R1** and one for **R2**.
However, **CIDR** allows us to assign different prefix length, it doesn't have to be **/24**.

→ CIDR practice:-
→ Let's get some practice calculating the number of hosts with different prefix length.
203.0.113.0/25. **203.0.113.0/26**. **203.0.113.0/27**, **/28**, **/29**, **/30**, **/31** and finally **/32**.
We put **/31** and **/32** in highlight because they're a little bit special, you'll see when you try to calculate it.
Here the formula to get the usable addresses in the network:

$$2^n - 2 = \text{usable addresses}$$

n = number of host bits

→ So, the solution will be **126**, **62**, **30**, **14**, **6**, **2**, **0**, **0**

→ Video Solution: Here is **203.0.113.0**, but this time with a **/25** mask.
Notice that the network portion of the address has extended into the first bit of the last octet, and the mask in dotted decimal is now written as **255.255.255.128**. We changed the color of the extra bit to purple, but it's part of the network portion, blue part.
→ Now, there are **7** bits in the host portion of the address, so the number of usable addresses is **2** to the power of **7**, minus **2**, which equals **126**. We only need **2** addresses, one for **R1** and one for **R2**, so we'll be wasting **124** addresses!
That's better than wasting **252** addresses with a **/24** prefix length, but still its wasteful.
And so on to the other prefix lengths. And the solution as we did it!
→ If we use a **/30** prefix length, the mask is written as **255.255.255.252** in dotted decimal. There are now only **2** host bits.
That means **2** usable addresses. So, this is perfect! That means **0** wasted addresses!

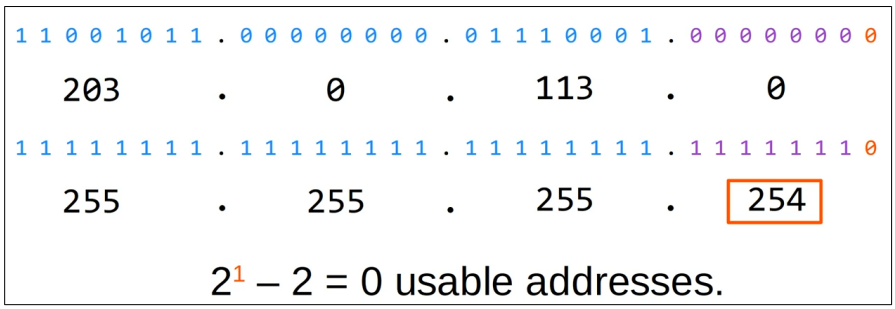
→ Before moving on to **/31** and **/32** let me clarify a little bit:
So, instead of **203.0.113.0/24**, we will use **203.0.113.0/30**, which is a *subnet* of that larger Class **C** network.
203.0.113.0/30 includes the address range of **203.0.113.0** through **203.0.113.3**.
This is how it is converted in binary: These are the **4** addresses in the network which we took up with that subnet.

```
203.0.113.0/30
= 203.0.113.0 - 203.0.113.3
1 1 0 0 1 0 1 1 . 0 0 0 0 0 0 0 0 . 0 1 1 1 0 0 0 1 . 0 0 0 0 0 0 0 0
1 1 0 0 1 0 1 1 . 0 0 0 0 0 0 0 0 . 0 1 1 1 0 0 0 1 . 0 0 0 0 0 0 0 1
1 1 0 0 1 0 1 1 . 0 0 0 0 0 0 0 0 . 0 1 1 1 0 0 0 1 . 0 0 0 0 0 0 1 0
1 1 0 0 1 0 1 1 . 0 0 0 0 0 0 0 0 . 0 1 1 1 0 0 0 1 . 0 0 0 0 0 0 1 1
```

→ So, What about other addresses in the **203.0.113.0/24** range?
The remaining address in the **203.0.113.0/24** address block (**203.0.113.4** – **203.0.113.255**),
Are now available to be used in other *subnets*! That's the magic of subnetting!

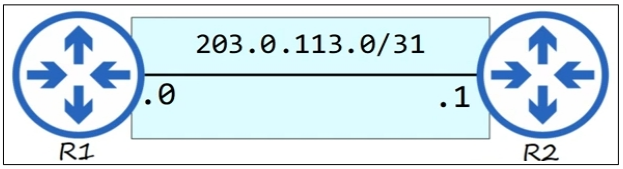
→ Instead of using **203.0.113.0/24** and wasting **252** addresses, we can use **/30** and waste no addresses.
Or, perhaps there is another way to make this more efficient? Let's look into it:

→ CIDR (/31):-
→ If we use a **/31** prefix length, the mask is written as **255.255.255.254** in dotted decimal.
There is now only **1** host bit, so that means '**0 usable addresses**'.
2 to the power of **1** is **2**, minus **2** for the network and broadcast addresses, which equals **0** usable addresses that we can assign.
→ So, you used to NOT be able to use **/31** network prefixes because of this.



HOWEVER, for a point-to-point connection like this it actually is possible to use a **/31** mask!

→ Back to the network, **203.0.113.0/31** network, **R1** is **203.0.113.0**, and **R2** is **203.0.113.1**.
The **203.0.113.0/31** network consists of the addresses from **203.0.113.0** through **203.0.113.1**, which is actually only TWO addresses.



→ Normally, this would be a problem, because it leaves no usable addresses after subtracting the network and broadcast addresses!
→ But for *point-to-point* networks like this, a dedicated connection like this between two Routers, there is actually no need for a network address or a broadcast address.
→ So, we can break the rules in this case and assign the only two addresses in this network to our Routers.

NOTE: If you try this configuration on a Cisco Router, you'll get a warning like this, reminding you to make sure that this is a *point-to-point* link, but it is a totally valid configuration.

```
Router(config-if)#ip address 203.0.113.0 255.255.255.254
% Warning: use /31 mask on non point-to-point interface cautiously
Router(config-if)#
```

→ Once again, The reminding addresses in the **203.0.113.0/24** address block, which are (**203.0.113.2** – **203.0.113.255**) are now available to be used in other networks.
But this time we saved even more addresses, using only **2** addresses instead of **4** for this *point-to-point* connection.
→ People still use **/30** for *point-to-point* connections at times, but **/31** masks are totally valid and more efficient than **/30**.
→ **/31** mask is RECOMMENDED!

→ CIDR (/32):-
→ **/32** mask is written as**255.255.255.255** in dotted decimal, making the entire address the network potion, there are no host bits.
→ If you calculate this using our formula, you will get **-1** usable addresses.
Clearly the formula doesn't work in this case.
You won't be able to use a **/32** mask in the case of *point-to-point* connection, and you will probably never use a **/32** mask to configure an actual interface.

HOWEVER, there are some uses for **/32** mask, for example: When you want to crate a Static Route, not to a network, but just specific host, you can use a **/32** mask to specify that exact host.

CIDR Notation: is the way of writing a prefix with a slash followed by the prefix length, like **/25**, **/26**, etc.
Because it was introduced with the **CIDR** system.
Previously, only the dotted decimal method was used.

NOTE: We have only learned how to *subnet* a Class **C** network, but we will look at Class **A** and Class **B** networks as well!

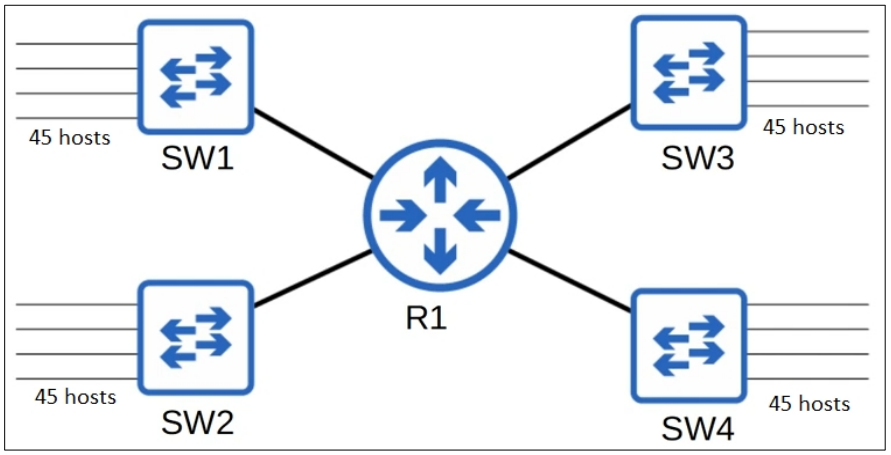
Here is a chart showing the dotted decimal subnet masks, and their equivalent in **CIDR** Notation:

Dotted Decimal	CIDR Notation
255.255.255.128	/25
255.255.255.192	/26
255.255.255.224	/27
255.255.255.240	/28
255.255.255.248	/29
255.255.255.252	/30
255.255.255.254	/31
255.255.255.255	/32

→ The process of Subnetting:-

→ ***Subnetting***: dividing a larger network into smaller networks, called ***subnets***.

Another one example of subnetting:



There are **4** networks connected to **R1**, with many hosts connected to each Switch.

There are **45** hosts per network, **R1** needs an IP address in each network so its address is included in that **45** host number.

You have received the **192.168.1.0/24** network,
And you must divide the network into four subnets that can accommodate the number of hosts required.

→ First off, are there enough addresses in the **192.168.1.0/24** network in he first place?
So, we need **45** hosts per network, including **R1**, but also remember that each network has a network and broadcast address, so that's plus **2**, so we need **47** addresses per subnet. **47** times **4** equals **188**, so there's no problem in terms of the number of hosts.
192.168.1.0/24 is a Class **C** network, so there are **256** addresses, so we will be able to assign **4** subnets to accommodate all hosts, no problem.

→ Let's see how we can calculate the subnets we need to make. We need **4** equal sized subnetw with enough room for at least **45** hosts.
Here, We've written out **192.168.1.0** with a **/30** mask, **255.255.255.252**.
I skipped **/32** and **/31**, since these aren't *point-to-point* links, we can't use **/31**, and definitely can't use **/32**.
Since there are **2** hosts bits, the formula to determine the number of usable addresses is **2** to the power of **2**, minus **2**.
So that means there are **2** usable addresses in a **/30** network. Clearly not enough room to accommodate the **45** hosts we have.

```
1 1 0 0 0.0.0.0 . 1 0 1 0 1 0 0 0 . 0 0 0 0 0 0 0 1 . 0 0 0 0 0 0 0 0
   192      .     168     .     1     .     0

1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 0 0
   255      .     255     .     255     .     252
```

→ Ho about if we use a **/29** mask to make these subnets, can we fit the **45** hosts we need? There are three bits, so the formula is **2** to the power of **3**, minus **2**. Therefore there are **6** usable addresses, not enough for **45** hosts.

```
1 1 0 0 0.0.0.0 . 1 0 1 0 1 0 0 0 . 0 0 0 0 0 0 0 1 . 0 0 0 0 0 0 0 0
   192      .     168     .     1     .     0

1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 0 0 0
   255      .     255     .     255     .     248
```

→ How about if we use **/28**? There are **4** host bits, so the formula is **2** to the power of **4**, minus **2**. So that means there are **14** usable addresses. And also not enough.

```
1 1 0 0 0.0.0.0 . 1 0 1 0 1 0 0 0 . 0 0 0 0 0 0 0 1 . 0 0 0 0 0 0 0 0
   192      .     168     .     1     .     0

1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 0 0 0 0
   255      .     255     .     255     .     240
```

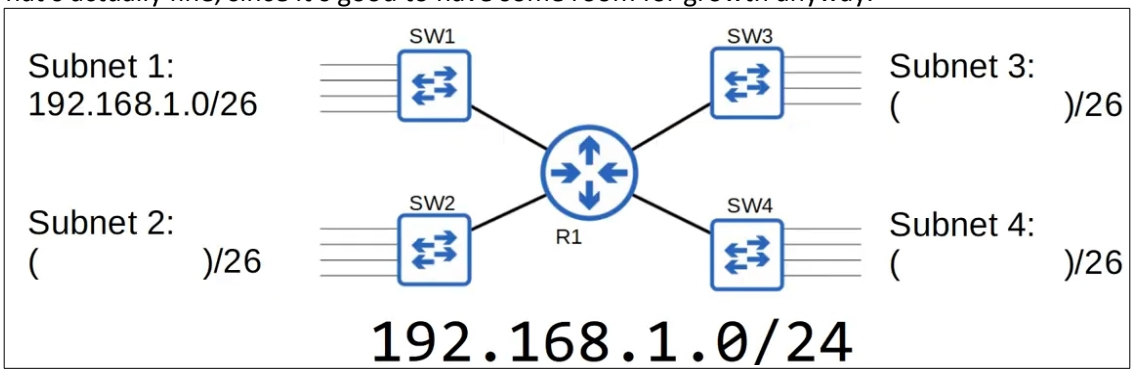
→ How about **/27**? There are **5** host bits, so the formula is **2** to the power of **5**, minus **2**. Equals **30** usable addresses. Not enough.

```
1 1 0 0 0.0.0.0 . 1 0 1 0 1 0 0 0 . 0 0 0 0 0 0 0 1 . 0 0 0 0 0 0 0 0
   192          .     168        .       1        .       0

1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 0 0 0 0 0
   255          .     255        .       255      .     224
```

→ How about **/26** mask? There are now **6** host bits, so the formula is **2** to the power of **6**, minus **2**.
That means there are **62** usable address! And finally we found a number!

```
1 1 0 0 0.0.0.0 . 1 0 1 0 1 0 0 0 . 0 0 0 0 0 0 0 1 . 0 0 0 0 0 0 0 0
   192          .     168        .       1        .       0

1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 0 0 0 0 0 0
   255          .     255        .       255      .     192
```

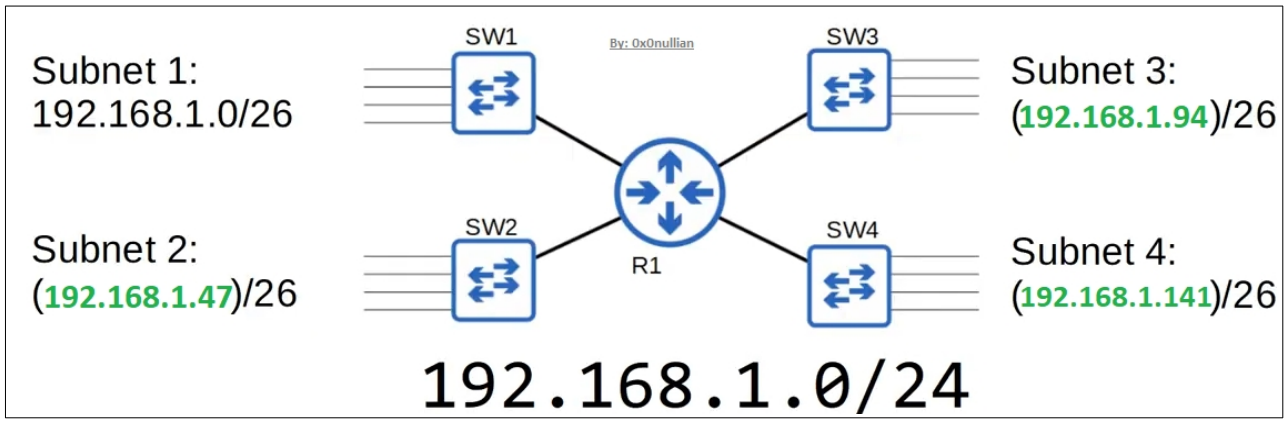
→ **/27** doesn't provide enough address space. **/26** provides more than we need, but we have to go0 with **/26**
Unfortunately we can't always make subnets have exactly the number of addresses we want. There might be some unused address space. That's actually fine, since it's good to have some room for growth anyway.



→ The first subnet (*Subnet1*) is **192.168.1.0/26**. What are the remaining subnets???
This will be a QUIZ to solve!
→ Hint: Find the broadcast address of *Subnet1*. The next address is the network address of *Subnet2*. Repeat the process for *Subnet3* and *Subnet4*.

The QUIZ answer:
→ As the *Subnet1* have **45** hosts, excluding the network address and the broadcast address, the network address is **192.168.1.0/26**, and the first IP address is **192.168.1.1/26** so the broadcast address will be after the last IP address.
The broadcast address is **192.168.1.46**.

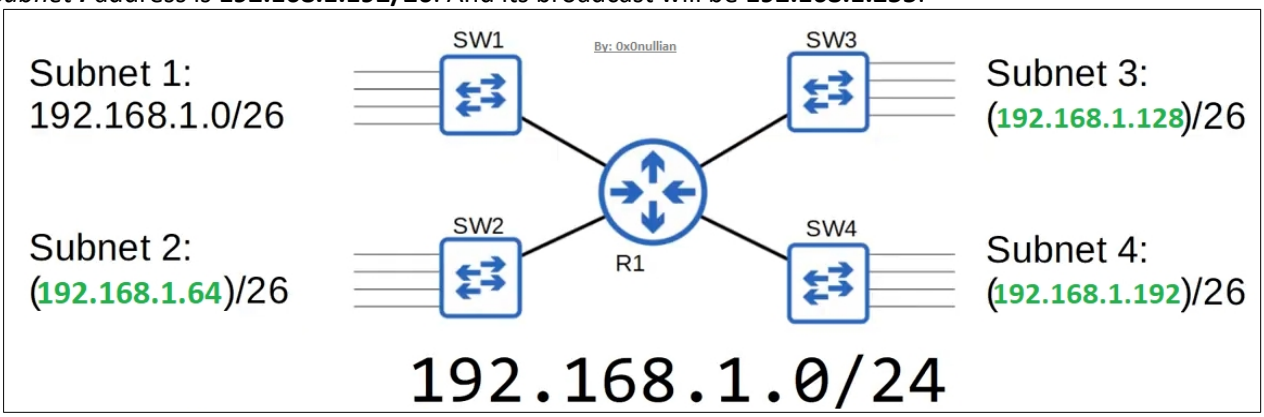
→ So the *Subnet2* address is **192.168.1.47/26**. And its broadcast will be **192.168.1.93**.
→ So the *Subnet3* address is **192.168.1.94/26**. And its broadcast will be **192.168.1.140**.
→ So the *Subnet**4*** address is **192.168.1.141/26**. And its broadcast will be **192.168.1.187**.



My Quiz answer was ***WRONG!***
As I solve it as there are only **47** hosts only, not depending on the prefix length which is **/26** which will be there **64** hosts!

The right ***ANSWER:***
→ As the *Subnet1* have **45** hosts, excluding the network address and the broadcast address, the network address is **192.168.1.0/26**, and the first IP address is **192.168.1.1/26** so the broadcast address will be after the last IP address. The broadcast address is **192.168.1.63**.

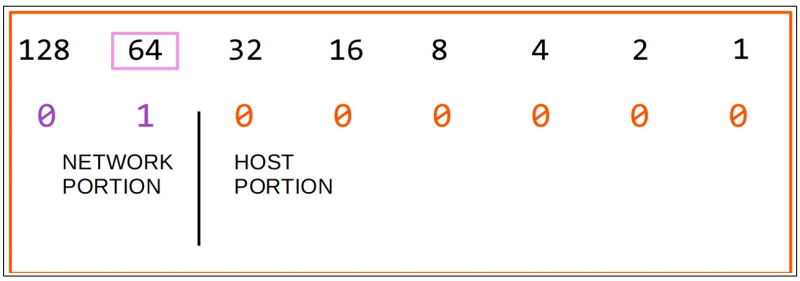
→ So the *Subnet2* address is **192.168.1.64/26**. And its broadcast will be **192.168.1.127**.
→ So the *Subnet3* address is **192.168.1.128/26**. And its broadcast will be **192.168.1.191**.
→ So the *Subnet**4*** address is **192.168.1.192/26**. And its broadcast will be **192.168.1.255**.

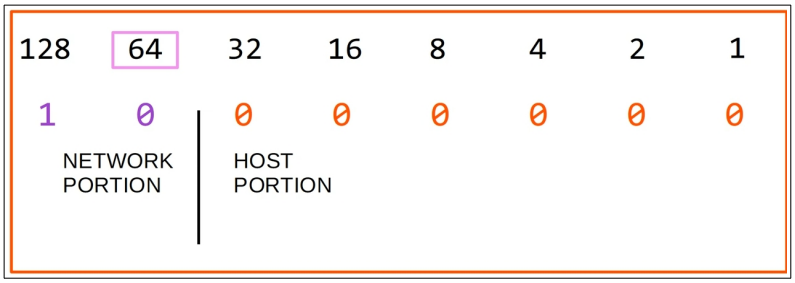


Day 14: Subnetting pt.2:-

→ Subnetting Practice (Class C):-

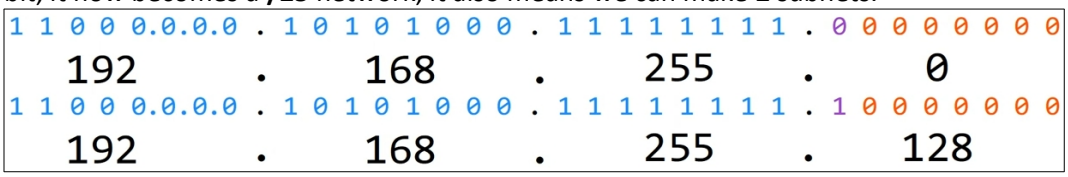
→ **Subnetting Trick:-** On the Last Video QUIZ:
The last bit of the network portion is **64**, this means that to find the next subnet we just have to add **64**.
Add **64** to the first subnet **192.168.1.0/26**, and then we get **192.168.1.64/26**. And then **192.168.1.128/26.** etc.





→ Let's try another similar exercise:
We've been given **192.168.255.0/24** network, and have been asked to divide the network into five subnets of equal size.
In this case, the number of each subnet hasn't been specified, so let's make five subnets that are as large as they can be.
All we have to do is to solve the problem of how many bits we have to *borrow* from the host portion?

- Currently, we don't borrow any bits, so we can't make any subnets, we just still have only one network, **192.168.255.0/24**.
- If we borrow one bit, it now becomes a **/25** network, it also means we can make **2** subnets.

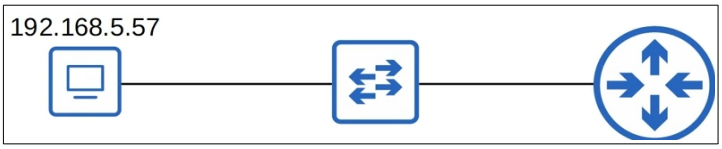


- All of he original network bits, the blue bits, can not be changed. That is the network we received.
- However, the purple bit, the bit we borrowed from the host portion, we can change it! And it can be either **0** or **1**.
- If it's **0**, we have **192.168.255.0/25** network. If it's **1**, we have **192.168.255.128/25** network.
- The formula of the number of subnets is 2^X, and **X** is the number of borrowed bits.

2^x = number of subnets
(x = number of 'borrowed' bits)

- We need **5** subnets, so borrowing one isn't enough.
- If we borrowed **2** bits, **2** to the power of **2** is **4**, and still also isn't enough. The prefix length is **/26**.
- Let's borrow another bit, If we borrow **3** bits, we can make **8** subnets.
- It is more than we need, but if we borrowed **2** bits only and use **/26** mask, we won't have enough.
- So, our first subnet will be **192.168.255.0/27**.
- Using the trick we have learned before, the last bit in the network portion is **32**, So now we should be able to calculate the other subnets, by just adding **32** to the subnet.
- *Subnet2* is **192.168.255.32/27**, *Subnet3* is **192.168.255.64/27**, *Subnet4* is **192.168.255.96/27**.
  and finally *Subnet5* is **192.168.255.128/27**.

→ Another kind of Questions:
Identify the host:
What does host **192.168.5.57/27** belong to?



- So, we have the IP address of the host, but we don't know the network address of the subnet.
- We write the host in binary and dotted decimal, and as it is **/27**, let's show the borrowed bits:

```
1 1 0 0 0.0.0.0 . 1 0 1 0 1 0 0 0 . 0 0 0 0 0 1 0 1 . 0 0 1 1 1 0 0 1
   192      .     168     .     5     .     57
```

- The three purple bits are borrowed and added to the network portion.
- Now to find the network address, we just need to change all of host bits to **0**.
- Now the last octet is, **0010000**. Change it back to dotted decimal and we get **192.168.5.32**

```
1 1 0 0 0.0.0.0 . 1 0 1 0 1 0 0 0 . 0 0 0 0 0 1 0 1 . 0 0 1 0 0 0 0 0
   192      .     168     .     5     .     32
```

- So, the host **192.168.5.57/27**, belongs to the subnet **192.168.5.32/27**.

This is a table of the different subnet sizes for Class **C** networks:

Prefix Length	Number of Subnets	Number of Hosts
/25	2	126
/26	4	62
/27	8	30
/28	16	14
/29	32	6
/30	64	2
/31	128	0 (2)
/32	256	0 (1)

→ For the number of Subnets, each additional bit that we borrow doubles the number of subnets.
→ For the number of Hosts, each host bit doubles the amount of hosts, except you have to subtract **2**, for network and broadcast.
→ Take note of **/31**, the number of hosts is **0**. It's because there is only a single host bit, there are only **2** possible addresses ;)
However, for a *point-to-point* connection, you can actually use a **/31** and assign those two addresses to each end of the connection, and have no network or broadcast addresses.
→ Also **/32** technically uses all bits for the network address, allowing no hosts, but you can assign a **/32** mask to identify a specific host when writing routes and such, and in some other special cases!
REMEMBER: The number of hosts indicates the number of *usable hosts*, excluding the network and broadcast addresses.

→ Subnetting Class B Networks:-

→ Looking back to the chart of Network Classes, We can see that there are many more host bits, and therefore many more possible subnets, that can be created with a Class **B** network than with a Class **C**.
HOWEVER, the process of subnetting is EXACTLY the same!

> The process of subnetting Class A, Class B, and Class C networks is EXACTLY THE SAME!

→ Let's start with an example of Subnetting Class **B** network:-
We've been given the **172.16.0.0/16** network. We're asked to create **80** subnets for our company's various LANs.
What prefix length should we use?

→ We can simply use the **2** to the power of **X** formula, where **X** is the number of '*borrowed*' bits.
If we borrowed any bits, we can't make any subnets.
If we borrow **1** bit, we will get **2** subnets, **2** to the power of **1** equals **2**.
This gives us a prefix length of **/17**, and If we write this sunet mask in dotted decimal it is **255.255.128.0**.

> 2^x = number of subnets
> (x = number of 'borrowed' bits)

```
1 0 1 0 1 1 0 0 . 0 0 0 1 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   172      .     16      .     0      .     0
Subnet Mask:
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   255      .     255     .     128    .     0
```

REMEMBER: When you enter commands in the Cisco CLI, you can't use the **CIDR** Notation like **/17**, you have to enter dotted decimal like **255.255.128.0**.

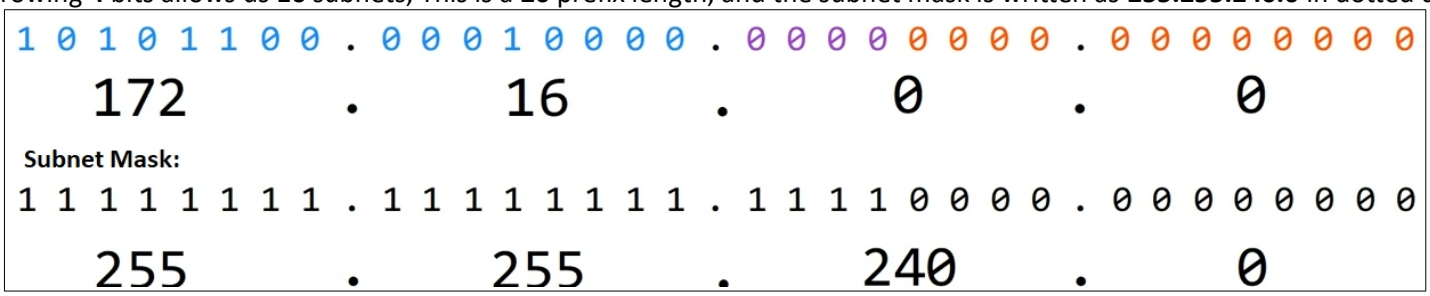
→ **2** subnets isn't enough, Let's borrow another bit:
Borrowing **2** bits allows us **4** subnets. This is a **/18** prefix length, and the subnet mask is written as **255.255.192.0** in dotted decimal.

```
1 0 1 0 1 1 0 0 . 0 0 0 1 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   172      .     16      .     0      .     0
Subnet Mask:
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   255      .     255     .     192    .     0
```

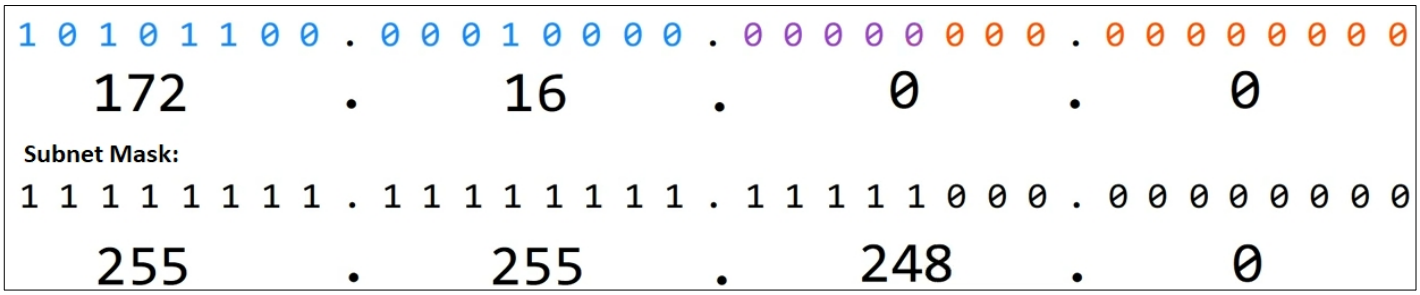
→ Let's borrow another bit:
Borrowing **3** bits allows us **8** subnets, This is a **19** prefix length, and the subnet mask is written as **255.255.224.0** in dotted decimal.

```
1 0 1 0 1 1 0 0 . 0 0 0 1 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   172      .     16      .     0      .     0
Subnet Mask:
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   255      .     255     .     224    .     0
```

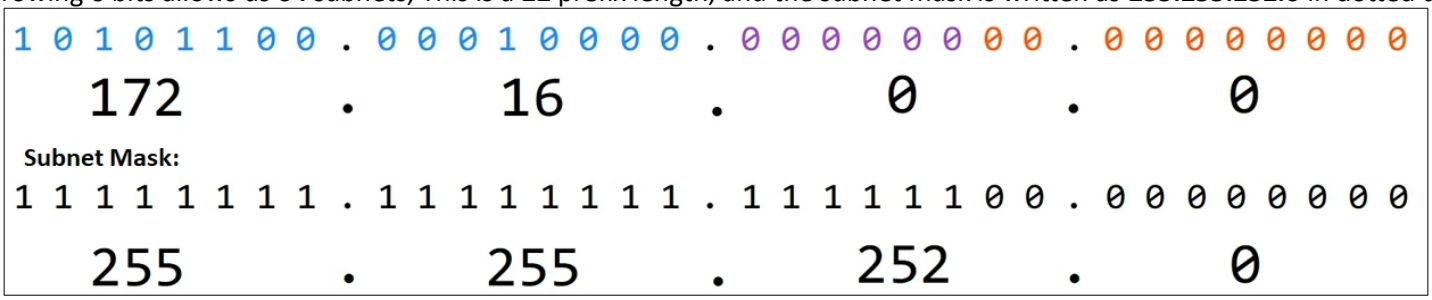
→ Let's borrow another bit:
Borrowing **4** bits allows us **16** subnets, This is a **20** prefix length, and the subnet mask is written as **255.255.240.0** in dotted decimal.



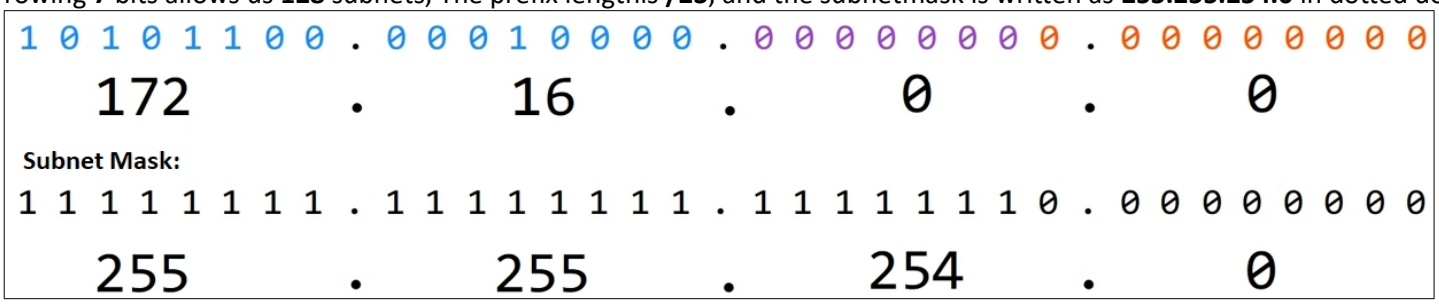
→ Let's borrow another bit:
Borrowing **5** bits allows us **32** subnets, This is a **21** prefix length, and the subnet mask is written as **255.255.248.0** in dotted decimal.



→ Let's borrow another bit:
Borrowing **6** bits allows us **64** subnets, This is a **22** prefix length, and the subnet mask is written as **255.255.252.0** in dotted decimal.



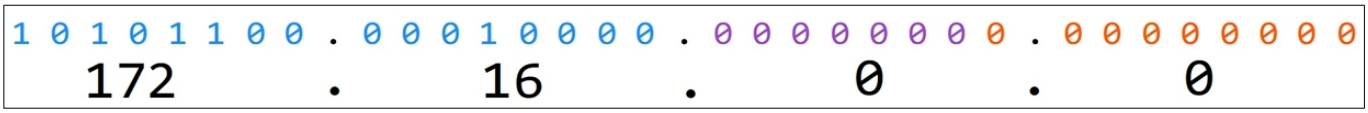
→ Let's borrow one more bit, which should be enough:
Borrowing **7** bits allows us **128** subnets, The prefix lengthis **/23**, and the subnetmask is written as **255.255.254.0** in dotted decimal.



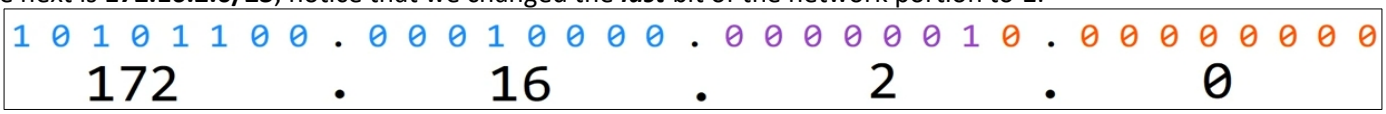
This is the Correct Answer!
We should use **/23** prefix length so we can create the **80** subnets we need.
128 subnets is more than we need, but **/22** only allows for **64**, which is not enough!

→ **Let's look at some of he subnets:**

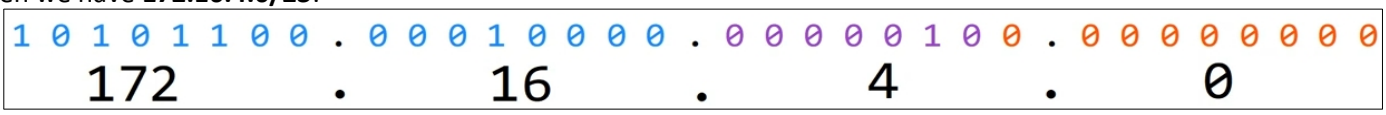
→ The first being **172.16.0.0/23** of course:



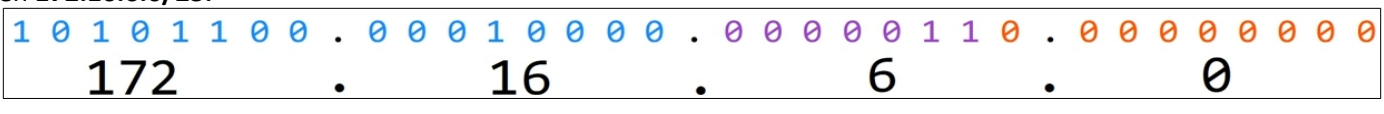
→ The next is **172.16.2.0/23**, notice that we changed the ***last*** bit of the network portion to **1**.



→ Then we have **172.16.4.0/23**:



→ Then **172.16.6.0/23:**

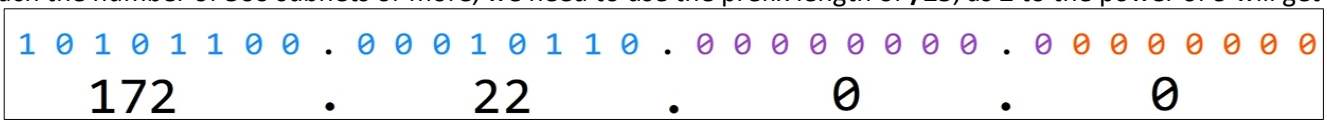


→ Another example:-

We've been given the **172.22.0.0/16** network. We're required to divide the network into **500** separate subnets. What prefix length should we use??

The Answer is:

As the last example, we will borrow a bit every time and calculate the number of subnets we get with the formula **2^x = subnets**. So, to reach the number of **500** subnets or more, we need to use the prefix length of **/25**, as **2** to the power of **9** will get **512**.



NOTE: You can borrow bits even from the last octet, so you can use **/25**, **/26**, **/27**, etc. even with a Class **B** network.

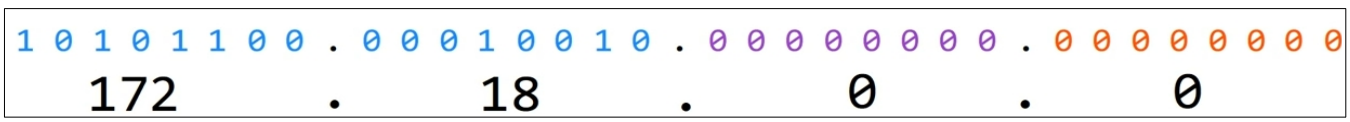
→ Another practice question:-

We've been given the **172.18.0.0/16** network. Our company requires **250** subnets with the same number of *hosts* per subnet. What prefix length should we use??

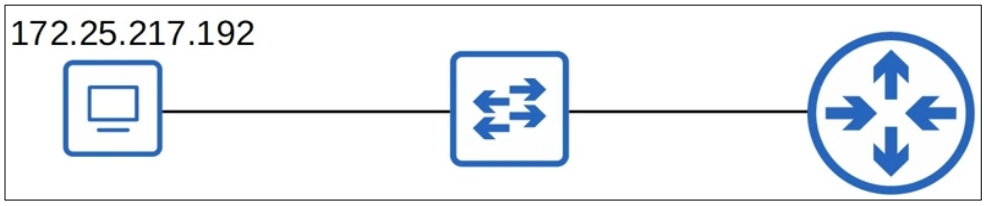
The Answer is:

As the last examples, we will borrow a bit every time and calculate the number of subnets we get with the formula **2^x = subnets**. So, to reach the number of **250** subnets or more, we need to use the prefix length of **/24**, as **2** to the power of **9** will get **256**.

And to calculate the number of hosts we will use the formula of **2^n – 2 = number of hosts**. While **n** is the number of the host bits. So, It will be **2** to the power of **8** minus **2** which equals **254** host per subnet.



→ Identify the subnet:-
What does host **172.25.217.192/21** belong to?



The Answer is:
- Write out the address in binary
- Change all of the host bits to **0**
- And then change the address back to the dotted decimal
- The Answer, by my own is **172.25.216.0/21**.

1 0 1 0 1 1 0 0 . 0 0 0 1 1 0 0 1 . 1 1 0 1 1 0 0 1 . 1 1 0 0 0 0 0 0

172 . 25 . 217 . 192

- Convert the address to binary:

1 0 1 0 1 1 0 0 . 0 0 0 1 1 0 0 1 . 1 1 0 1 1 0 0 0 . 0 0 0 0 0 0 0 0

- Chnage it back to dotted decimal:

172 . 25 . 216 . 0

→ Here's a chart showing the number of available subnets, and the number of available host addresses for each prefix length when sunetting a Class **B** network:

Prefix Length	Number of Subnets	Number of Hosts	Prefix Length	Number of Subnets	Number of Hosts
/17	2	32766	/25	512	126
/18	4	16382	/26	1024	62
/19	8	8190	/27	2048	30
/20	16	4094	/28	4096	14
/21	32	2044	/29	8192	6
/22	64	1022	/30	16384	2
/23	128	510	/31	32768	0 (2)
/24	256	254	/32	65536	0 (1)

- Just know the pattern, for each borrowed bit, the number of subnets doubles, **2**, **4**, **8**, **16**, **32**, etc.
- For each host bit, the number of addresses in each subnet doubles, however you have to subtract **2** to identify the number of usable host addresses.

→ Let's Go to the QUIZ:

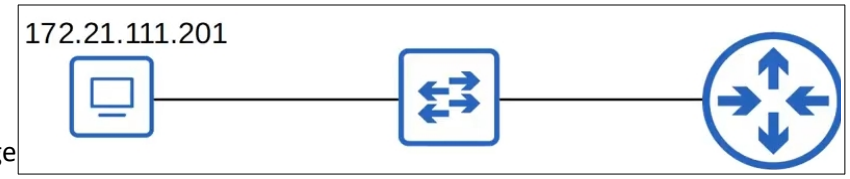
→ Question **1**:
We've been given the **172.30.0.0/16** network. Our company requires **100** subnets with the at least **500** hosts per subnet.
What prefix length should we use??

Answer **1**:

We will borrow a bit every time to reach the number of **100** subnets with the formula 2^x **= subnets**, so we will borrow **7** bits.

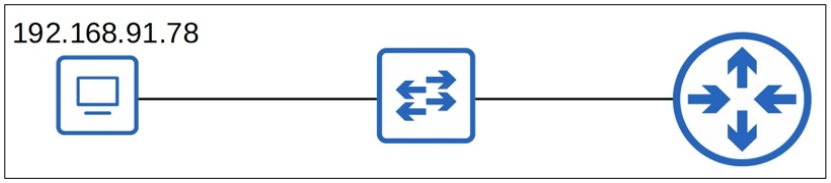
AS **2** to the power of **7** equals **128** bits. And each subnet will have at least **512** subnets due to the formula 2^n **– 2 = hosts**.
So the prefix we should use is **/23**.

→ Question **2**:
What does host **172.21.111.201/20** belong to?



Answer **2**:
We will convert the address to binary, and change the borrowed bits from the binary to **0**, and then we will change it back to dotted decimal. And the host belongs to the network **172.21.96.0/20**.

→ Question **3**:
What is the broadcast address of the network **192.168.91.78/26** belongs to?



Answer **3**:
The subnet address will be **192.168.91.64/26**.
There is also **62** available addresses in this subnet.
And as you know, to get the next subnet, we change the last bit of the network -borrowed- bits to **1**, so the next subnet is **192.168.91.128/26**.
And due to that, the broadcast address is **192.168.91.127**.

→ Question **4**:
You divide the **172.16.0.0/16** network into **4** subnets of equal size. Identify the network and broadcast addresses of the **2nd** subnet.

Answer **4**:
To divide the network to **4** subnets, we need to borrow only **2** bits. So, we will use the prefix length of **/18**.
So to get the network address of the **2nd** subnet, we just change the last bit of the network portion to **1**.
So the network address will be **172.16.64.0/18**.
Simply to get the broadcast address is to change all of the host bits to **1**.
So it will be **172.16.127.255**.

→ Question **5**:
You divide the **172.30.0.0/16** network into subnets of **1000** hosts each. How many subnets are you able to make?

Answer **5**:
To get the number of **1000** hosts, we have to make the host bits to be **10** bits, As **2** to the power of **10** minus **2** equals **1,022** hosts.
So we will use the prefix length of **/22**.
And to get the number of subnets, as we borrowed **6** bits, so it will be **2** to the power of **6** which equals **64** subnets!

Thanks for everything!
@0x0nullian

Day 15: Subnetting pt.3:-

→ Subnetting Class A Networks:-

Look at the *Size of rest bit* field, that is the host portion:
There are **24** host bits that we can borrow from, meaning lots of rooms to make subnets.

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)

OK, just a reminder:

The process of subnetting Class A, Class B, and Class C networks is EXACTLY THE SAME!

→ Let's do a practice question subnetting Class **A** networks:
We've been given the **10.0.0.0/8** network. We must create **2000** subnets which will be distributed to various enterprises.
What is the prefix length should we use? How many *usable addresses* will be in each subnet?

It's **/8**, so only the first octet is the network portion, and we have **3** whole octets to borrow from and make subnets.

```
0 0 0 0 1 0 1 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   10       .       0       .       0       .       0
Subnet Mask:
1 1 1 1 1 1 1 1 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   255      .       0       .       0       .       0
```

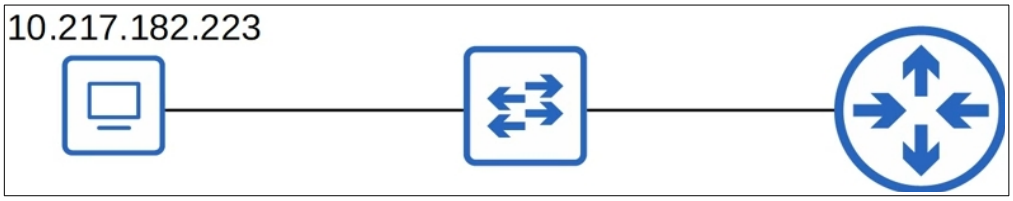
So, to reach the level of dividing the network to **2000** subnets, we need to borrow **11** bits. 2^{11} = **2,048** subnets.

```
0 0 0 0 1 0 1 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   10       .       0       .       0       .       0
Subnet Mask:
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 0 0 0 0 0 . 0 0 0 0 0 0 0 0
   255      .      255      .      224      .       0
```

This means we will use a **/19** prefix length.
Well, there are **13** host bits remaining, that means each subnet will contain $2^{13} - 2$ = **8,190** hosts per subnet.

This the same process we used for Class **B** and Class **C** networks, the numbers are just larger.

→ Let's do another practice question:
PC1 has an IP address of **10.217.182.223/11**. Identify the following:
1) Network and Broadcast addresses.
2) First and Last usable addresses.
3) Number of usable/host addresses.



Answer:

- We convert the IP to binary, changing all the host bits to **0**, to get the network address and once more to **1** to get the broadcast. It's not **/8** mask, It is **/11**, so there are **3** borrowed bits. The subnet mask will be **255.192.0.0/11**.

1- The network address is **10.192.0.0/11**, and the broadcast address is **10.255.255.255**.

2- First usable address is: **10.192.0.1** and the last usable address is: **10.255.255.254**.

3- As the host bits contains **23** bits, the number of usable addresses is $2^{21} - 2 = 8{,}097{,}150$. hosts.

→→ I made one mistake in the last usable address and the broad cast address, We just need to change ONLY the *host bits*.
So, the broadcast address will be **10.223.255.255**

```
0 0 0 0 1 0 1 0 . 1 1 0 1 1 0 0 1 . 1 0 1 1 0 1 1 0 . 1 1 0 1 1 1 1 1
   10      .    217    .    182    .    223

0 0 0 0 1 0 1 0 . 1 1 0 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1
   10      .    223    .    255    .    255
```

And the last usable address will be: **10.223.255.254**.

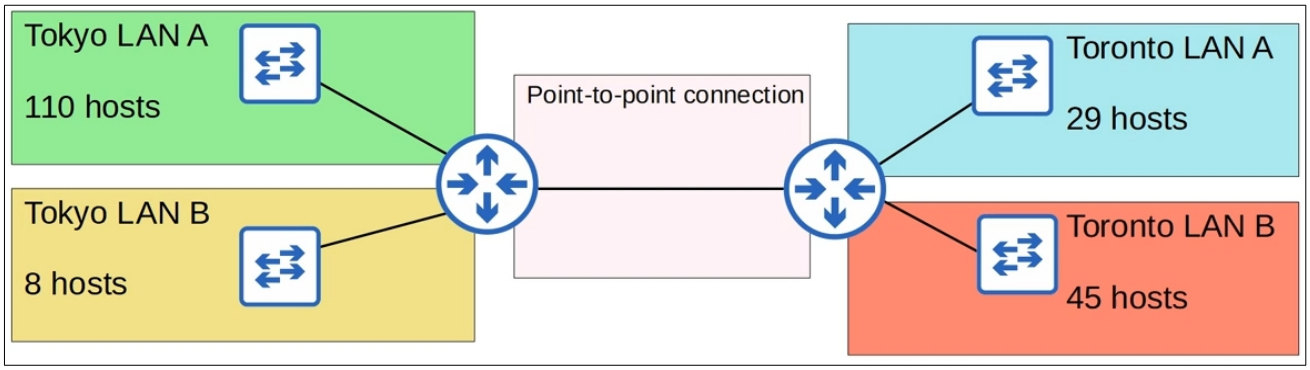
→ VLSM, Variable-Length Sbubnet Masks:-

→ Until now, we have practiced subnetting used **FLSM**. Stands for **Fixed-Length Subnet Masks**.
This means that all of the subnets use the same prefix length (ie. subnetting a Class **C** network into **4** subnets using **/26**).

→ **VLSM**, stands for **Variable-Length Subnet Masks**, is the process of creating subnets of different sizes, to make your use of network addresses more efficient.

→ **VLSM** is more complicated than **FLSM**, but it's easy if you follow the steps correctly.

Here's an example small enterprise network, with two LANs in Tokyo and two LANs in Toronto.
There's also a *point-to-point* connection between the two Routers that we must assign IP addresses for



We are assigned the **192.168.1.0/4** network, and must divide it into **5** subnets to provide IP addresses for all hosts in the enterprise network.
If we were to try this with **FLSM**, we would need to borrow **3** bits to make enough subnets. That would leave five host bits left.
5 host bits only allows **30** host addresses, so that's not enough addresses for Tokyo LAN A, or Toronto LAN B.
However, if we use **VLSM** we can assign different subnet sizes to each LAN, which will allow us to make sure each LAN has enough addresses available.

→ The steps to subnet a network using VLSM:-

1) Assign the largest subnet at the start of the address space.
2) Assign the second-largest after it.
3) Repeat the process until all subnets have been assigned, from largest to smallest.

If we looked at our enterprise network, that means we will assign subnets in this order:
First, Tokyo LAN A, Toronto LAN B, Toronto LAN A, Tokyo LAN B, and finally the *point-to-point* connection between the two Routers.

→ Let's assign **Tokyo LAN A** network: *all solved!*
We decided to use a **/25** prefix length, because it leaves **7** host bits, which means a total of **128** address, so **126** usable addresses.
So, the network address will be **192.168.1.0/25**.
Convert all of the host bits to **1**, and here is the network address, **192.168.1.127**.
The first usable address is the network address **+ 1**, and the last usable address is the broadcast address – **1**.
The first usable is: **192.168.1.1**, and the last usable is **192.168.1.126**.
Total number of usable host addresses is **126**.
That's the **/25** subnet uses *half* of the address space of the **192.168.1.0/24** network, but that's no problem! Using **VLSM**, we can assign smaller subnets to these other LANs, and you'll see that there is enough address space left.

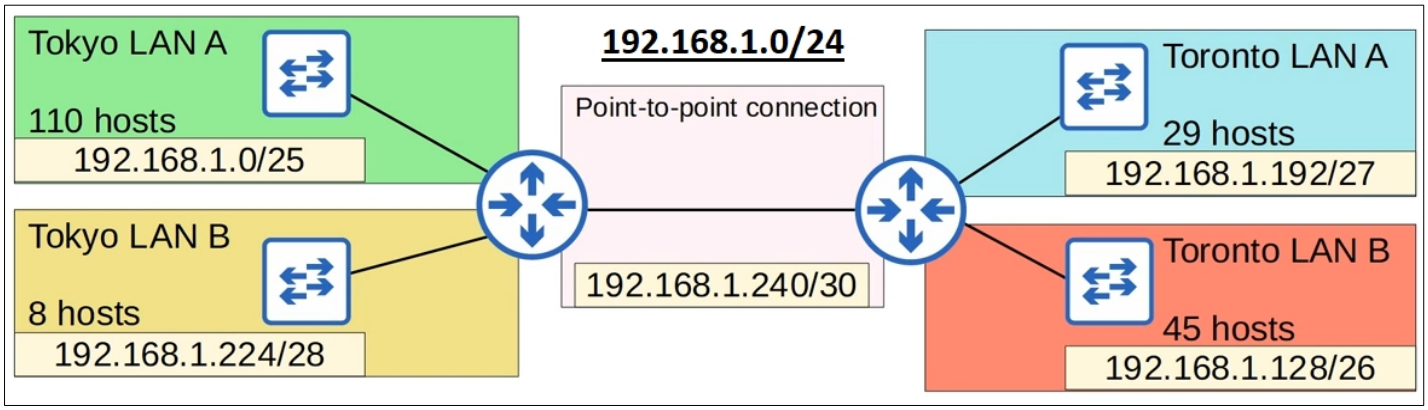
→ Let's assign **Toronto LAN B** network:
The broadcast address of Tokyo LAN A network is **192.168.1.127**, so the network address of Toronto LAN B is: **192.168.1.128/??**.
So, what's the prefix length should we use? We need a subnet for **45** hosts, so we need to borrow another bit, so the prefix is **26**.
The network address is: **192.168.1.128/26**, and the broadcast is **192.168.1.191**.
The first usable address is the network address **+ 1**, and the last usable is the broadcast – **1**.
The first usable is: **192.168.1.129**, and the last usable is: **192.168.1.190**.

→ Let's assign **Toronto LAN A** network:
The broadcast address of Toronto LAN B network is **192.168.1.191**, so the network address of Toronto LAN A is: **192.168.1.192/??**.
So, what's the prefix length should we use? We need a subnet for **29** hosts, so we need to borrow another bit, so the prefix is **27**.
The network address is: **192.168.1.192/27**, and the broadcast is **192.168.1.223**.
The first usable address is the network address **+ 1**, and the last usable is the broadcast – **1**.
The first usable is: **192.168.1.193**, and the last usable is: **192.168.1.222**.

→ Let's assign **Tokyo LAN B** network:
The broadcast address of Toronto LAN A network is **192.168.1.223**, so the network address of Tokyo LAN B is: **192.168.1.224/??**.
So, because Tokyo LAN B requires **8** hosts, we must use a **/28** prefix length, which allows **14** host addresses.
Possible Mistake here is using a **/29** prefix length. Although **/29** allows **8** addresses, remember we must subtract two for the network and the broadcast addresses. So, really **/29** only allows **6** usable addresses.
The network address is: **192.168.1.224/28**, and the broadcast is **192.168.1.239**.
The first usable address is the network address **+ 1**, and the last usable is the broadcast – **1**.
The first usable is: **192.168.1.225**, and the last usable is: **192.168.1.238**.

→ Let's assign ***point-to-point*** connection:
We've used address space until **192.168.1.239**. There's not a lot of space left, but that's no problem.
Point-to-point connection only requires **2** addresses.
192.168.1.239 is the broadcast address of Tokyo LAN B, therefore **192.168.1.240** is the network address of the point-to-point.
Now, what prefix length should we use???????
As We've learned before, it is possible to use a **/31** prefix length for a subnet requiring only two hosts.
HOWEVER, for the CCNA test, if you're asked what prefix length to use for a subnet that requires two hosts,
It is RECOMMENDED ***not*** using a **/31**.
Instead, what other prefix length allows for **2** hosts? It is **/30**. So, the **192.168.1.240/30** is the network address, and **192.168.1.243** is the broadcast address.
The first usable address is **192.168.1.241**, and the last usable address is **192.168.1.242**.
One for **Toronto** Router and one for **Tokyo** Router.

Okay, so we successfully subnetted this network using **VLSM**, and there is still a little bit of address space left.



NOTE: Each subnet uses a different prefix length.
If we tried o use the same prefix length for each subnet, there wouldn't be enough address space, but with **VLSM** we were able to do it and leave some extra address space at the end.

→ Additional Practice resources for subnetting:

→ http://subnetting.org/
→ http://www.subnettingquestions.com/
→ https://subnettingpractice.com/

End of Subnetting ;)
By: @0x0nulian