

# how to master CCNA

"The roadmap" to your  
CCNA certificate



**"The roadmap"**  
to your CCNA  
certificate

# René Molenaar

All contents copyright C 2002-2013 by René Molenaar. All rights reserved. No part of this document or the related files may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher.

**Limit of Liability and Disclaimer of Warranty:** The publisher has used its best efforts in preparing this book, and the information provided herein is provided "as is." René Molenaar makes no representation or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose and shall in no event be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential, or other damages.

**Trademarks:** This book identifies product names and services known to be trademarks, registered trademarks, or service marks of their respective holders. They are used throughout this book in an editorial fashion only. In addition, terms suspected of being trademarks, registered trademarks, or service marks have been appropriately capitalized, although René Molenaar cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark, registered trademark, or service mark. René Molenaar is not associated with any product or vendor mentioned in this book.

## Introduction

One of the things I do in life is work as a Cisco Certified System Instructor (CCSI) and after teaching CCNA for a few years I've learned which topics people find difficult to understand. This is the reason I created <http://gns3vault.com> where I offer free Cisco labs and videos to help people learn networking. The problem with networking is that you need to know what you are doing before you can configure anything. Even if you have all the commands you still need to understand *what* and *why* you are typing these commands. I created this book to give you a compact guide which will provide you the answer to *what* and *why* to help you master the CCNA exam.

I have tried to put all the important keywords in **bold**. If you see a **term or concept** in **bold** it's something you should remember / write down and make sure you understand it since its core knowledge for your CCNA!

One last thing before we get started. When I'm teaching I always advise students to create mindmaps instead of notes. Notes are just lists with random information while mindmaps show the relationship between the different items. If you are reading this book on your computer I highly suggest you download "Xmind" which you can get for free here:

<http://xmind.net>

If you are new to mindmapping, check out "Appendix A – How to create mindmaps" at the end of this book where I show you how I do it.

I also highly recommend you to follow me along when I'm demonstrating the configuration examples. Boot up GNS3 and/or your switches and configure the examples I'm showing you by yourself. You'll learn more by *actively* working on the equipment compared to just *passive* reading.

Enjoy reading my book and good luck getting your CCNA certification!

René Molenaar

P.S. If you have any questions or comments about this book, please let me know:

E-mail: [info@gns3vault.com](mailto:info@gns3vault.com)  
Website: [gns3vault.com](http://gns3vault.com)  
Facebook: [facebook.com/gns3vault](https://facebook.com/gns3vault)  
Twitter: [twitter.com/gns3vault](https://twitter.com/gns3vault)  
Youtube: [youtube.com/gns3vault](https://youtube.com/gns3vault)

## Index

|  |     |
|--|-----|
| Introduction .....   | 3   |
| 1. Lab Equipment.....                                      | 5   |
| 2. Basics of networking .....                              | 10  |
| 3. The OSI-Model .....                                     | 16  |
| 4. The network layer: IP Protocol .....                    | 24  |
| 5. The Transport Layer: TCP and UDP .....                  | 40  |
| 6. Ethernet: Dominating your LAN for over 30 years.....    | 48  |
| 7. Introduction to Cisco IOS .....                         | 58  |
| 8. Hubs, Bridges and Switches .....                        | 87  |
| 9. Virtual LANs (VLANs), Trunks and VTP .....              | 102 |
| 10. Etherchannel (Link Aggregation).....                   | 143 |
| 11. Spanning-Tree (STP).....                               | 152 |
| 12. Binary, Subnetting and Summarization.....              | 183 |
| 13. IP Routing .....                                       | 208 |
| 14. FHRP (First Hop Redundancy Protocols) .....            | 229 |
| 15. Distance Vector Routing Protocols.....                 | 249 |
| 16. OSPF – Link-state routing protocol.....                | 264 |
| 17. EIGRP – Cisco’s Hybrid Routing Protocol .....          | 294 |
| 18. Security: Keeping the bad guys out. ....               | 312 |
| 19. Network and Port address Translation (NAT & PAT) ..... | 330 |
| 20. Wide area networks .....                               | 342 |
| 21. Introduction to IPv6.....                              | 379 |
| 22. IPv6 NPD and Host Configuration.....                   | 400 |
| 23. IPv6 Routing .....                                     | 409 |
| 24. Virtual Private Networks .....                         | 425 |
| 25. Network Management .....                               | 433 |
| 26. IOS Licensing .....                                    | 457 |
| 27. Final Thoughts.....                                    | 464 |
| Appendix A – How to create mindmaps .....                  | 465 |



## 1. Lab Equipment

*"If I had eight hours to chop down a tree, I'd spend six hours sharpening my ax"*  
~Abraham Lincoln

Before we are going to start on our networking journey we will take a look at the networking equipment that you will need. If you want to master the CCNA exam you'll have to do two things:

- Read this book so you learn about all the different protocols and **understand the theory**.
- Implement your knowledge by **configuring** these protocols on our routers and switches.

So what equipment should you get?

For most of the labs you can use GNS3. This is an emulator that runs the Cisco IOS software but you can only **emulate routers...no switches**. You can download GNS3 for free from <http://gns3.net> but you'll have to supply the IOS image yourself. Cisco owns the copyright on IOS so it can't be shared freely. I suggest using the 3640 or 3725 router in GNS3.



Courtesy of Cisco Systems, Inc. Unauthorized use not permitted.

The closest you can get to emulate a switch in GNS3 is inserting this NM16-ESW Etherswitch module in your virtual router.

It adds 16 switch ports to your virtual router and supports basic switching features. Unfortunately this module is very limited and I don't recommend using it for CCNA.

GNS3 isn't very difficult to work with but there is one thing you need to be aware of. Most people complain that whenever they start an emulated router that they see their CPU jump to 100%. You can fix this by setting a correct IDLEPC value. If you are configuring GNS3 you need to check this video where I explain you how to do it:

<https://www.youtube.com/watch?v=NkEv6v6rqlA>

So what do we need? My advice is to use **GNS3 for all your routing labs** and buy some **real physical switches for the switching labs**. Don't be scared...I'm not going to advise you to buy ultra-high tech brand new switches! We are going to buy used Cisco switches that are easy to find and they won't burn a hole in your wallet...

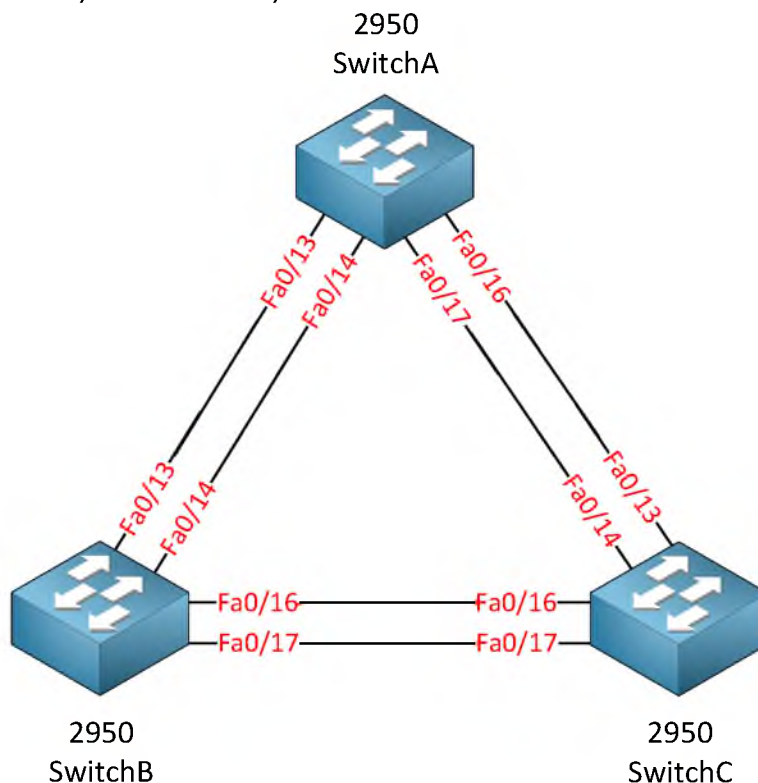
Without further ado...here are our candidates:



Courtesy of Cisco Systems, Inc. Unauthorized use not permitted.

**Cisco Catalyst 2950:** This is a layer 2 switch that does **everything** you need for CCNA.

If you look at eBay you can find the Cisco Catalyst 2950 for around \$30. It doesn't matter if you buy the 8, 24 or 48 port model. Not too bad right? Keep in mind you can sell them once you are done with CCNA without losing (much) money. This switch is cheap and perfect for CCNA! Once you have your switches you should connect them like this:



If you plan to study CCNP after completing CCNA I can highly recommend swapping one Cisco Catalyst 2950 for a **Cisco Catalyst 3550**.



Courtesy of Cisco Systems, Inc. Unauthorized use not permitted.

**Cisco Catalyst 3550:** It offers pretty much the same features as the 2950 but it also supports routing which we require for CCNP.

What about other switch models? Anything else we can use for CCNA?

- The Cisco Catalyst 2960 is the successor of the Cisco Catalyst 2950, it's a great layer 2 switch but more expensive.
- The Cisco Catalyst 3560 is the successor of the Cisco Catalyst 3550, it also offers routing features but it's quite more expensive...around \$300 on eBay.
- The Cisco Catalyst 3750 is also a switch that can do routing but it's very expensive.

My advice is to get the 3x Cisco Catalyst 2950 or 2x Cisco Catalyst 2950 and 1x Cisco Catalyst 3550 if you want to study CCNP after your CCNA.

Are there any switches that you should **NOT** buy?

- Don't buy the Cisco Catalyst 2900XL switch; you'll need at least the Cisco Catalyst 2950 switch. Many features are not supported on the Cisco Catalyst 2900XL switch.
- Don't buy the Cisco Catalyst 3500XL switch, same problem as the one above.

You also have to buy some cables:



Above you see the blue Cisco console cable. It probably comes with the switch but make sure you have at least one. You'll need this to configure your switches.



If your computer doesn't have any serial ports to connect your blue Cisco console cable you need to get one of these. It's a USB to serial port converter.



Courtesy of König Electronic Inc. Unauthorized use not permitted.

I also like to use one of these. It's a USB connector with 4x RS-232 serial connectors you can use for your blue Cisco console cables to connect to your switches.

It saves the hassle of plugging and unplugging your console cable between your switches.

The one I'm using is from KÖNIG and costs around \$30. Google for "USB 4x RS-232" and you should be able to find something similar.



Between the switches you'll require UTP cables. There's a difference between straight through and crossover cables (we'll talk about that later in the book). Modern switches and network cards support auto-sensing so it really doesn't matter what kind of cable you use.

If you are going to connect your 2950 switches to each other make sure you **buy crossover cables** since they don't support auto-sensing!

It will be useful if you have one old extra computer or laptop that you can use to connect to your switches.

Now you know the equipment that you need, it's time to dive into networking!

## 2. Basics of networking

Before we start digging into complex stuff we'll have a little talk about networks.

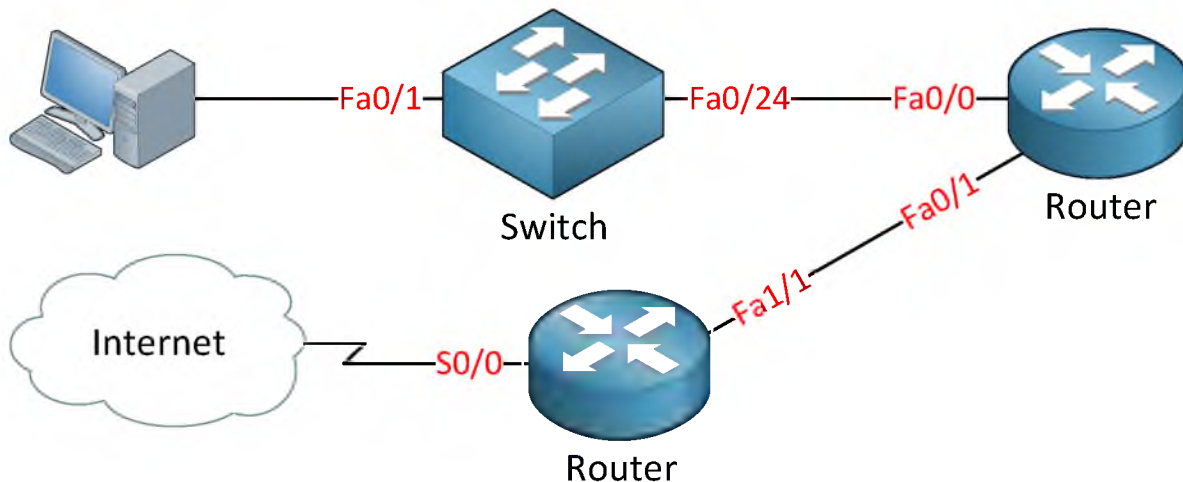
What is a network anyway?

A network is just a collection of devices and end systems connected to each other and able to communicate with each other. These could be computers, servers, smartphones, routers etc. A network could be as large as the internet or as small as your two computers at home sharing files and a printer.

Some of the components that make up a network:

- **Personal Computers (PC):** These are the endpoint of your network, sending and receiving data.
- **Interconnections:** These are components that make sure data can travel from one device to another, you need to think about:
  - Network Cards: they translate data from your computer in a readable format for the network.
  - Media: network cables, perhaps wireless.
  - Connectors: the plug you plug in your network card.
- **Switches:** These boxes are network devices which provide a network connection for your end devices like PC's.
- **Routers:** Routers interconnect networks and choose the best path to each network destination.

If you are going to work with Cisco you'll have to get used to some network diagrams like the one below:



So what do we see in the network diagram above? First of all we see a computer connected to a switch. On the switch side you see "Fa0/1" which means the computer is connected to the FastEthernet 0/1 interface on the switch side. The 0 is the controller number (usually 0 on smaller switches) and the 1 is the port number. Our switch is connected to a router using its FastEthernet 0/24 interface. Our routers are connected using FastEthernet as well. The router at the bottom has a connection to the Internet using a Serial connection.



Don't worry about what a switch or router is and the difference between them; we'll get to that later!

So why do we use networks? I think this one is obvious since you are using networks on a daily basis but let's sum up what we use networks for:

- **Applications:** Sending data between computers, sharing files.
- **Resources:** Network printers, network cameras.
- **Storage:** Using a NAS (Network attached storage) will make your storage available on the network. Many people use one at home nowadays to share files, videos and pictures between computers.
- **Backup:** Using a central backup server where all computers send their data to for backup.
- **VoIP:** Voice over IP is becoming more important every day and replacing analog telephony.

We are all using applications on a daily basis but if we look at them with a network-minded view we can divide them in 3 different categories:

- **Batch applications**
  - File transfers like FTP, TFTP, perhaps a HTTP download. Could be a backup at night.
  - No direct human interaction.
  - High bandwidth is important but not critical.

A batch application is something you just let run and you don't care if it takes a minute more or less since nobody is "waiting" for a response. This could be a backup job overnight. It doesn't matter if it takes an hour or more; however, if it takes days then it's a problem.



*TFTP is like a 'stripped down' version of FTP and is used sometimes to copy files from and to a Cisco router or switch.*

- **Interactive applications**
  - Human-to-Human interaction
  - Someone is waiting for a response, so response time (delay) is important.

With interactive applications you need to think about someone who is working on a database server and sending commands. Once you press enter you want it to respond fast but a second more or less is perhaps not THAT annoying. Another example is two users who are using a chat application, you don't want to wait 20 seconds before you receive the message from another user but a second more or less doesn't matter.

- **Real-time applications**
  - Also Human-to-Human interaction
  - VoIP (Voice over IP) or live Video conferencing.
  - End-to-end delay is critical.

Imagine you are talking to someone on the phone using Voice over IP and you need to wait 2 seconds before you hear a reply...this is VERY annoying and it's hard to have a

conversation like that. Everything above 300ms of delay (1000ms is a second) you will have a hard time having a good conversation since it'll be more like a "walkie-talkie" conversation. Latency is critical when using VoIP or live Video. A delay above 150ms (1/8 of a second) is noticeable.

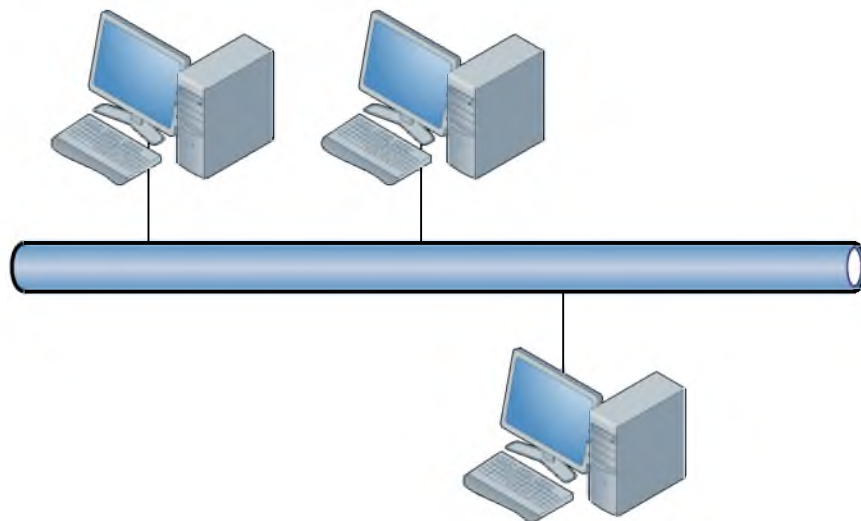
When we look at networks we have different types of "Topologies" and we have two different topologies:

- **Physical topology**
- **Logical topology**

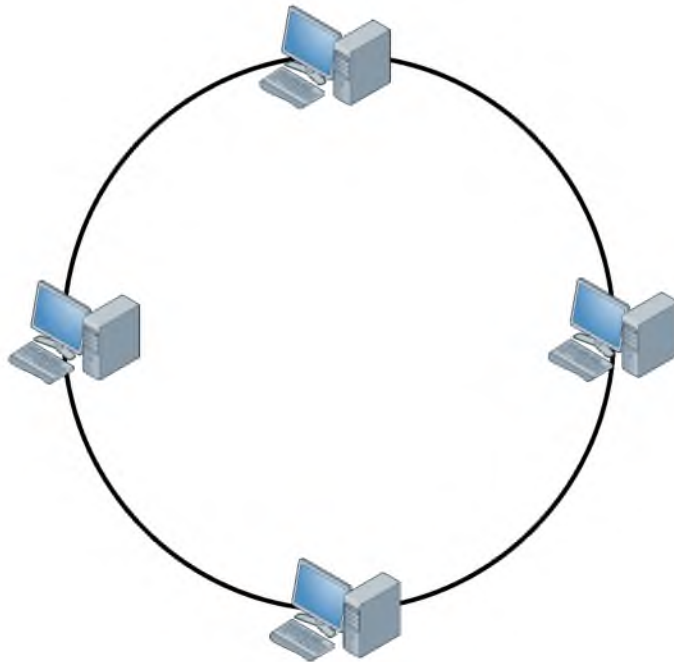
There's an important difference between the two. The physical topology is what the network looks like and how all the cables and devices are connected to each other. The logical topology is *the path our data signals take through the physical topology*.

There are multiple types of physical topologies:

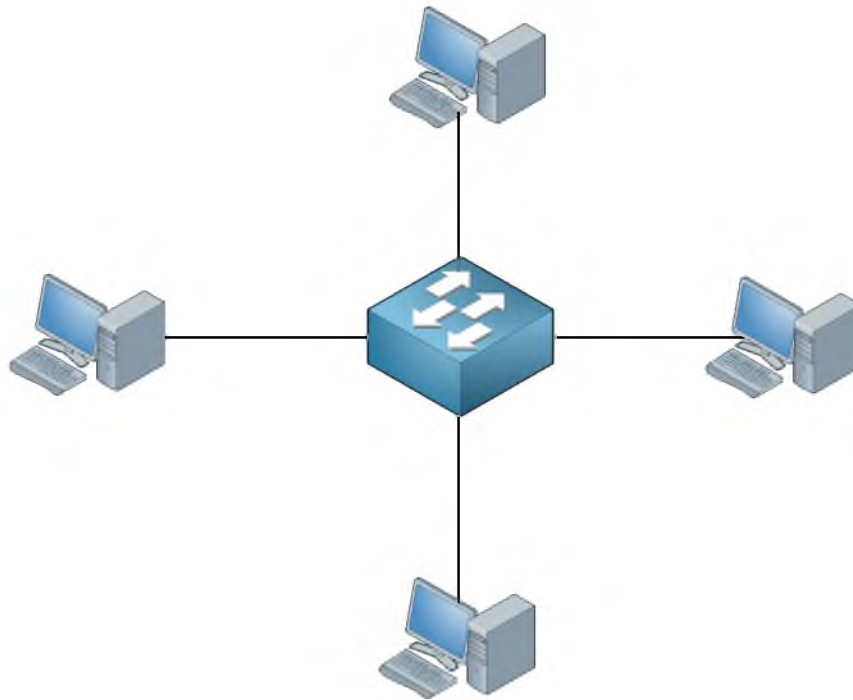
- **Bus topology:** One of the first networks was based on coax-cables. This was basically just one long cable and every device was connected to it. At the end of the cable you had to place a terminator. If the cable breaks then your network is down.



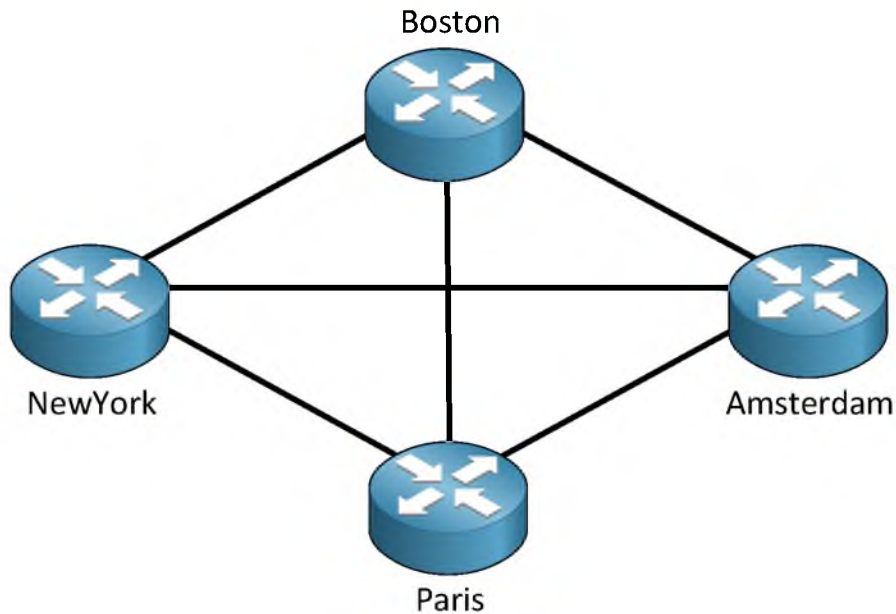
- **Ring topology:** All computers and network devices are connected on a cable and the last two devices are connected to each other to form a "ring". If the cable breaks your network is down. There's also a "dual-ring" setup for redundancy, this is just another cable to make sure if one cable breaks your network isn't going down.



- **Star topology:** All our end devices (computers) are connected to a central device creating a star model. This is what we use nowadays on local area networks (LAN) with a switch in the middle. The physical connections we normally use is UTP (Unshielded twisted pair) cable. Of course when your switch goes down your network is down as well.

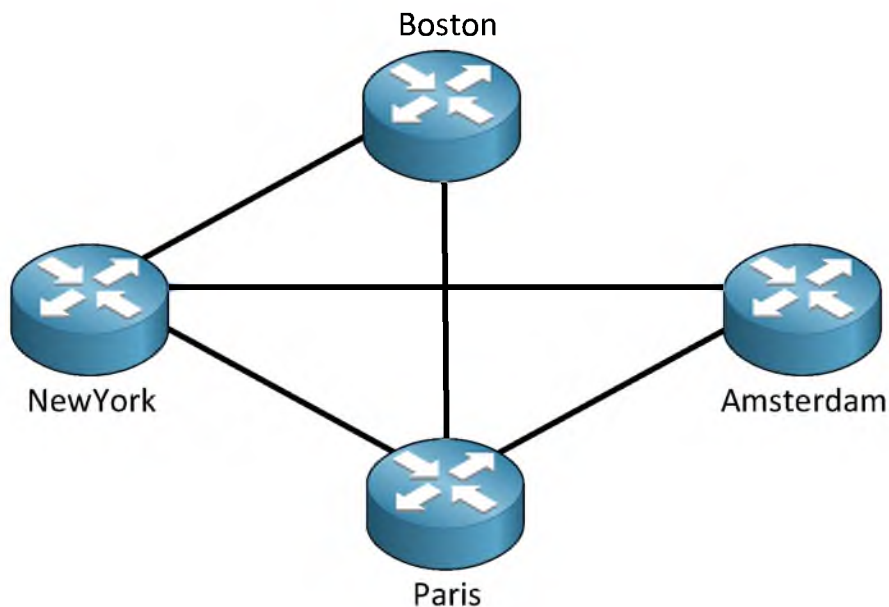


The example above is what we normally use on our local area networks (LAN). Now let's take a look at the following picture where we have a company with multiple sites in different cities.



In the example above every router is connected to every other router. This, of course, is very resistant to failure since a single link failure will not bring our network down. The downside of this setup is that it's very expensive. You need multiple links between the sites and each router needs extra interfaces. This is what we call **full-mesh**.

Another option is to make sure the important sites have connections to all other sites like in the following picture.



Here you can see router New York has a connection to all other routers, Boston is only connected to New York and Amsterdam has a connection to New York and Paris. This is a trade-off between fault tolerance and cost (it's always about money right?). We call this **partial-Mesh**.

In the next chapter we'll dive deeper into the basics of networking.

### 3. The OSI-Model

In the beginning the development of networks was chaotic. Each vendor had its own proprietary solution. The bad part was that one vendor's solution was not compatible with another vendor's solution. This is where the idea for the OSI-model was born, having a layered approach to networks our hardware vendors would design hardware for the network, and others could develop software for the application layer. Using an open model which everyone agrees on means we can build networks that are compatible with each other.

To fix this problem the International Organization for Standardization (ISO) researched different network models and the result is the OSI-model which was released in 1984. Nowadays most vendors build networks based on the OSI model and hardware from different vendors is compatible....excellent!

The OSI-model isn't just a model to make networks compatible; it's also one of the **BEST** ways to teach people about networks. Keep this in mind since I'll be referring a lot to the OSI-model, it's very useful!

|         |              |
|---------|--------------|
| Layer 7 | Application  |
| Layer 6 | Presentation |
| Layer 5 | Session      |
| Layer 4 | Transport    |
| Layer 3 | Network      |
| Layer 2 | Data Link    |
| Layer 1 | Physical     |

***"All People Seem To Need Data Processing"***

This is the OSI-model which has seven layers; we are working our way from the bottom to the top.



Let's start at the physical layer:

- **Physical Layer:** This layer describes stuff like voltage levels, timing, physical data rates, physical connectors and so on. Everything you can "touch" since it's physical.
- **Data Link:** This layer makes sure data is formatted the correct way, takes care of error detection and makes sure data is delivered reliably. This might sound a bit vague now, for now try to remember this is where "Ethernet" lives. MAC Addresses and Ethernet frames are on the Data Link layer.
- **Network:** This layer takes care of connectivity and path selection (routing). This is where IPv4 and IPv6 live. Every network device needs a unique address on the network.
- **Transport:** The transport layer takes care of transport, when you downloaded this book from the Internet the file was sent in segments and transported to your computer.
  - **TCP** lives here; it's a protocol which send data in a reliable way.
  - **UDP** lives here; it's a protocol which sends data in an unreliable way.

I'm taking a short break here, these four layers that I just described are important for **networking**, and the upper three layers are about **applications**.

- **Session:** The session layer takes care of establishing, managing and termination of sessions between two hosts. When you are browsing a website on the internet you are probably not the only user of the webserver hosting that website. This webserver needs to keep track of all the different "sessions".
- **Presentation:** This one will make sure that information is readable for the application layer by formatting and structuring the data. Most computers use the ASCII table for characters. If another computer would use another character like EBCDIC than the presentation layer needs to "reformat" the data so both computers agree on the same characters.
- **Application:** Here are your applications. E-mail, browsing the web (HTTP), FTP and many more.

## ***"People Do Need To See Pamela Anderson"***

This one normally gives me more smiles when I'm teaching CCNA in class and it's another way to remember the OSI-Model.

P = Physical  
D = Data Link  
N = Network  
T = Transport  
S = Session  
P = Presentation  
A = Application

Remember that you can't skip any layers in the OSI-model, it's impossible to jump from the Application layer directly to the Network layer. You always need to go through all the layers to send data over the network.

Let's take a look at a real life example of data transmission.

1. You are sitting behind your computer and want to download some files of a local webserver. You start up your web browser and type in the URL of your favorite website. Your computer will send a message to the web server requesting a certain web page. You are now using the HTTP protocol which lives on the application layer.
2. The presentation layer will structure the information of the application in a certain format.
3. The session layer will make sure to separate all the different sessions.
4. Depending on the application you want a reliable (TCP) or unreliable (UDP) protocol to transfer data towards the web server, in this case it'll choose TCP since you want to make sure the webpage makes it to your computer. We'll discuss TCP and UDP later.
5. Your computer has a unique IP address (for example 192.168.1.1) and it will build an IP packet. This IP packet will contain all the data of the application, presentation and session layer. It also specifies which transport protocol it's using (TCP in this case) and the source IP address (your computer 192.168.1.1) and the destination (the web server's IP address).
6. The IP packet will be put into an Ethernet Frame. The Ethernet frame has a source MAC address (your computer) and the destination MAC address (web server). More about Ethernet and MAC addresses later.
7. Finally everything is converted into bits and sent down the cable using electric signals.

Once again, you are unable to "skip" any layers of the OSI model. You always have to work your way through ALL layers. If you want a real life story converted to networking land just think about the postal service:

1. First you write a letter.
2. You put the letter in an envelope.
3. You write your name and the name of the receiver on the envelope.
4. You put the envelope in the mailbox.
5. The content of the mailbox will go to the central processing office of the postal service.
6. Your envelope will be delivered to the receiver.
7. They open the envelope and read its contents.

If you put your letter directly in the mailbox it won't be delivered. Unless someone at the postal office is friendly enough to deliver it anyway, in network-land it doesn't work this way!

Going from the application layer all the way down to the physical layer is what we call **encapsulation**. Going from the physical layer and working your way up to the application layer is called **de-encapsulation**.

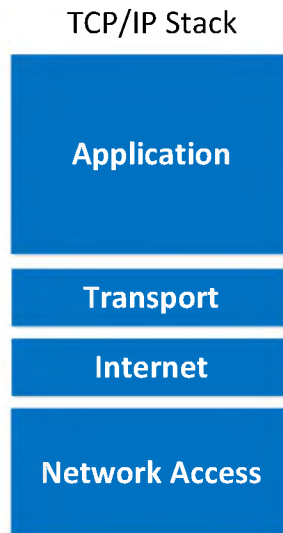
Now you know about the OSI-model, the different layers and the function of each layer. During peer-to-peer communication each layer has 'packets of information'. We call these protocol data units (PDU). Now every unit has a different name on the different layers:

- Transport layer: Segments; For example we talk about **TCP segments**.
- Network layer: Packets; For example we talk about **IP packets** here.
- Data link layer: Frames; For example we talk about **Ethernet frames** here.

This is just terminology but don't mix up talking about IP frames and Ethernet packets...

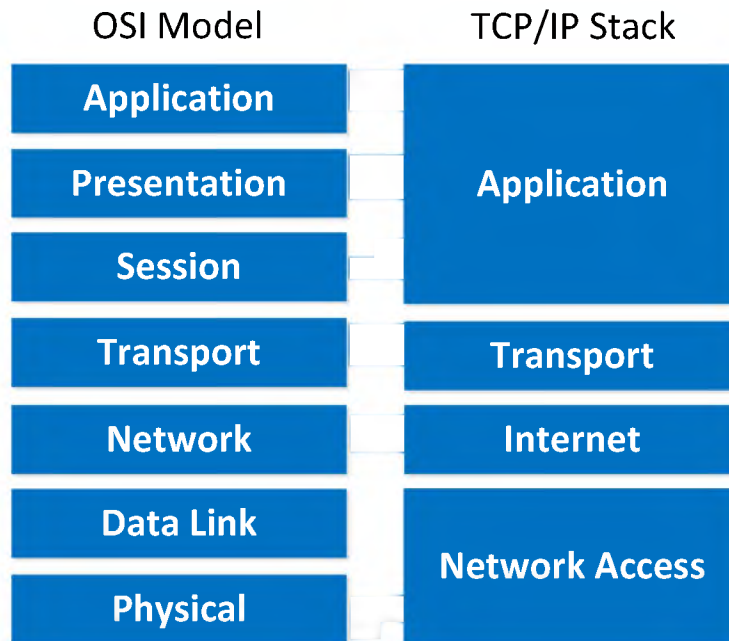
Excellent so now you know everything you need about the OSI-model and the different layers. We'll be looking at the different layers throughout this book so you'll get some more "practice" remembering them.

Besides the OSI-model there was another organization that created a similar model which never became quite as popular. However for your CCNA you'll need to know what it looks like. It's called the TCP/IP stack and it's similar except some of the layers are combined and have different names.



As you can see the upper three layers are now combined to the "Application layer". The network layer is called the "Internet" layer and the bottom 2 layers are combined into the "Network Access" layer.

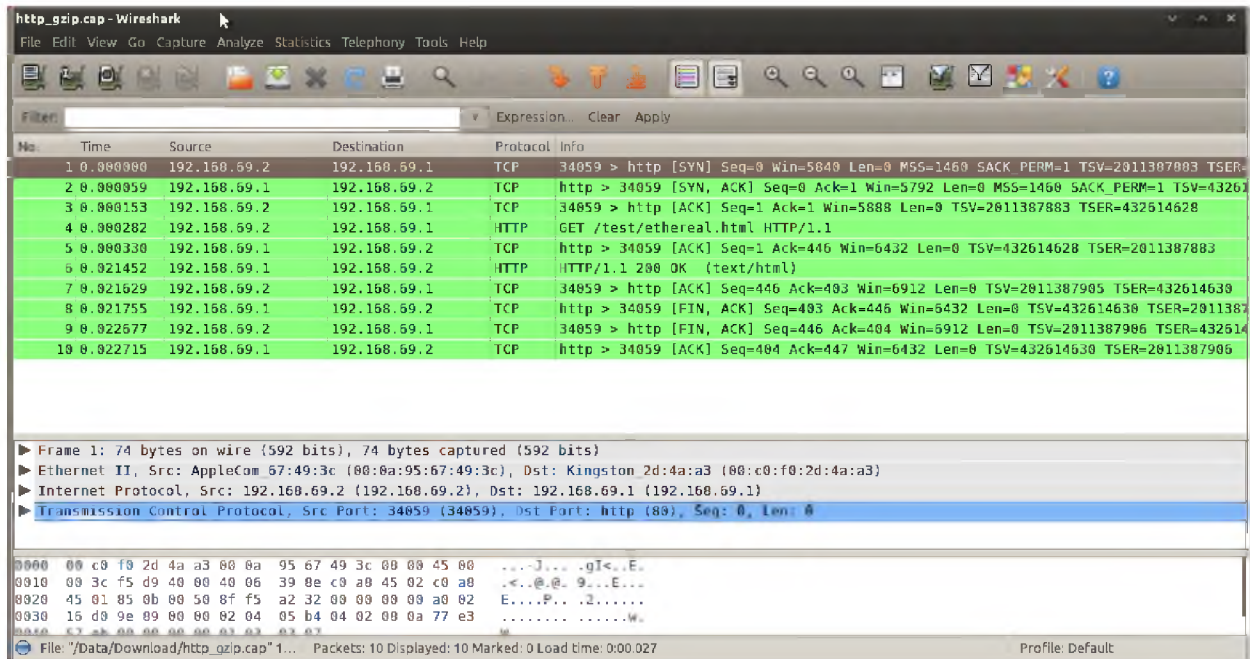
Here's a comparison between the two models:



Basically it's the same idea, same model except with some layers combined and different names. The physical and data link layer are combined into the network access layer. The network layer is now the internet layer and the session, presentation and application layer are combined into a single application layer.

I want to show you an example of what this looks like on a "live" network and the best way to do this is by using Wireshark. Wireshark is a protocol sniffer which will show you all the data that is being sent and received on your network card.

You can download Wireshark (it's free) from <http://wireshark.org>.



The example in the picture above is a capture of a computer requesting a webpage from a webserver. I didn't capture this one myself since the Wireshark website has a lot of good example captures. If you want to look at this capture on your own computer you can download it here:

[http://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=view&target=http\\_gzip.capture](http://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=view&target=http_gzip.capture)

| No. | Time     | Source       | Destination  | Protocol |
|-----|----------|--------------|--------------|----------|
| 1   | 0.000000 | 192.168.69.2 | 192.168.69.1 | TCP      |
| 2   | 0.000059 | 192.168.69.1 | 192.168.69.2 | TCP      |
| 3   | 0.000153 | 192.168.69.2 | 192.168.69.1 | TCP      |
| 4   | 0.000282 | 192.168.69.2 | 192.168.69.1 | HTTP     |
| 5   | 0.000330 | 192.168.69.1 | 192.168.69.2 | TCP      |
| 6   | 0.021452 | 192.168.69.1 | 192.168.69.2 | HTTP     |
| 7   | 0.021629 | 192.168.69.2 | 192.168.69.1 | TCP      |
| 8   | 0.021755 | 192.168.69.1 | 192.168.69.2 | TCP      |
| 9   | 0.022677 | 192.168.69.2 | 192.168.69.1 | TCP      |
| 10  | 0.022715 | 192.168.69.1 | 192.168.69.2 | TCP      |

You can see there are ten IP packets here, with the source IP address and the destination IP address. It also shows you which protocol this IP packet is carrying.

```
▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: AppleCom_67:49:3c (00:0a:95:67:49:3c), Dst: Kingston_2d:4a:a3 (00:c0:f0:2d:4a:a3)
▶ Internet Protocol, Src: 192.168.69.2 (192.168.69.2), Dst: 192.168.69.1 (192.168.69.1)
▶ Transmission Control Protocol, Src Port: 34059 (34059), Dst Port: http (80), Seq: 0, Len: 0
```

Here you see one of the Ethernet frames. Do you see the different layers of the OSI-model?

- Frame 1 / Ethernet II: This is the Data Link layer.
- Internet Protocol: This is the Network layer.
- Transmission Control Protocol: This is the Transport layer.

If we click on the arrows we can see its contents.

```
▼ Frame 4: 511 bytes on wire (4088 bits), 511 bytes captured (4088 bits)
  Arrival Time: Oct 29, 2004 07:21:00.402698000 CEST
  Epoch Time: 1099027260.402698000 seconds
  [Time delta from previous captured frame: 0.000129000 seconds]
  [Time delta from previous displayed frame: 0.000129000 seconds]
  [Time since reference or first frame: 0.000282000 seconds]
  Frame Number: 4
  Frame Length: 511 bytes (4088 bits)
  Capture Length: 511 bytes (4088 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ip:tcp:http]
  [Coloring Rule Name: HTTP]
  [Coloring Rule String: http || tcp.port == 80]
▼ Ethernet II, Src: AppleCom_67:49:3c (00:0a:95:67:49:3c), Dst: Kingston_2d:4a:a3 (00:c0:f0:2d:4a:a3)
  ▶ Destination: Kingston_2d:4a:a3 (00:c0:f0:2d:4a:a3)
  ▶ Source: AppleCom_67:49:3c (00:0a:95:67:49:3c)
  Type: IP (0x0800)
```

I just clicked on the arrows and you can see the contents of the Ethernet Frame. Don't worry if you have no idea what you see here we'll talk about it later. What I want to show you here is the last line, it says "Type: IP (0x0800)".

What it means is that this computer is carrying an IP packet. Let's see if we can see the contents of this IP packet.

```
▼ Internet Protocol, Src: 192.168.69.2 (192.168.69.2), Dst: 192.168.69.1 (192.168.69.1)
  Version: 4
  Header length: 20 bytes
  ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 497
  Identification: 0xf5db (62939)
  ▶ Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  ▶ Header checksum: 0x37d7 [correct]
  Source: 192.168.69.2 (192.168.69.2)
  Destination: 192.168.69.1 (192.168.69.1)
```



Interesting...we can see the source IP and destination IP address. If you look closely you see there's a line which says "Protocol: TCP (6)". This is how the IP packet specifies which transport protocol it is carrying, in this case TCP.

Let's take a look at that TCP segment:

```
▼ Transmission Control Protocol, Src Port: 34059 (34059), Dst Port: http (80), Seq: 1, Ack: 1, Len: 445
  Source port: 34059 (34059)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 1      (relative sequence number)
  [Next sequence number: 446      (relative sequence number)]
  Acknowledgement number: 1      (relative ack number)
  Header length: 32 bytes
  ► Flags: 0x18 (PSH, ACK)
  Window size: 5888 (scaled)
  ► Checksum: 0x16ca [validation disabled]
  ► Options: (12 bytes)
  ► [SEQ/ACK analysis]
```

Don't let all this information get to you, I only want to show you the field that says "Destination port: http (80)". This is how the transport layer tells us for which application this information is meant, we are using port numbers to do so. In this case port 80 for HTTP traffic.

Pretty neat huh? If you feel like it play around a bit with Wireshark and look at some of the packets. If you want to see some pre-captured packets check out the Wireshark website:

<http://wiki.wireshark.org/SampleCaptures>

We are now at the end of this chapter, you have learned about the OSI-model and it's different layers and seen some Wireshark captures to see the different layers in action.

If you want a visual representation of the OSI-model and how a network functions you should check out the "Warriors of the Net" movie. It's a 13 minute free movie which shows you how IP packets make their way to their destination; I think it's a great watch so grab a snack and let this information sink in:

<http://www.warriorsofthe.net/movie.html>

## 4. The network layer: IP Protocol

Let's talk about IP!

IP (Internet Protocol) determines where we are going to send packets to by looking at the destination IP address. How we determine where to send them is up to the routing protocol, we'll talk more about routing later.

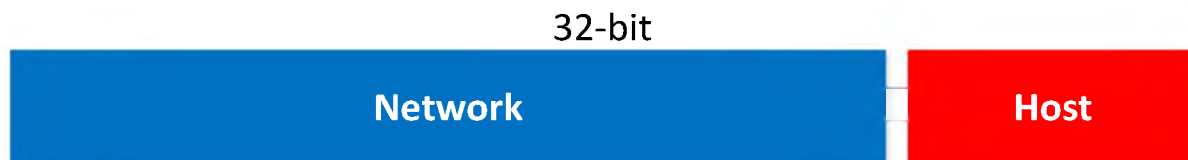
IP uses Packets called IP packets to carry information. Every IP packet is a single unit of information and besides data it carries information to determine where to send the packet.

Let's take a look at some of its characteristics:

- Operates at the **network layer** of the OSI model.
- Connectionless protocol: IP itself does not setup a connection, in order to transport data you need the "transport" layer and use TCP or UDP.
- Every packet is treated independently; there is no order in which the packets are arriving at their destination.
- Hierarchical: IP addresses have a hierarchy; we'll discuss this a bit more in depth when we talk about subnetting and subnet masks.

We need an IP address to uniquely identify each network device on the network. An IP address is just like a phone number (I'm talking about regular phone numbers, no cellphones). Everyone in a city who has a phone at home has a unique phone number where you can reach them.

An IP address is 32-bit and consists of 2 parts, the network part and the host part:



The IP address is 32-bit but we write it down in 4 blocks of 8 bits. 8 bits is what we call a "byte". So the IP address will look like this:



The network part will tell us to which "network" the IP address will belong, you can compare this to the city or area code of a phone number. The "host" part uniquely identifies the network device; these are like the last digits of your phone number.

Take a look at this IP address which you might have seen before since it's a common IP address on local area networks:

192.168.1.1

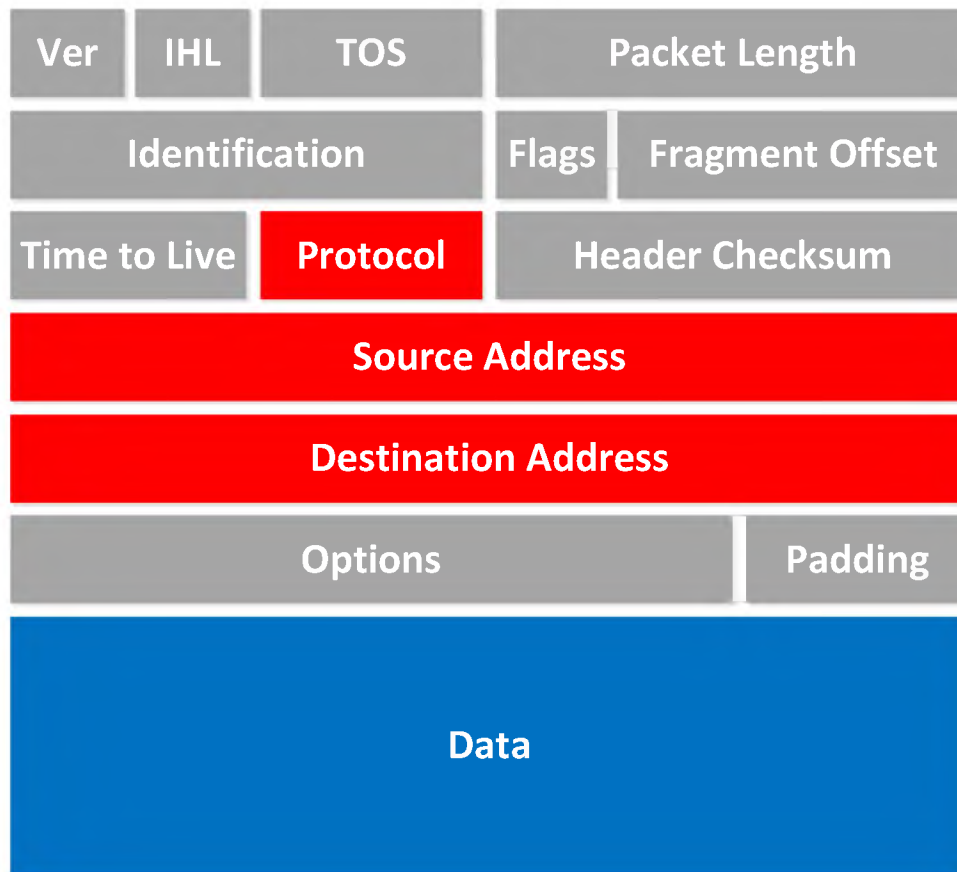
For this IP address the first 3 bytes are the "network" address and the last byte is the "host" address:



Ok awesome...but why are the first 3 bytes the "network" part and why is the last byte the "host" part? Good question! I only gave you the IP address but you might remember that if you configure an IP address you also have to specify the subnet mask. Our IP address 192.168.1.1 would come along with the subnet mask 255.255.255.0.

The subnet mask tells your computer which part is the "network" part and which part is the "host" part. Despite the name it does not "hide" or "mask" anything. We'll talk about binary and subnetting calculations later on, for now just hold the thought that your subnet mask tells us which part of the IP address is the "network" part and which part is for "hosts".

Let's take a look at an actual IP packet:



There are a lot of fields there! Now don't go look over them and feel puzzled that you have no idea what they are about. For now there are only a few fields that are interesting to us. The fields we don't care about are in gray, I want to focus on the red and blue fields.

- Protocol: Here you will find which protocol we are using on top of IP, this is how we specify which **transport layer** protocol we are using. So you'll find TCP, UDP or perhaps something else in here.
- Source Address: Here you will find the IP address of the device that created this IP packet.
- Destination Address: This is the IP address of the device that should receive the IP packet.
- Data: this is the actual data that we are trying to get to the other side.

That wasn't so bad right? No need to worry about the other fields for your CCNA. Let me show you the screenshot of Wireshark from a few pages ago again:

```

▼ Internet Protocol, Src: 192.168.69.2 (192.168.69.2), Dst: 192.168.69.1 (192.168.69.1)
  Version: 4
  Header length: 20 bytes
  ► Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 497
  Identification: 0xf5db (62939)
  ► Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  ► Header checksum: 0x37d7 [correct]
  Source: 192.168.69.2 (192.168.69.2)
  Destination: 192.168.69.1 (192.168.69.1)

```

Do you recognize all the fields? You can see it's not just theoretical stuff we are talking about...you can actually see what is going on and check out the content of an IP packet.

Let's take another look at an IP address:

192.168.1.1

What do we know about this IP address? First of all we know it's a 32-bit value, so in binary it will look like this:

**110000001010100000000100000001**

Now this is a number that is not very human-friendly so to make our life easier we can at least put this number into "blocks" of 8 bits. 8 bits is also called a byte or an octet.

**11000000**

**10101000**

**00000001**

**00000001**

Now we can convert each byte into decimal, let's take the first block and convert it from binary to decimal using the following table:

| Bits | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------|-----|----|----|----|---|---|---|---|
|      | 0   | 0  | 0  | 0  | 0 | 0 | 0 | 0 |

First block:

**11000000**

| Bits     | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|----------|-----|----|----|----|---|---|---|---|
| <b>0</b> | 1   | 1  | 0  | 0  | 0 | 0 | 0 | 0 |

$$128 + 64 = 192$$

Second block:

**10101000**

| Bits     | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|----------|-----|----|----|----|---|---|---|---|
| <b>0</b> | 1   | 0  | 1  | 0  | 1 | 0 | 0 | 0 |

$$128 + 32 + 8 = 168$$

Third block:

**00000001**

| Bits     | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|----------|-----|----|----|----|---|---|---|---|
| <b>0</b> | 0   | 0  | 0  | 0  | 0 | 0 | 0 | 1 |

Only the last bit, so that's 1.

Fourth block:

**00000001**

| Bits     | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|----------|-----|----|----|----|---|---|---|---|
| <b>0</b> | 0   | 0  | 0  | 0  | 0 | 0 | 0 | 1 |

Same as the third block, the decimal number 1.

Gives us the IP address:

**192**

**168**

**1**

**1**



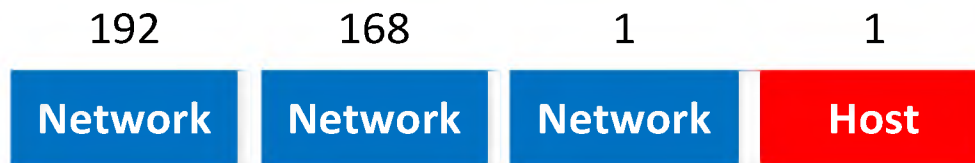
Excellent so now you know why IP addresses look like this and why we write them down like this, we even did some basic binary to decimal calculations.

One last thing to look at and that's the different classes that we have for networks. Maybe you have heard of class A,B or C networks before. Our IP address that we just used (192.168.1.1) is an example of a class C network.

We have 3 different classes to work with:

- Class A
- Class B
- Class C

So what's the difference between them? The difference between them is how many hosts you can fit in each network, let me show you an example:



The first 3 octets which are in blue are the "network" part of this IP address. The red part is for "hosts". So we can use the last octet (octet or byte is the same thing) for our hosts to give them an unique IP address.

The following computers will be in the same network:

192.168.1.1  
192.168.1.2  
192.168.1.3

As you can see their "network" part is the same.

A computer with 192.168.2.1 is not in the same network since it's "network" part is different, it's 192.168.2.X compared to 192.168.1.X.

What do you think your computer will do when it wants to send an IP packet to another network? You can find the answer on your own computer:

If you are using Windows just hit the start button, type CMD and press enter. Use the **ipconfig** command to lookup the IP information:

```
C:\Documents and Settings\Computer>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.1.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.254
```

The computer above is in network 192.168.1.X. When it wants to send something to another network it will use its **default gateway**. This will be your router; in the example above the router has IP address 192.168.1.254.

Back to our classes; let me start off by showing you the difference between the classes:



If you use a class A network you can have a LOT of hosts in each network that you create.

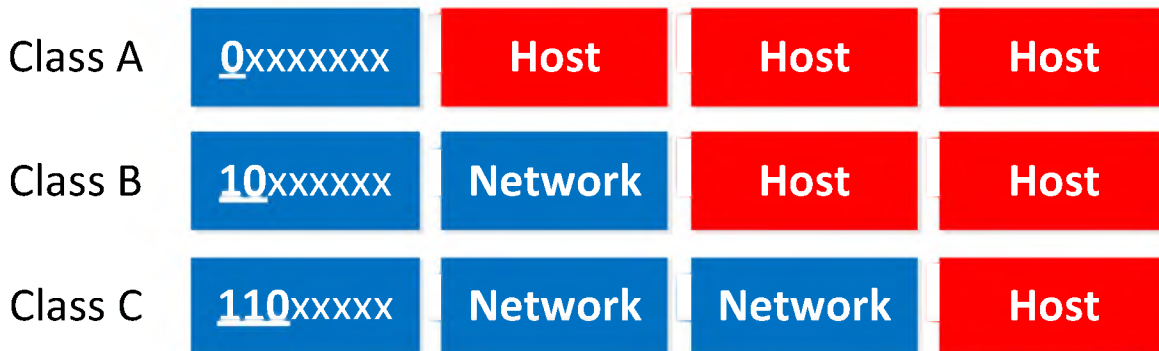


If you use a class B you can build more networks, but fewer hosts per network.



And with class C you can build a LOT of networks but only with a few hosts in each network.

I just told you 192.168.1.1 is a class C IP address. How do I know this? It's because the first bits are "fixed" for the different classes, let me show you this:



- Class A: The first bit always has to be 0.
- Class B: The first 2 bits always have to be 10.
- Class C: The first 3 bits always have to be 110.

So if you calculate this from binary to decimal you'll get the following ranges:

- Class A starts at 0.0.0.0
- Class B starts at 128.0.0.0
- Class C starts at 192.0.0.0

So what are the exact ranges that we have?

- Class A: 0.0.0.0 – 126.255.255.255
- Class B: 128.0.0.0 – 191.255.255.255
- Class C: 192.0.0.0 – 223.255.255.255

Hmm now this raises 2 questions:

- If you look closely, do you see a 127.0.0.0 subnet? It's not in the class A range so what happened to it?
- Why does Class C stop at 223.255.255.255?

To answer the first question: Go to your command prompt of your computer and type in "ping 127.0.0.1" and you'll get a response. This network range is being used as "loopback". Your loopback interface is something to check if your IP stack is OK.

To answer the second question I have to tell you that there's actually a class D range, we don't use those IP addresses to assign to computers but it's being used for "multicast". We'll get back to multicast later in the book; it starts with the 224.0.0.0 range.

The last thing I need to tell you about classes is the difference between "**private**" and "**public**" IP addresses.

- Public IP addresses are **used on the Internet**.
- Private IP addresses **are used on your local area network** and should not be used on the Internet.

These are the Private IP address ranges:

Class A: 10.0.0.0 – 10.255.255.255  
Class B: 172.16.0.0 – 172.31.255.255  
Class C: 192.168.0.0 – 192.168.255.255

Do you see our 192.168.1.1 example IP address falls within class C and is a private IP address? I like to use this IP address since it's most common to people, it's used a lot on home networks and SOHO (small office home office) routers.

Is there anything else we need to know about IP addresses? Well yes, one last thing! There are 2 IP addresses we cannot use on our network.

- **Network address.**
- **Broadcast address.**

The network address cannot be used on a computer as an IP address because it's being used to "define" the network. Routers will use the network address as you will discover later in the book.

The broadcast address cannot be used on a computer as an IP address because it's used by broadcast applications. A broadcast is an IP packet that will be received by **all devices** in your network.

So how do we recognize these two IP addresses that we cannot use? Let me give you an example for this:



Let's use the Class C range and our IP address 192.168.1.1.



We need to look at the last octet which is being used for hosts. If we set all the bits to 0 in our "host" part then we have the network address:



So 192.168.1.0 is the network address in this case and we are unable to use this IP address for computers.

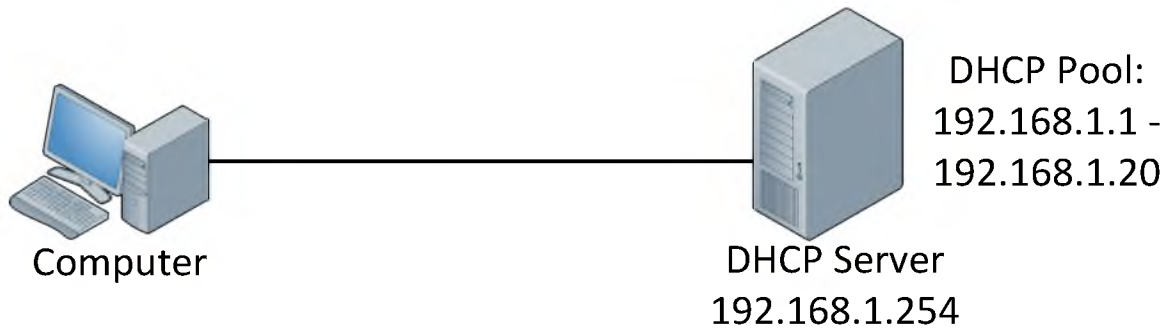
If we set all the bits to 1 we'll have a broadcast IP address and we also cannot use this for computers.



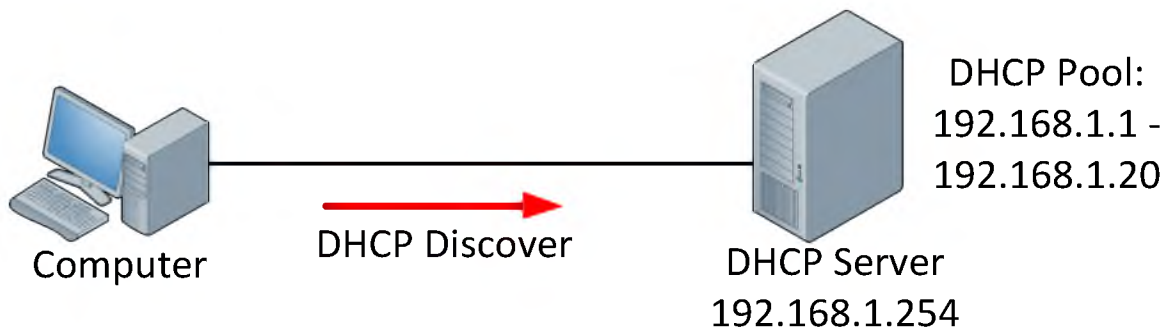
So in summary:

- Set all the host bits to 0 gives you the network address.
- Set all the host bits to 1 gives you the broadcast address.
- These 2 IP addresses we cannot use for computers.

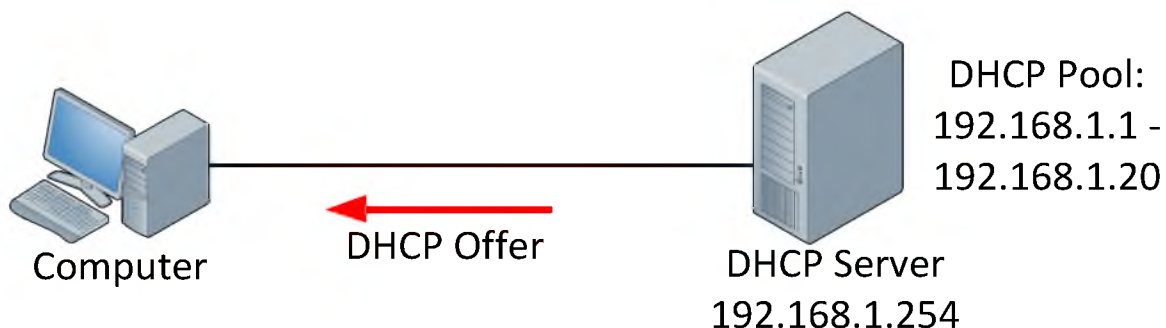
IP addresses can be configured **statically** or **dynamically**. If you go the static way you have to configure the IP address yourself on your computer, router or switch. Dynamic means we **use DHCP (Dynamic Host Configuration Protocol)**. DHCP is a server process that assigns IP addresses from a "pool" to network devices. A cisco router can be used as a DHCP server but you will also see this often on Microsoft or Linux servers. Here's how it works:



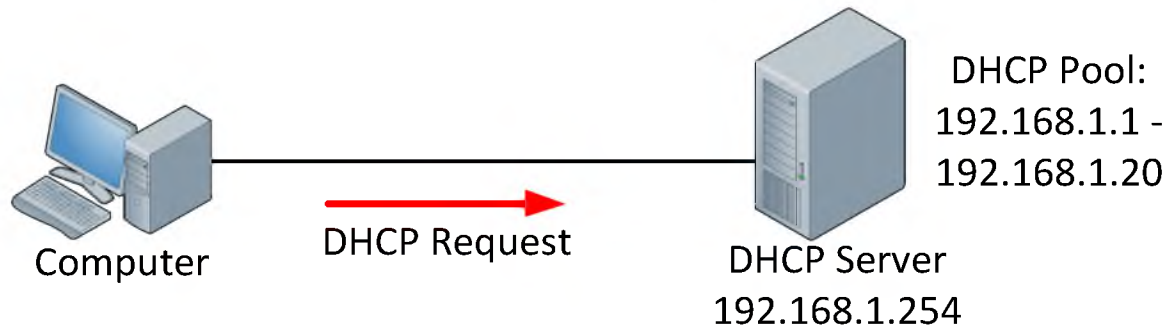
On the left side we see a computer without an IP address, on the right side is a DHCP server with IP address 192.168.1.254. A DHCP pool has been configured with IP address 192.168.1.1 – 192.168.1.20. Once the computer boots it will request an IP address by broadcasting a **DHCP discover** message:



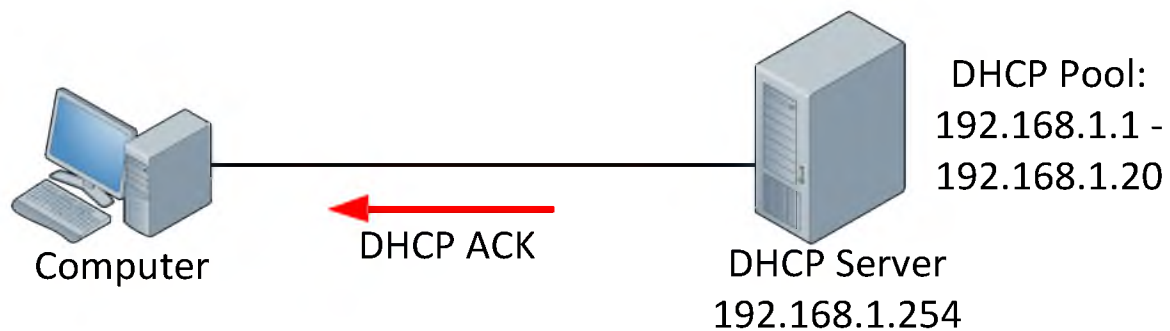
The computer has no IP address so it will broadcast this DHCP discover message. The DHCP server will hear this message and respond as following:



The DHCP server will send a **DHCP offer** message which contains the IP address that the computer can use. Besides giving an IP address we can also supply a **default gateway**, a **DNS server IP address** and some other options. We are not done now...there are two more steps:

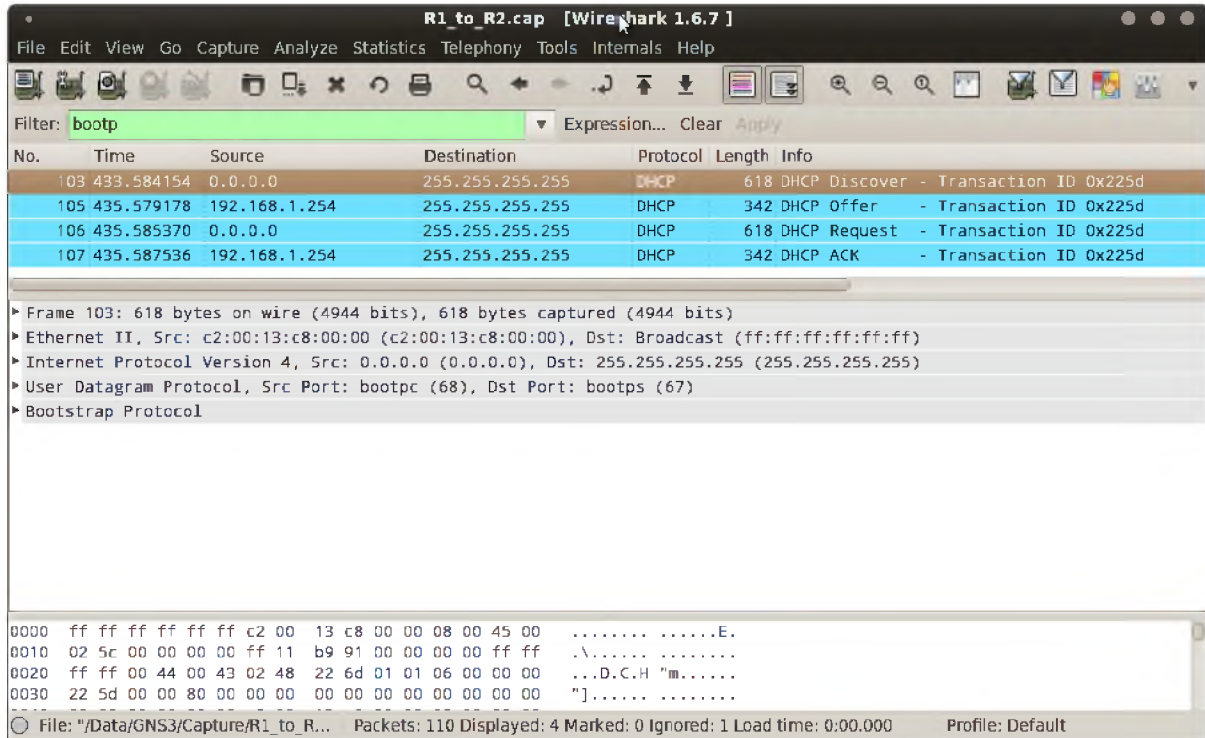


After receiving the DHCP offer our computer will send a **DHCP request** to ask if it's OK to use this information...



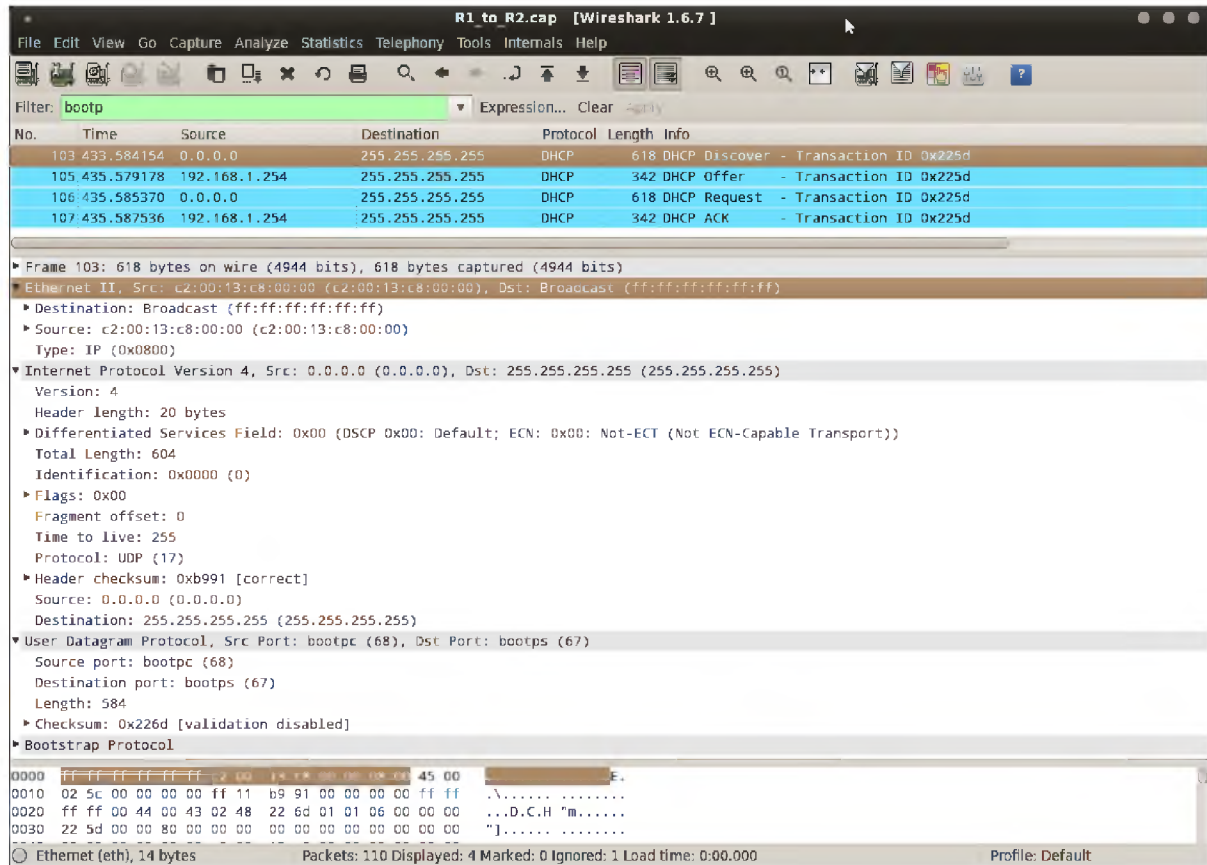
And the final step in this process will be a **DHCP ACK** from the DHCP server to "acknowledge" the request from the computer.

Here's what it looks like in wireshark:



Above you see the DHCP Discover, Offer, Request and ACK messages.





**R1\_to\_R2.cap [Wireshark 1.6.7]**

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: **bootp** Expression... Clear

| No. | Time       | Source        | Destination     | Protocol | Length | Info                                  |
|-----|------------|---------------|-----------------|----------|--------|---------------------------------------|
| 103 | 433.584154 | 0.0.0.0       | 255.255.255.255 | DHCP     | 618    | DHCP Discover - Transaction ID 0x225d |
| 105 | 435.579178 | 192.168.1.254 | 255.255.255.255 | DHCP     | 342    | DHCP Offer - Transaction ID 0x225d    |
| 106 | 435.585370 | 0.0.0.0       | 255.255.255.255 | DHCP     | 618    | DHCP Request - Transaction ID 0x225d  |
| 107 | 435.587536 | 192.168.1.254 | 255.255.255.255 | DHCP     | 342    | DHCP ACK - Transaction ID 0x225d      |

▶ Frame 105: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits)

▶ Ethernet II, Src: c2:01:13:c8:00:00 (c2:01:13:c8:00:00), Dst: Broadcast (ff:ff:ff:ff:ff:ff)

▶ Internet Protocol Version 4, Src: 192.168.1.254 (192.168.1.254), Dst: 255.255.255.255 (255.255.255.255)

▶ User Datagram Protocol, Src Port: bootps (67), Dst Port: bootpc (68)

▼ Bootstrap Protocol

Message type: Boot Reply (2)  
 Hardware type: Ethernet  
 Hardware address length: 6  
 Hops: 0  
 Transaction ID: 0x0000225d  
 Seconds elapsed: 0

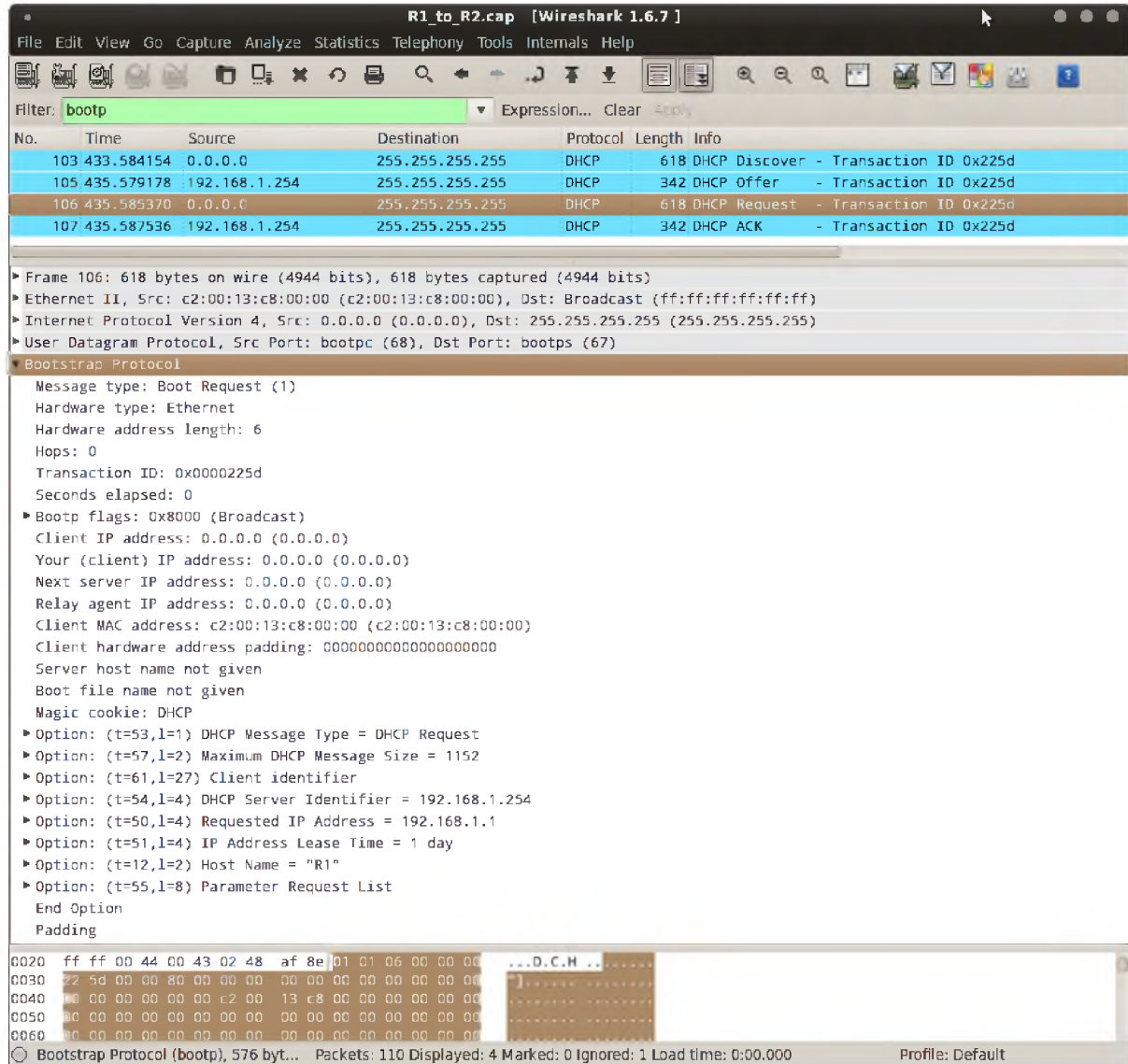
▶ Bootp flags: 0x8000 (Broadcast)  
 Client IP address: 0.0.0.0 (0.0.0.0)  
 Your (client) IP address: 192.168.1.1 (192.168.1.1)  
 Next server IP address: 0.0.0.0 (0.0.0.0)  
 Relay agent IP address: 0.0.0.0 (0.0.0.0)  
 Client MAC address: c2:00:13:c8:00:00 (c2:00:13:c8:00:00)  
 Client hardware address padding: 00000000000000000000  
 Server host name not given  
 Boot file name not given  
 Magic cookie: DHCP

▶ Option: (t=53,l=1) DHCP Message Type = DHCP Offer  
 ▶ Option: (t=54,l=4) DHCP Server Identifier = 192.168.1.254  
 ▶ Option: (t=51,l=4) IP Address Lease Time = 1 day  
 ▶ Option: (t=58,l=4) Renewal Time Value = 12 hours  
 ▶ Option: (t=59,l=4) Rebinding Time Value = 21 hours  
 ▶ Option: (t=1,l=4) Subnet Mask = 255.255.255.0  
 End Option  
 Padding

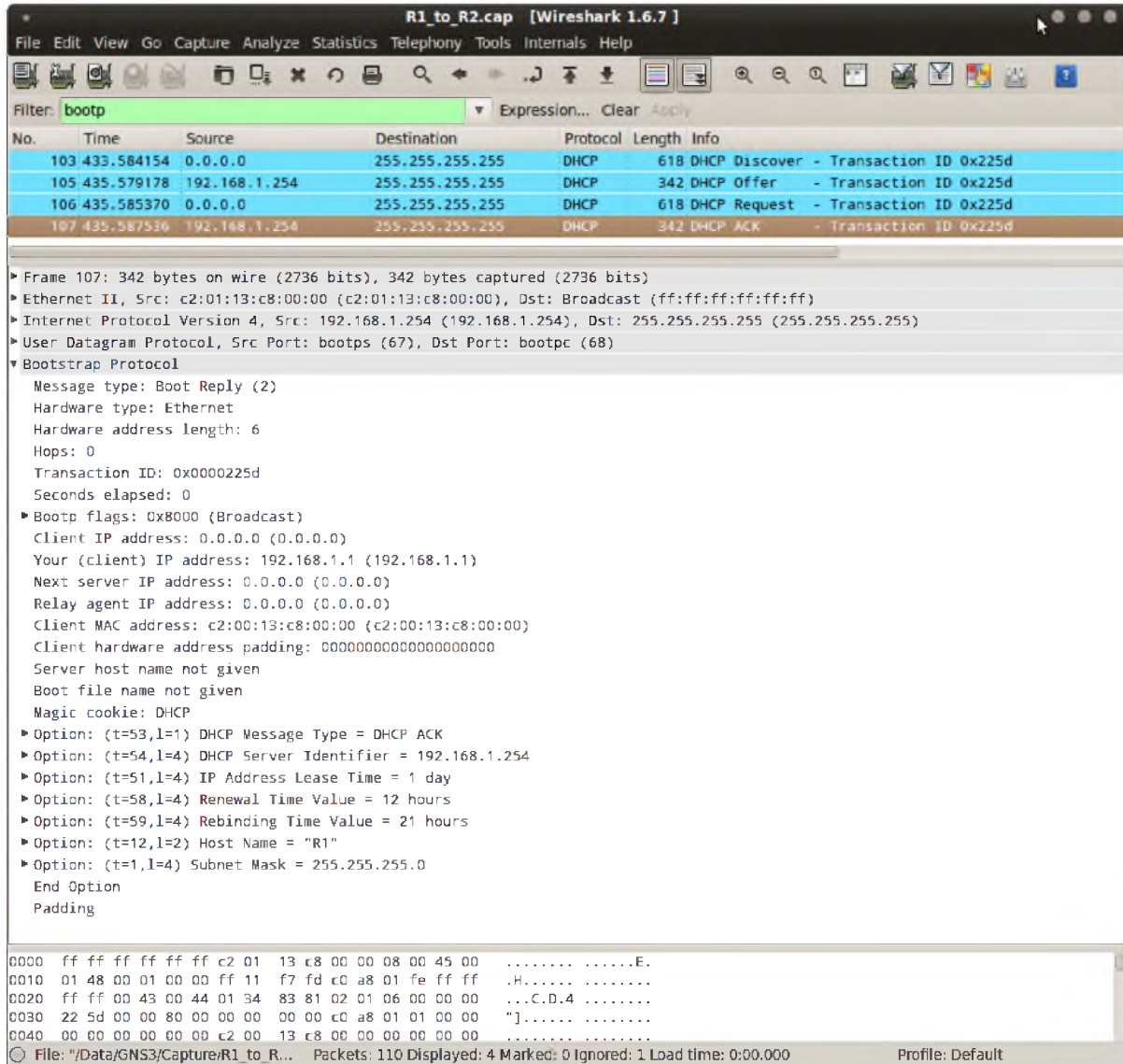
0020 ff ff 00 43 00 44 01 34 b9 df 02 01 06 00 00 00 ...C.D.4..  
 0030 72 5d 00 00 80 00 00 00 00 00 c0 a8 01 01 00 00  
 0040 00 00 00 00 00 00 00 c2 00 13 c8 00 00 00 00 00  
 0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
 0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Bootstrap Protocol (bootp), 300 bytes... Packets: 110 Displayed: 4 Marked: 0 Ignored: 1 Load time: 0:00.000 Profile: Default

The DHCP server will respond with the DHCP offer message. You can see this because the source IP address is 192.168.1.254 (the DHCP server) and when we look at the packet you can see that it is giving IP address 192.168.1.1 to the computer.



The computer will respond with a DHCP request to ask if it's ok to use this information...



And last but not least, here's the DHCP ACK telling the computer it's ok to use the information. That's all I wanted to show you about DHCP for now.

And that's the end of this chapter; you should now have a basic understanding of IP. In the "Binary, Subnetting and Summarization" chapter we will dive deeper into IP and in the "IP Routing" chapter we will look at routers and how they "route" IP packets.

## 5. The Transport Layer: TCP and UDP

Let's work our way up the OSI-model, we just covered IP and now it's time to pick a "transport" protocol. Keep in mind IP is "nothing more" but a number (ok that's very simplistic) but I want to make sure you understand we need a transport protocol for actually setting up the connection and sending data between our computers.

In this chapter I want to focus on the transport protocols that are used most of the time:

- **TCP (Transmission Control Protocol)**
- **UDP (User Datagram Protocol)**

So why do we have 2 different transport protocols here, why do we care and when do we need one over another?

The short answer is:

- TCP is a **reliable** protocol.
- UDP is a **unreliable** or **best-effort** protocol.

Unreliable you might think? Why do I want data transport which is unreliable? Does that make any sense? Let me tell you a little story to explain the difference between the two protocols.

You are sitting behind your computer and downloading the latest greatest movie in 1080P HD with 7.1 surround super sound directly from Universal studio's brand new "download on demand" service (hey you never know...it might happen one day...). This file is 20GB and after downloading 10GB there's something going wrong and a couple of IP packets don't make it to your computer, as soon as the entire download is done you try to play the movie and you get all kind of errors. Unable to watch the movie you are frustrated and head for the local dvd rental place to watch some low-quality movie...

Ok maybe I exaggerate a bit but I think you get the idea; you want to make sure the transport of your download to your computer is **reliable** which is why we use TCP. In case some of the IP packets don't make it to your computer you want to make sure this data will be retransmitted to your computer!

In our second story you are the network engineer for a major company and you just told your boss how awesome this brand new open source Voice over IP solution is. You decide to implement this new VoIP solution and to get rid of all the analog phones but your users are now complaining big time that their phone call quality is horrible. You contact the open source VoIP solution provider and you find out that they thought it would be a good idea to use a **reliable** transport protocol like TCP since well, we want phone calls to be reliable right?

Wrong thinking! TCP does error correction which means that data that didn't make it to your computer will be retransmitted. How weird will your phone call sound if you are talking to someone and you hear something that they said a few seconds ago? It's real-time so we don't want retransmission. It's better to send VoIP packets and lose a few than retransmitting them afterwards, your VoIP codec can also fix packet loss up to a certain degree. In this example we'll want to use a **best effort** or **unreliable** protocol which is UDP.

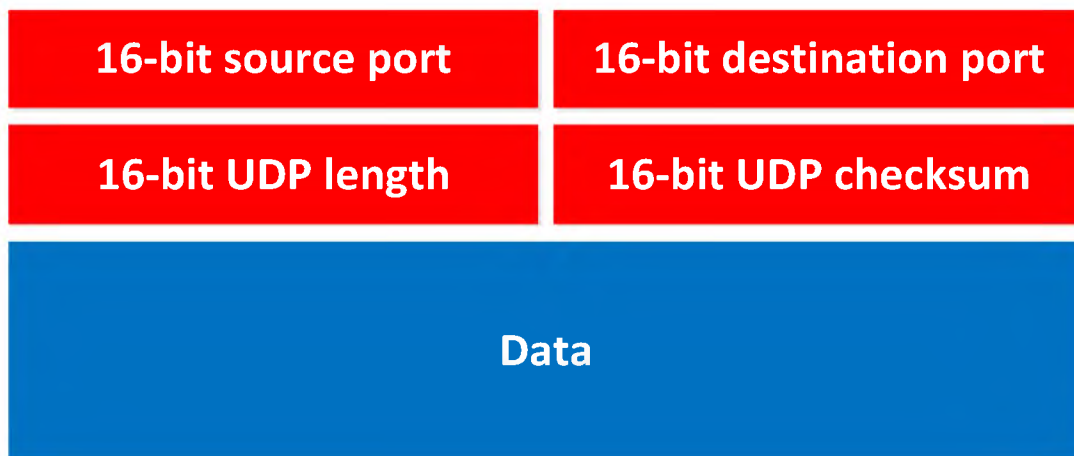
|                         | TCP                                   | UDP                       |
|-------------------------|---------------------------------------|---------------------------|
| <b>Connection Type:</b> | Connection-oriented                   | Connectionless            |
| <b>Sequencing:</b>      | Yes                                   | No                        |
| <b>Usage:</b>           | Downloads<br>File Sharing<br>Printing | VoIP<br>Video (streaming) |

What do we have in the table above? First of all you see "connection type". TCP is **connection-oriented** which means it will **"setup" a connection** and then start transferring data. UDP is connectionless which means it will just start sending and doesn't care if it arrives yes or not. The connection that TCP will setup is called the **"3 way handshake"** which I will show you in a minute.

Sequencing means that we use a **sequence number**, if you download a big file you need to make sure that you can put all those packets back in the right order. As you can see UDP does not offer this feature, there's no sequence number there.

So what about VoIP? Don't we need to put those packets back in order at the receiver side? Well actually yes we do otherwise we get some strange conversations. UDP does not offer this "sequencing" feature though...let me tell you a little secret: for VoIP it's not just UDP that we use but we also use RTP which does offer sequencing! (And some other cool features we need for VoIP).

Let's take a look at an UDP header:



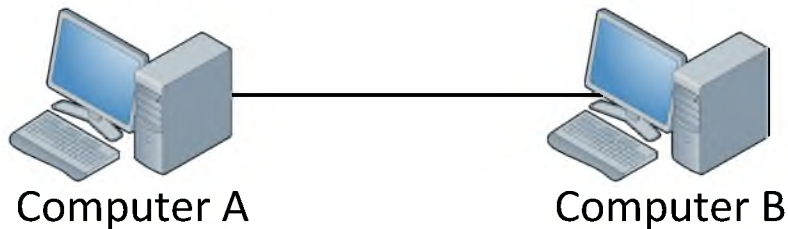
You can see how simple it is, it has the source and destination port number (this is how we know for which application the data is meant), there's a checksum and the length.

Let's sum up what we now know about UDP:

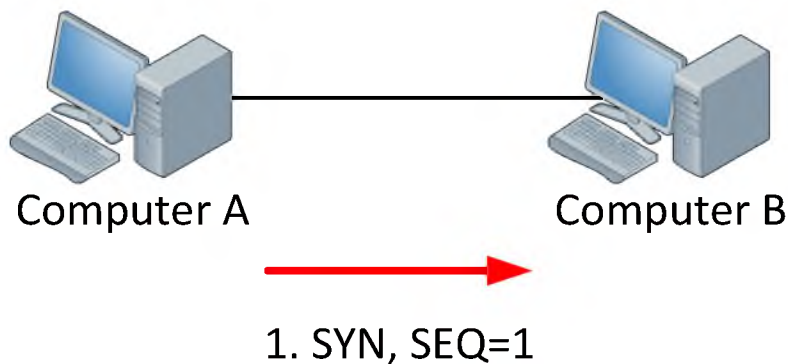
- It operates on the transport layer of the OSI model.
- Is a connectionless protocol, does not setup a connection...just sends data.
- Limited error correction because we have a checksum.
- Best-effort or unreliable protocol.
- No data-recovery features.



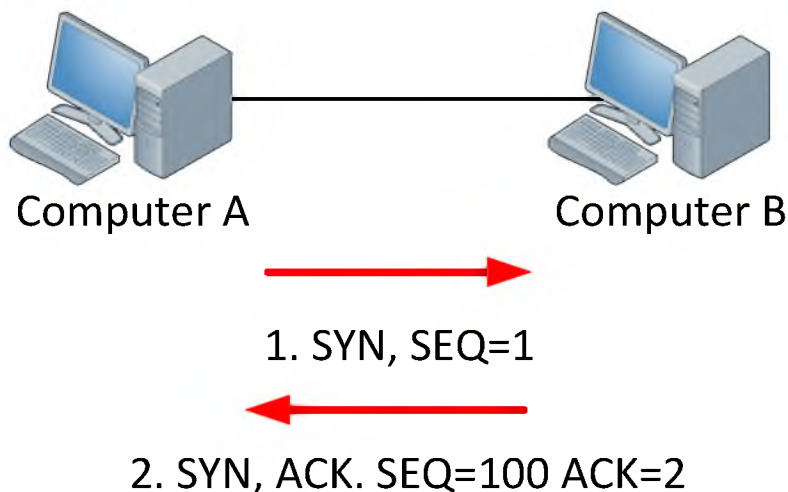
Now let's see what TCP can offer us. First of all since TCP is a reliable protocol it will "setup" a connection before we start sending any data. This connection is called the "**3 way handshake**".



Computer A wants to send data to computer B in a reliable way, so we are going to use TCP to accomplish this. First we will setup the connection by using a 3-way handshake, let me walk you through the process:



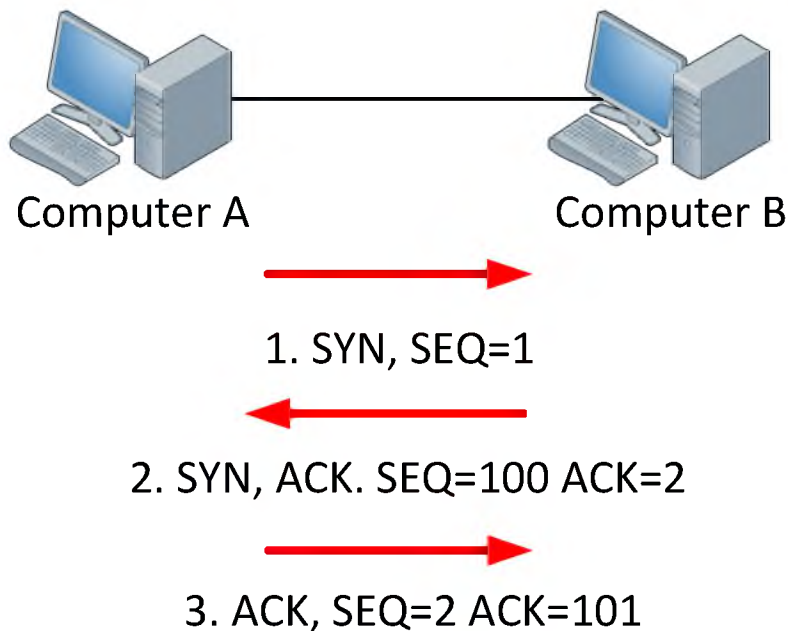
First our computer A will send a **TCP SYN**, telling computer B that it wants to setup a connection. There's also a sequence number and to keep things simple I picked number 1.



Computer B will respond to computer A by sending a **SYN,ACK** message back. You can see it picks its own sequence number 100 (I just picked a random number) and it sends ACK=2.



ACK=2 means that it acknowledges that it has received the TCP SYN from computer A which had sequence number 1 and that it is ready for the next message with sequence number 2.



The last step is that computer A will send an **acknowledgement** towards computer B in response of the SYN that computer B sent towards computer A. You can see it sends ACK=101 which means it acknowledges the SEQ=100 from computer B. Since computer B sent a ACK=2 towards computer A, computer A now knows it can send the next message with sequence number 2.

To simplify things a little bit, it looks like this:

- Computer A sends a **TCP SYN**. (I want to talk to you)
- Computer B sends a **TCP SYN,ACK**. (I accept that you want to talk to me, and I want to talk to you as well)
- Computer A sends a **TCP ACK**. ( I accept that you want to talk to me)

Let me show you an example in Wireshark what this looks like on a real network:

|            |                 |                 |     |                                     |
|------------|-----------------|-----------------|-----|-------------------------------------|
| 1 0.000000 | 192.168.1.2     | 174.143.213.184 | TCP | 54841 > http [SYN] Seq=0 Win=5840 L |
| 2 0.046770 | 174.143.213.184 | 192.168.1.2     | TCP | http > 54841 [SYN, ACK] Seq=0 Ack=1 |
| 3 0.046803 | 192.168.1.2     | 174.143.213.184 | TCP | 54841 > http [ACK] Seq=1 Ack=1 Win= |

In this example computer with IP address 192.168.1.2 wants to setup a connection with 174.143.213.184 and it's sending a TCP SYN.

174.143.213.184 is responding by sending a TCP SYN,ACK in return.

Finally 192.168.1.2 sends a TCP ACK to finish the 3 way handshake.

Let's see those packets in detail, first we look at the TCP SYN:

```
▼ Transmission Control Protocol, Src Port: 54841 (54841), Dst Port: http (80), Seq: 0, Len: 0
  Source port: 54841 (54841)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 0      (relative sequence number)
  Header length: 40 bytes
  ► Flags: 0x02 (SYN)
    Window size: 5840
  ► Checksum: 0x85f0 [validation disabled]
  ► Options: (20 bytes)
```

You can see in the "Flags" section that the SYN-bit has been set. On the top right you can see "Seq: 0" which is the sequence number.

```
▼ Transmission Control Protocol, Src Port: http (80), Dst Port: 54841 (54841), Seq: 0, Ack: 1, Len: 0
  Source port: http (80)
  Destination port: 54841 (54841)
  [Stream index: 0]
  Sequence number: 0      (relative sequence number)
  Acknowledgement number: 1  (relative ack number)
  Header length: 40 bytes
  ► Flags: 0x12 (SYN, ACK)
```

In this example you see that in the "Flags" section both the SYN and ACK bit are set, also on the top you can see "Seq:0" and "Ack:1". This computer is acknowledging the SYN-bit from the other computer.

```
▼ Transmission Control Protocol, Src Port: 54841 (54841), Dst Port: http (80), Seq: 1, Ack: 1, Len: 0
  Source port: 54841 (54841)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 1      (relative sequence number)
  Acknowledgement number: 1  (relative ack number)
  Header length: 32 bytes
  ► Flags: 0x10 (ACK)
    Window size: 5888 (scaled)
  ► Checksum: 0x9529 [validation disabled]
  ► Options: (12 bytes)
  ► [SEQ/ACK analysis]
```

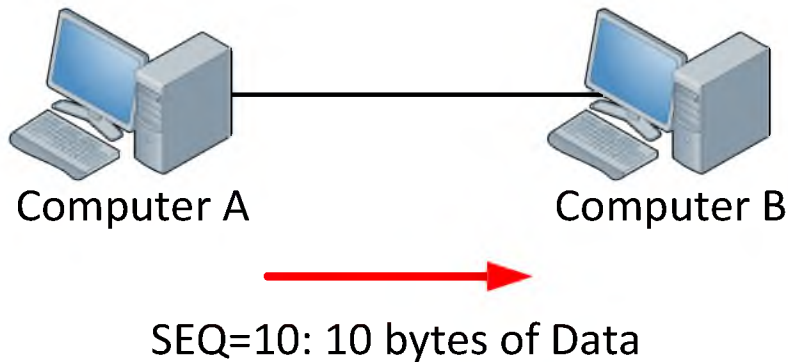
This is the final step in the process where our computer that that started the 3 way handshake sets the ACK-bit and acknowledges the SYN from the other side.

Are you following me so far? If you want to play a bit just start up Wireshark and see if you can capture a 3 way handshake yourself on your computer. Take a look at the different TCP packets and see if you can find the SYN, SYN-ACK and ACK's. Also check the different sequence numbers and see if you can find a pattern.

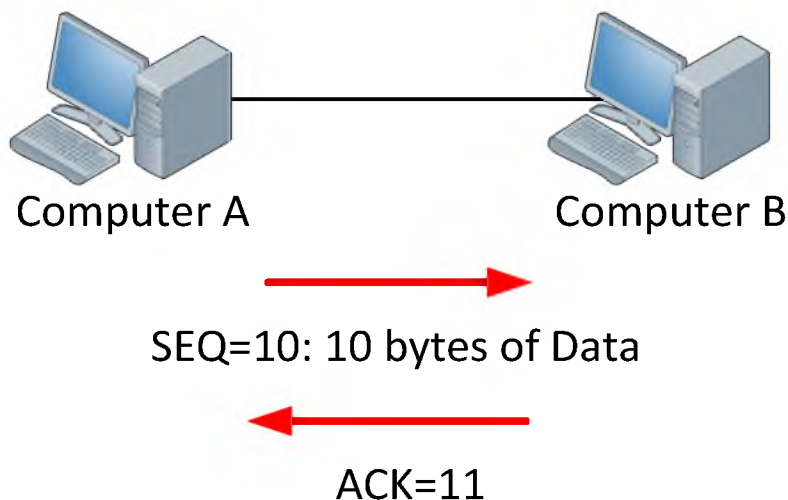
Phew so we have setup a connection using the 3 way handshake! Now we can start sending data...what else does TCP offer us? One of the things is **"flow control"**.

Imagine you have a fast computer transmitting data to a smartphone, obviously the computer could overburden the smartphone with traffic which is why we have flow control. In each TCP segment the receiver can specify in the "receive window" field how much data in bytes it wants to receive.

Our sending computer can only send data up to this size so the smartphone doesn't get overburdened. The more data you can send each time the higher your throughput will be. Let's look at an example of how this all fits together:



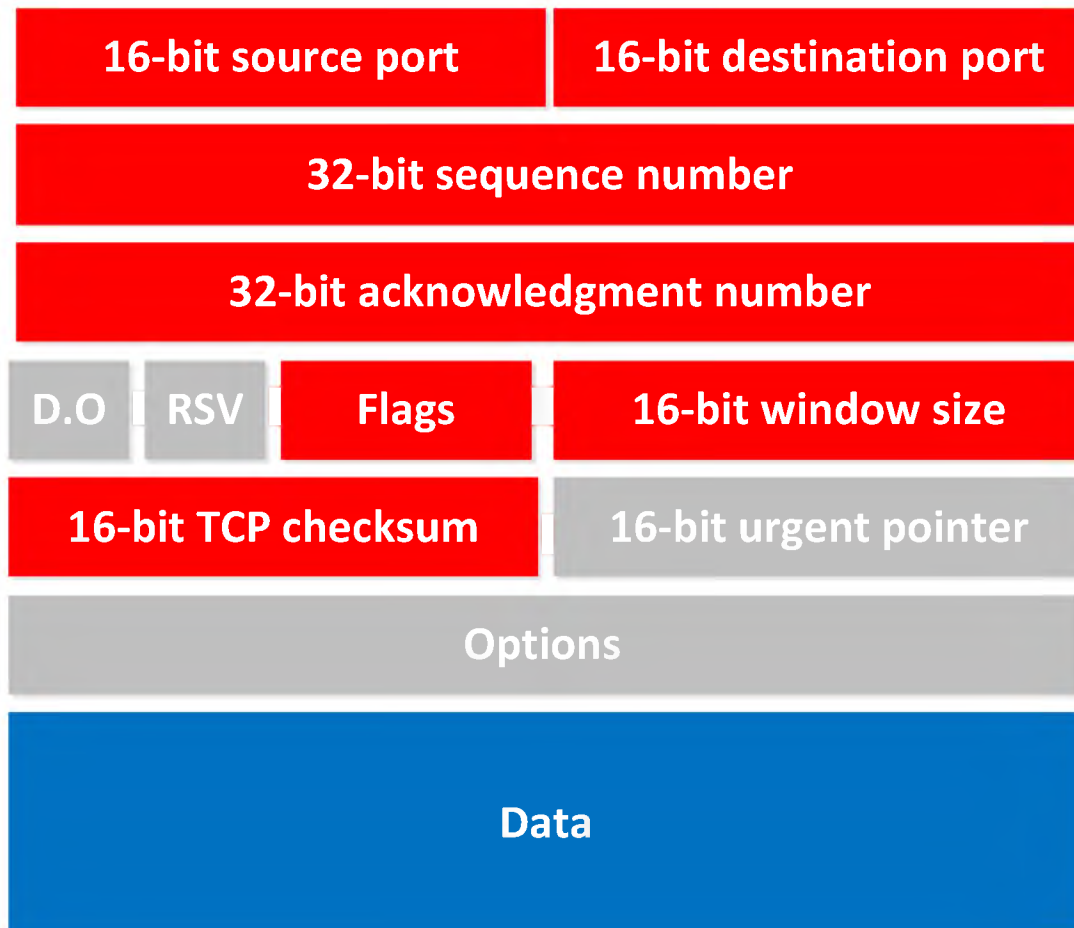
Computer A has setup a connection with Computer B by using the 3 way handshake. We are sending 10 bytes of Data which means our "window size" is 10 bytes. The sequence number is 10.



Computer B is going to respond by sending "ACK=11" which means "thanks I received your 10 bytes, now send me #11 and the rest". TCP is a reliable protocol which is why we have to acknowledge everything we are receiving.

The larger your window size, the higher your throughput will be. This makes sense because you are sending fewer ACK's compared to the data you are sending.

TCP is a fairly complex protocol and if we look at the header you'll see it has a lot more fields than UDP has:



The fields in gray are not important for us; everything in red is what I would like to tell you about.

As you can see there's a 16-bit source and destination port, **port numbers are used to determine for which application** this data is meant (This is how we go from the transport layer up to the higher layers in the OSI-model).

You can see we have 32-bits that are used for our sequence numbers, and there's also 32-bits for the acknowledgment (ACK) reserved.

The "Flags" field is where TCP sets the different message types like "SYN" or "ACK".

Window size has a 16-bit field which specifies how many bytes of data you will send before you want an acknowledgment from the other side.

Finally there's a checksum and of course our data, the stuff we are actually trying to send to the other side.

Let's sum up what we have learned about TCP:

- It's a reliable protocol.
- Before you send data you will setup the connection by using the 3 way handshake.
- After sending X amount of bytes you will receive an acknowledgment (ACK) from the other side.
- How many bytes you send before you get an ACK is controlled by using the "window size".
- TCP can do retransmissions.

That's the end of this chapter. If you want to see TCP in action the best way to do it by using Wireshark and capturing some traffic of your computer while you are browsing the Internet. See if you can track the sequence numbers, 3 way handshake etc.

## 6. Ethernet: Dominating your LAN for over 30 years

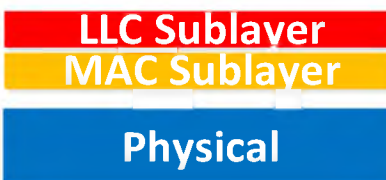
The title of this chapter might sound like something from a movie but in a sense it's true. On our Local area networks (LAN) we basically only run Ethernet, there's nothing else that we do. So let's talk a bit about Ethernet and LANS.

What is a LAN anyway? The term is a bit vague but roughly you can say that a network which is in a **single building or perhaps a campus area with multiple buildings is what we call "local" area network or LAN**. If you would have a connection to an ISP or perhaps a leased line to connect your headquarters network to a branch office, that's where we talk about a **WAN (Wide area network)**. LAN doesn't have anything to do with size, so a network with 2 computers is just as good a LAN as having 2,000 computers in a building.

Ethernet is the protocol that we are running on our LAN. So what layer(s) of the OSI model do you think Ethernet will describe? If you are thinking "Data link" layer you got it right but it also describes the physical layer.



Now here things will get a bit funky, Ethernet describes the Data link layer but it has been split up in two pieces, so it looks like this:



So there are sublayers called "**LLC**" which stands for **Logical Link Control** and "**MAC**" which stands for "**Media Access Control**". You have probably seen or heard about MAC addresses before.

The logical link control layer does a couple of things like error correction. We don't care about this as much nowadays because we use TCP which does error correction on the transport layer. Keep in mind that Ethernet was invented a long time ago and we used to have a lot of other network protocols besides IP like IPX, AppleTalk, Novell etc.

The MAC sublayer is more interesting to us; let me describe its functions and why we need it. First of all every device on our LAN has a unique identifier on the data link layer, this is our "MAC address". Just as an IP address is a unique identifier on the network layer (layer 3) we have the MAC address as a unique identifier on the data link layer (layer 2).

One of the other things that our MAC sublayer does is taking care of channel access. This makes it possible so computers connected to the same physical medium can access and share it. What do I mean by "same physical medium"? We have to take a little history lesson here.



Do you remember those network cables? If you don't...good for you! I have to be honest I never worked with these networks on a "professional" level but I did use them for home networks at the time (of course to play games over the LAN...not to build websites about networking like I do nowadays...☺). All computers in the network were connected to a single long black coax cable (our physical medium) and were sharing the network. A network like this was **half-duplex** which means that **only 1 computer was able to send traffic and the others had to wait**. Nowadays we have **full-duplex** which means all devices can send and receive at the same time! Remember the first chapter where I talked about bus, ring and star topologies? This is our bus topology right here! What do you think would happen if two computers would start sending data at the exact same moment?

That's right...you get a **collision**! Electrical signals bouncing into each other and no data transmission at all...

Maybe you also remember our old friend the "Hub":



Courtesy of Netgear Inc. Unauthorized use not permitted

That's right, that's about the first star topology network we had. The problem with our hub is that it's nothing more but an **electrical repeater**. If you use a hub for your network, it's running half-duplex which means you can get collisions as well!



*A hub is not the same as a switch, and there's no such thing as a "hub switch". More about this in the "Hubs, Bridges and Switches" chapter!*

Back to our MAC sublayer, if you are running a half-duplex network we need to make sure that whenever there's a collision on the network we have a solution. There is one and this protocol is called **CSMA/CD**.

CS = Carrier Sense

MA = Multi Access

CD = Collision Detection

Carrier sense means we can "listen" on the cable to hear if anything is going on, in other words if another computer is sending data at this moment. Multi access means everyone can access our physical medium but it has to be clear...no other computer should be sending at that moment.

In case 2 computers send at the same time we have a collision, since we can detect this (its carrier sense right) CSMA/CD will solve this as following:

1. The two computers that had the collision will start jamming the physical medium; this will ensure nobody else can transmit at that moment.
2. The two computers each start a random clock.
3. When the time of the random clock elapses they retransmit.

Since the clock is random, both computers will have a different timer and one of them will send its data before the other. By jamming the physical medium we will be certain that no other computer will get a chance to send data before them.



Enough about the MAC sublayer. Let me give you an example of an Ethernet Frame:



The most important fields for us are "Dest" which stands for **destination** and the **source address**; this is where the **MAC addresses** fit in.

Just for fun let me describe the other fields a bit. Preamble and SOF "Start of Frame delimiter" are a string of alternating 0's and 1's to tell the receiver that an Ethernet frame is incoming. Length is of course the size of our Ethernet Frame, 802.2 Header/Data is where the LLC sublayer or data fits in. At the end you'll find a FCS (Frame Check Sequence) to see if the frame is OK or corrupted.

So what does a MAC address look like? Let's have a look:



A MAC address is 48-bits and consists of a couple of fields:

1. BC which stands for **broadcast**; If your Ethernet frame is a broadcast than you have to set this bit to 1.
2. Local: this bit has to be set when you change your MAC address. Normally a MAC address is unique on the planet; if you change it it's only **locally unique** within your network.
3. **OUI** which stands for **Organization Unique Identifier**; every network vendor has received 22 bits that identifies them.
4. The last 24 bits are **Vendor Assigned**; the network vendor will use these bits to give each network device a unique MAC address.

We write down MAC addresses in hexadecimal so it will look like something like this:

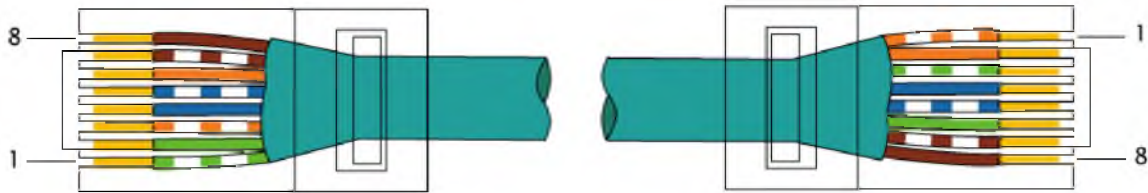
00:00:0C:52:31:04

Any idea who's MAC address this is? Take a wild guess....Cisco of course!

There's one last thing I want to show you about LAN and Ethernet, not the most exotic topic but something you need to know if you want to pass your CCNA. It's about the different cables that we have.

You have probably familiar seen UTP (Unshielded Twisted Pair) cabling but did you know we have two different types of cables?

- **Straight-through**
- **Crossover**



The plug on the left side is straight-through and the one on the right side is crossover.

The difference is how the wires are connected in the RJ-45 plug. A straight-through cable has the same wire layout on both sides. Crossover cables have the crossover wire layout on one side and the straight-through on the other side.

Why do we care about this? Nowadays it doesn't matter much which cable you use since most computers, laptops and networking hardware is **auto-sensing** (it's called Auto-MDIX) which means it will automatically detect how the wires are connected in the RJ-45 plug and it'll work.

If you are using Cisco routers or switches you need to make sure you use the correct cables though.

When and where do we need which cable?

- Hubs and switches are seen as "network devices".
- Computers, servers and routers are seen as "host devices".

Why do we call a router a host device and not a network device? Well try to think of it this way...if you don't configure a Cisco router it's not going to route anything for you and it's nothing more but a "computer". We need to enable a routing protocol ourselves...besides it will help you remember which cable you need to use. More about routing later!

- Network device <-> Host device: straight-through cable.
- Host device <-> Host device: crossover cable.
- Network device <- -> Network device: crossover cable.

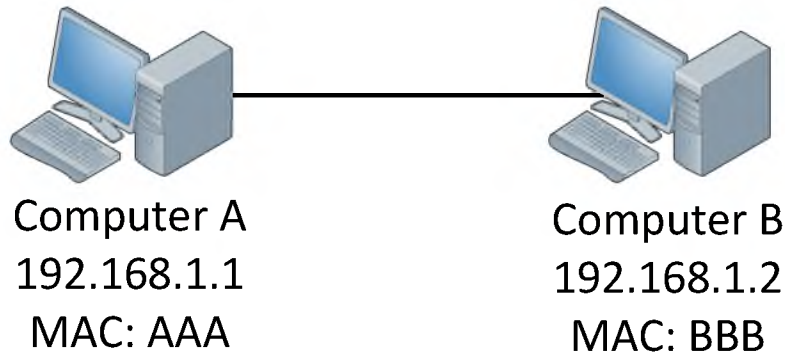
So between a computer and a switch you will use a straight-through cable. Connect 2 switches to each other and you'll need a crossover cable, the same applies if you connect to routers to each other. Router to computer (both host devices) you need a crossover cable as well.

Now we are talking about cables...do you know what the official name is for the blue Cisco console cable? You need this cable to configure your switch or router from your computer.



It's called a **rollover cable**. I didn't make up the name but this is CCNA material.

You have now learned about Ethernet, IP, TCP, UDP or in other words layer 1 up to 4 of the OSI-model. There is one more thing I'd like to explain to you:



In the picture above we have two computers, computer A and computer B and you can see their IP addresses and their MAC addresses.

We are sitting behind computer A, open up a command prompt and type:

```
C:\Users\vmware>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:
Reply from 192.168.1.2: bytes=32 time=15ms TTL=57
Reply from 192.168.1.2: bytes=32 time=15ms TTL=57
Reply from 192.168.1.2: bytes=32 time=14ms TTL=57
Reply from 192.168.1.2: bytes=32 time=17ms TTL=57

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 14ms, Maximum = 17ms, Average = 15ms
```

You know about the OSI-model and also know we have to go through all the layers.

Ping uses the **ICMP** protocol and IP uses the network layer (layer 3). Our IP packet will have a source IP address of 192.168.1.1 and a destination IP address of 192.168.1.2. Next step will be to put our IP packet in an Ethernet frame where we set our source MAC address AAA and destination MAC address BBB.

Now wait a second...how does computer A know about the MAC address of computer B? We know the IP address because we typed it but there is no way for computer A to know the MAC address of computer B. There is another protocol we have that will solve this problem for us, it's called **ARP (Address Resolution Protocol)**.

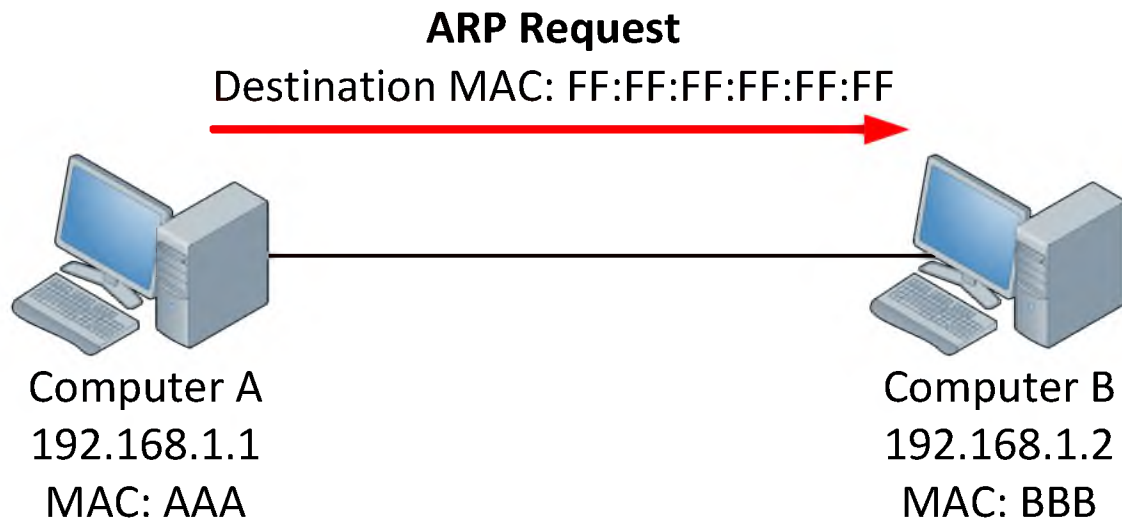
Let me show you how it works:

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

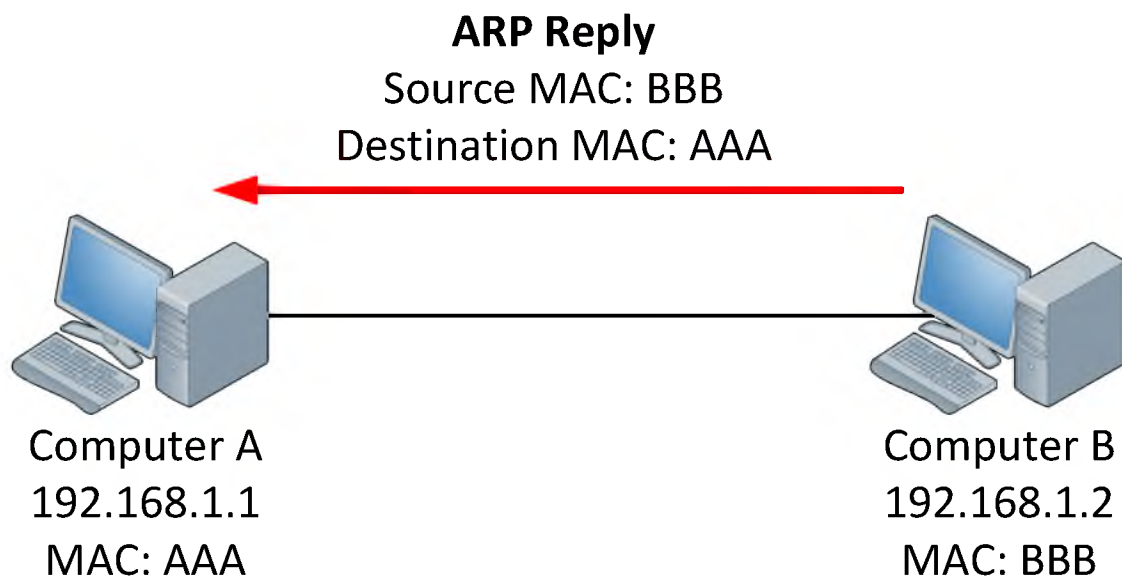
Interface: 192.168.1.1 --- 0xb
  Internet Address      Physical Address      Type
  192.168.1.2           00-0c-29-63-af-d0    dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff    static
  224.0.0.22            01-00-5e-00-00-16    static
  224.0.0.252          01-00-5e-00-00-fc    static
```

In the example above you see an example of an **ARP table** on a Computer A. As you can see there is only one entry, this computer has learned that the IP address 192.168.1.2 has been mapped to the MAC address 00:0C:29:63:AF:D0.

Let's take a more detailed look at ARP and how it functions:

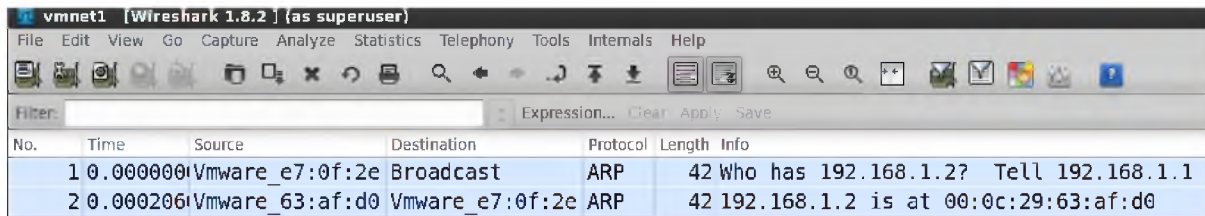


In this example we have two computers and you can see their IP address and MAC address. We are sitting behind computer A and we want to send a ping to computer B. The ARP table is empty so we have no clue what the MAC address of computer B is. The first thing that will happen is that computer A will send an **ARP Request**. This message basically says "Who has 192.168.1.2 and what is your MAC address?" Since we don't know the MAC address we will use the broadcast MAC address for the destination (FF:FF:FF:FF:FF:FF). This message will reach all computers in the network.



Computer B will reply with a message **ARP Reply** and is basically saying "that's me! And this is my MAC address". Computer A can now add the MAC address to its ARP table and start forwarding data towards computer B.

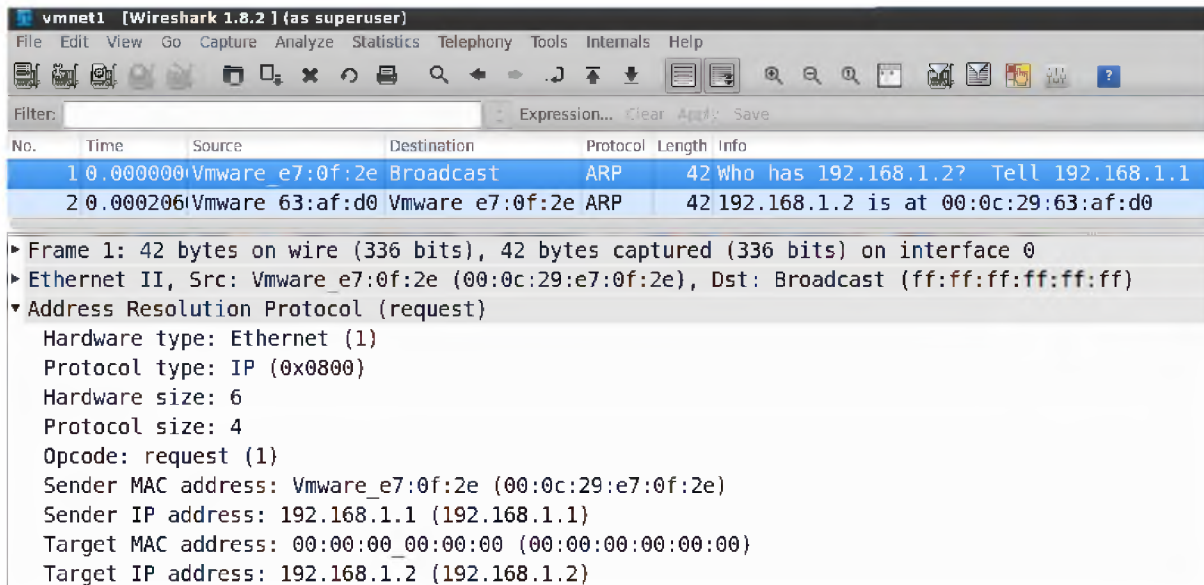
If you want to see this in action you can look at it in Wireshark:



| No. | Time     | Source          | Destination     | Protocol | Length | Info                                  |
|-----|----------|-----------------|-----------------|----------|--------|---------------------------------------|
| 1   | 0.000000 | Vmware_e7:0f:2e | Broadcast       | ARP      | 42     | Who has 192.168.1.2? Tell 192.168.1.1 |
| 2   | 0.000206 | Vmware_63:af:d0 | Vmware_e7:0f:2e | ARP      | 42     | 192.168.1.2 is at 00:0c:29:63:af:d0   |

Above you see the ARP request for Computer A that is looking for the IP address of Computer B. The source MAC address is the MAC address of computer A, the destination MAC address is "Broadcast" so it will be flooded on the network.

The second packet is the ARP reply. Computer B will send its MAC address to Computer A. Here's a detailed look:



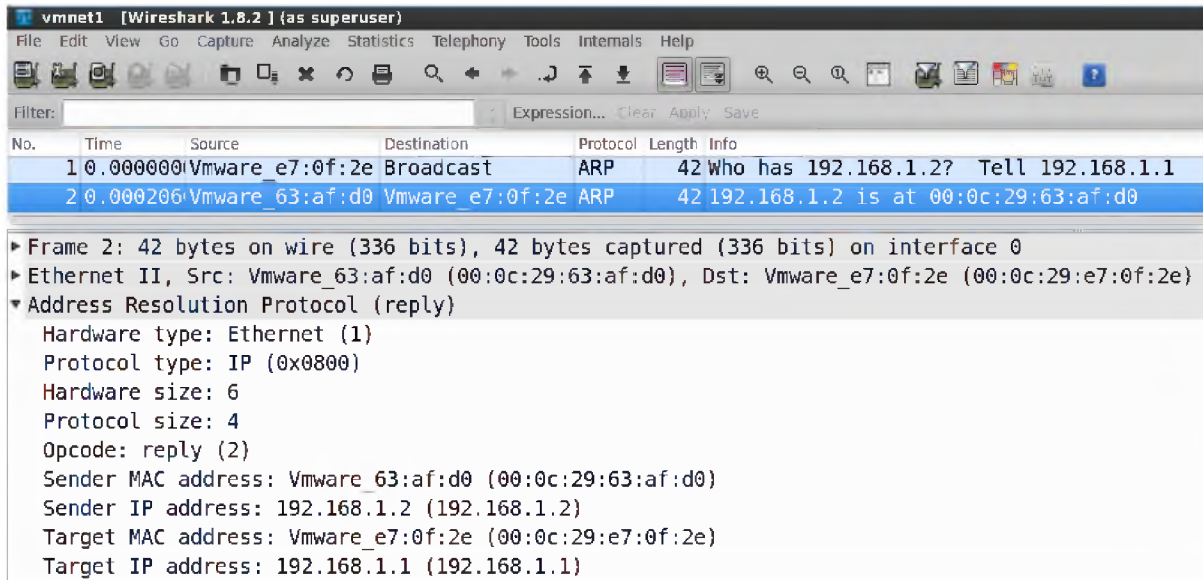
| No. | Time     | Source          | Destination     | Protocol | Length | Info                                  |
|-----|----------|-----------------|-----------------|----------|--------|---------------------------------------|
| 1   | 0.000000 | Vmware_e7:0f:2e | Broadcast       | ARP      | 42     | Who has 192.168.1.2? Tell 192.168.1.1 |
| 2   | 0.000206 | Vmware_63:af:d0 | Vmware_e7:0f:2e | ARP      | 42     | 192.168.1.2 is at 00:0c:29:63:af:d0   |

▶ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0  
 ▶ Ethernet II, Src: Vmware\_e7:0f:2e (00:0c:29:e7:0f:2e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
 ▶ Address Resolution Protocol (request)  
     Hardware type: Ethernet (1)  
     Protocol type: IP (0x0800)  
     Hardware size: 6  
     Protocol size: 4  
     Opcode: request (1)  
     Sender MAC address: Vmware\_e7:0f:2e (00:0c:29:e7:0f:2e)  
     Sender IP address: 192.168.1.1 (192.168.1.1)  
     Target MAC address: 00:00:00\_00:00:00 (00:00:00:00:00:00)  
     Target IP address: 192.168.1.2 (192.168.1.2)

Above you can see the ARP request.

And here's the ARP reply:



You can see that Computer B sends its MAC address in the ARP reply to Computer A.

Enough about ARP and Ethernet, in the next chapter we'll discuss the difference between hubs, bridges and switches.

## 7. Introduction to Cisco IOS

In this chapter I'm going to show you how Cisco IOS works and how to create a basic configuration.

Just like a computer a switch or router requires an operating system to support the hardware. Cisco IOS is the operating system that you will find on the switches and routers and some other devices like wireless access points.

When you work with Cisco routers and switches you will do most of the configuration using the **CLI (Command Line Interface)**. For some of you this might prove challenging in the beginning and it will take some time to become familiar with the CLI, however once you get used to it I promise that it's the fastest and most convenient method to configure routers or switches.

The CLI can be accessed by using the blue Cisco console cable (it's called a rollover cable) or remotely using telnet or SSH. I'll show you how to do this later in this chapter.

Cisco also offers a GUI (Graphical User Interface):

- CNA (Cisco Network Assistant) for switches.
- SDM (Security and Device Manager) or CCP (Cisco Configuration Professional) for routers.

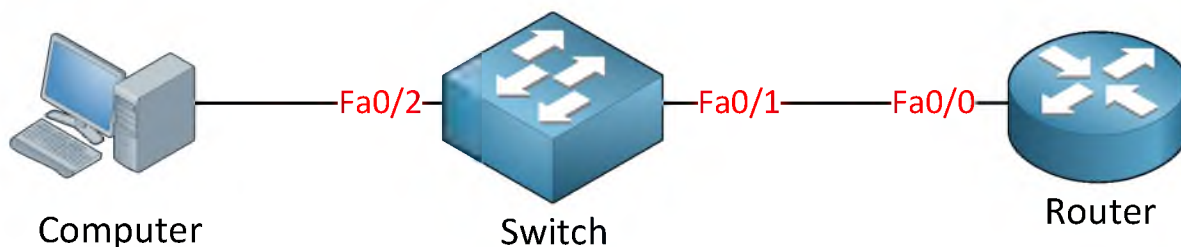
SDM was the first version of the GUI but now it has been replaced by CCP.

Since Cisco updated the CCNA exam(s) in 2013, they completely **removed SDM and CCP from the CCNA blueprint**. You will only have to work with the CLI.

The advantage of a GUI is that it has wizards that let you configure complex things with a few clicks. The downside however is that A) you might have no idea what you are doing and B) when you need to troubleshoot you'll need the CLI 9 out of 10 times. I'm not a big fan of the GUI but it's best to see for yourself.

We will start with the basic configuration of a Cisco device. First I will use a switch to demonstrate the CLI but the same commands work on a router. Secondly I will demonstrate CCP on a router.

This is the topology that I am using:





Let's take a switch out of the box and start it, see what it does shall we? I'll be using the following items:



Courtesy of Cisco Systems, Inc. Unauthorized use not permitted.

First of all we need to have a switch. I have a Cisco Catalyst 3560 that I'll use for my demonstration.



Secondly we'll need one of those Cisco console cables or we can't connect our computer to the switch.

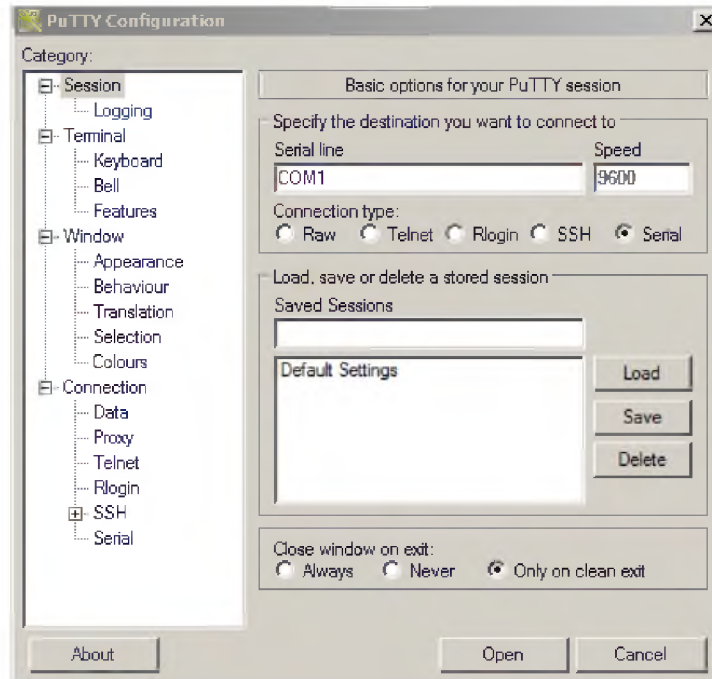


If you don't have a COM / serial port on your computer or laptop, use your USB to serial cable. The last thing you require is an application to connect to your serial port.

Putty is a good free application to start with, you can download it here:

<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

When you start putty it looks like this:



Make sure you select **serial** and type in the correct COM port number. If you don't know the COM port number you can look it up in the windows device manager. You need to leave speed at **9600**. Click on **open** and you will have access to your switch.

When you start a switch for the first time its initial configuration is enough to make it work and "switch" traffic for the computers connected to it.

As soon as you power on the switch this is what it will do:

1. Check the hardware.
2. Locate the Cisco IOS image.
3. Locate and apply configuration (if available).

This is what it looks like on a real switch:

```
Boot Sector Filesystem (bs) installed, fsid: 2
Base ethernet MAC Address: 00:11:bb:0b:36:00
Xmodem file system is available.
The password-recovery mechanism is enabled.
Initializing Flash...
flashfs[0]: 8 files, 4 directories
flashfs[0]: 0 orphaned files, 0 orphaned directories
flashfs[0]: Total bytes: 15998976
flashfs[0]: Bytes used: 10424320
flashfs[0]: Bytes available: 5574656
flashfs[0]: flashfs fsck took 9 seconds.
...done Initializing Flash.
```

Above you see that it's checking the flash drive of the switch. Next step is to load the IOS image that it found on the flash drive:

```
Loading "flash:c3560-advipservicesk9-mz.122-46.SE.bin"
```

```
Interrupt within 5 seconds to abort boot process.
```

```
Loading "flash:/c3560-advipservicesk9-mz.122-
```

```
44.SE1.bin"...@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
File "flash:/c3560-advipservicesk9-mz.122-44.SE1.bin" uncompressed and  
installed, entry point: 0x3000  
executing...
```

IOS images are stored on the flash drive in a compress format, it will be uncompressed and copied to the RAM of the switch.

Now IOS is loaded you will see something like this:

#### Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) of the Commercial Computer Software - Restricted Rights clause at FAR sec. 52.227-19 and subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS sec. 252.227-7013.

cisco Systems, Inc.  
170 West Tasman Drive  
San Jose, California 95134-1706

**Cisco IOS Software, C3560 Software (C3560-ADVIPSERVICESK9-M), Version 12.2(44)SE1, RELEASE SOFTWARE (fc1)**

Copyright (c) 1986-2008 by Cisco Systems, Inc.

Compiled Fri 07-Mar-08 00:10 by weiliu

Image text-base: 0x00003000, data-base: 0x01900000

Above you see this banner and the IOS version that I'm running. This is a Cisco 3560 switch. Next step is that IOS will check the flash drive:

```
Initializing flashfs...
```

```
flashfs[1]: 8 files, 4 directories
```

```
flashfs[1]: 0 orphaned files, 0 orphaned directories
```

```
flashfs[1]: Total bytes: 15998976
```

```
flashfs[1]: Bytes used: 10424320
```

```
flashfs[1]: Bytes available: 5574656
```

```
flashfs[1]: flashfs fsck took 10 seconds.
```

```
flashfs[1]: Initialization complete....done Initializing flashfs.
```

And once it's done it will do a POST (Power On Self-Test):

```
POST: CPU MIC register Tests : Begin
POST: CPU MIC register Tests : End, Status Passed

POST: PortASIC Memory Tests : Begin
POST: PortASIC Memory Tests : End, Status Passed

POST: CPU MIC interface Loopback Tests : Begin
POST: CPU MIC interface Loopback Tests : End, Status Passed

POST: PortASIC RingLoopback Tests : Begin
POST: PortASIC RingLoopback Tests : End, Status Passed

POST: Inline Power Controller Tests : Begin
POST: Inline Power Controller Tests : End, Status Passed

POST: PortASIC CAM Subsystem Tests : Begin
POST: PortASIC CAM Subsystem Tests : End, Status Passed

POST: PortASIC Port Loopback Tests : Begin
POST: PortASIC Port Loopback Tests : End, Status Passed

Waiting for Port download...Complete
```

Once the POST is done we'll get a final warning:

```
This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and
use. Delivery of Cisco cryptographic products does not imply
third-party authority to import, export, distribute or use encryption.
Importers, exporters, distributors and users are responsible for
compliance with U.S. and local country laws. By using this product you
agree to comply with applicable laws and regulations. If you are unable
to comply with U.S. and local laws, return this product immediately.

A summary of U.S. laws governing Cisco cryptographic products may be found
at:
http://www.cisco.com/wwl/export/crypto/tool/stqrg.html

If you require further assistance please contact us by sending email to
export@cisco.com.
```

And finally you'll see an overview of the hardware that this switch offers:

```
cisco WS-C3560-24PS (PowerPC405) processor (revision G0) with 122880K/8184K
bytes of memory.
Processor board ID CAT0832N0G3
Last reset from power-on
1 Virtual Ethernet interface
24 FastEthernet interfaces
2 Gigabit Ethernet interfaces
The password-recovery mechanism is enabled.

512K bytes of flash-simulated non-volatile configuration memory.
Base ethernet MAC Address      : 00:11:BB:0B:36:00
Motherboard assembly number    : 73-9299-01
Power supply part number      : 341-0029-03
Motherboard serial number     : CATXXXXXX
Power supply serial number    : DTHXXXXXX
Model revision number         : G0
Motherboard revision number   : E0
Model number                   : WS-C3560-24PS-S
System serial number          : CATXXXXXXXX
Top Assembly Part Number      : 800-24791-01
Top Assembly Revision Number  : K0
Version ID                    : N/A
Hardware Board Revision Number : 0x09
```

| Switch | Ports | Model         | SW Version  | SW Image                |
|--------|-------|---------------|-------------|-------------------------|
| *      | 1 26  | WS-C3560-24PS | 12.2(44)SE1 | C3560-ADVIPSERVICESK9-M |

Once the switch is done you finally get to see this message:

**Press RETURN to get started!**

If the switch does not have a configuration, you'll see the following:

```
--- System Configuration Dialog ---

Would you like to enter the initial configuration dialog? [yes/no]:
```

If you type **yes** and press enter it will walk you through a wizard where you can configure some basic settings.



*Even without a configuration our switch will work just like any other "unmanaged" switch. If you connect computers to it they will be able to communicate with each other.*

I'm going to skip it since we'll configure everything ourselves. You'll end up with this after skipping the wizard:

```
Switch>
```

Right now you are in **user mode** and you can recognize it because of the > symbol. When you are in user mode you don't have full access to the device. What we want is **privileged mode** which is also known as **enable mode**.

This is how we do it:

```
Switch>enable
Switch#
```

That's it! We are now in privileged mode where we have full access to our device. You can recognize it because of the # symbol.

If I want to return back to user mode I can do this:

```
Switch#disable
Switch>
```

You'll probably never use it but you can type **disable** to get back to user mode.

So you have full access to your device...now what? Welcome to the marvelous world of typing commands to get things done. Let's start with a simple example. We'll configure the clock on our switch so I can demonstrate how the CLI works:

```
Switch#cl?
clear  clock
```

Whenever I partially type a command I can use the ? to see my options. I typed in "cl?" and the CLI tells me that there are two commands that start with the letters "cl". There's the "clear" command and the "clock" command. Let's try the clock:

```
Switch#clock
% Incomplete command.
```

When you see **% incomplete command** the CLI is expecting more information. What does it want from us? Let's find out:

```
Switch#clock ?
set    Set the time and date
```

It wants us to type "set" so we can set the time and date. Let's obey and do it:

```
Switch#clock set
% Incomplete command.
```

It's still incomplete...let's see why:

```
Switch#clock set ?
hh:mm:ss  Current Time
```

Now we are getting somewhere. I need to type in the time...let's do it:

```
Switch#clock set 14:51:50
% Incomplete command.
```

The time is right but it IOS tells us it's expecting something more ...oh CLI what do you want from me?

```
Switch#clock set 14:51:50 ?
<1-31> Day of the month
MONTH Month of the year
```

It wants a day and month so let's give it what it wants:

```
Switch#clock set 14:51:50 25 1
                        ^
% Invalid input detected at '^' marker.
```

When I try to type the month something goes wrong. This means that it's expecting a different input and what I did is not acceptable. The **^ symbol** tells us what is invalid.

I should have typed "January" instead of the number "1". Let's finish the clock:

```
Switch#clock set 14:51:50 25 January 2013
Switch#
```

Once you type in a command that is correct and press enter you won't see anything like "command accepted". Only a fresh new empty line proves to us that the command has been accepted.

The cool thing about the command line is that you don't have to **fully type** commands. Let me give you an example:

```
Switch#clo ?
set Set the time and date
```

Typing the letters "clo" is enough for IOS to understand that I meant the clock command. This works everywhere:

```
Switch#clo s ?
hh:mm:ss Current Time
```

Just typing "s" is enough for IOS to understand that I meant "set". If you don't type enough letters you will see this:

```
Switch#cl
% Ambiguous command: "cl"
```

Your switch will tell you **ambiguous command** which means it doesn't know what you mean, here's why:

```
Switch#cl?
clear clock
```

Both "clear" and "clock" start with "cl" so IOS doesn't know which of the two commands you want to use.

The CLI offers a couple of useful **shortcuts** for us to use:

1. You can press the **TAB** button to **auto-complete** a command or keyword. This is VERY useful. If you type "clo" and then press TAB it will auto-complete "clo" to "clock".
2. **CTRL-A** brings your cursor to the beginning of the line. This is faster than pressing the left arrow.
3. **CTRL-E** brings your cursors to the end of the line. This is faster than pressing the right arrow.
4. **CTRL-SHIFT-6** interrupts processes like a PING.
5. **CTRL-C** aborts the current command that you were typing and exits configuration mode.
6. **CTRL-Z** ends configuration mode.

Cisco IOS keeps a history of all the commands you previously typed in. You can view them with the following command:

```
Switch#show history
enable
show history
```

Above you see an overview with the commands I have used so far. By default it will only save the last 10 typed commands but we can increase the history size:

```
Switch#terminal history size 30
```

Use the **terminal history size** command to change it. I've set it to 30 commands.

By pressing the UP or DOWN arrow you can browse through commands you have previously used.



If you want to see an overview of your device's capabilities you can use the following command:

```
Godzilla#show version
Cisco IOS Software, C3560 Software (C3560-ADVIPSERVICESK9-M), Version
12.2(44)SE1, RELEASE SOFTWARE (fc1)
Copyright (c) 1986-2008 by Cisco Systems, Inc.
Compiled Fri 07-Mar-08 00:10 by weiliu
Image text-base: 0x00003000, data-base: 0x01900000

ROM: Bootstrap program is C3560 boot loader
BOOTLDR: C3560 Boot Loader (C3560-HBOOT-M) Version 12.2(44)SE5, RELEASE
SOFTWARE (fc1)

Godzilla uptime is 1 hour, 41 minutes
System returned to ROM by power-on
System restarted at 14:24:00 UTC Fri Jan 25 2013
System image file is "flash:/c3560-advipservicesk9-mz.122-44.SE1.bin"

This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and
use. Delivery of Cisco cryptographic products does not imply
third-party authority to import, export, distribute or use encryption.
Importers, exporters, distributors and users are responsible for
compliance with U.S. and local country laws. By using this product you
agree to comply with applicable laws and regulations. If you are unable
to comply with U.S. and local laws, return this product immediately.

A summary of U.S. laws governing Cisco cryptographic products may be found
at:
http://www.cisco.com/wvl/export/crypto/tool/stqrg.html

If you require further assistance please contact us by sending email to
export@cisco.com.

cisco WS-C3560-24PS (PowerPC405) processor (revision G0) with 122880K/8184K
bytes of memory.
Processor board ID CAT0832N0G3
Last reset from power-on
1 Virtual Ethernet interface
24 FastEthernet interfaces
2 Gigabit Ethernet interfaces
The password-recovery mechanism is enabled.

512K bytes of flash-simulated non-volatile configuration memory.
Base ethernet MAC Address       : 00:11:BB:0B:36:00
Motherboard assembly number     : 73-9299-01
Power supply part number        : 341-0029-03
Motherboard serial number       : CATXXXXXXXX
Power supply serial number      : DTHXXXXXXXX
Model revision number           : G0
Motherboard revision number     : E0
```

**Show version** will display our model, hardware, interfaces and more. We also saw this output when we just started the switch.

Let's take a closer look at the interfaces that this switch has:

```
Godzilla#show ip interface brief
```

| Interface          | IP-Address | OK? | Method | Status | Protocol |
|--------------------|------------|-----|--------|--------|----------|
| Vlan1              | unassigned | YES | unset  | up     | up       |
| FastEthernet0/1    | unassigned | YES | unset  | down   | down     |
| FastEthernet0/2    | unassigned | YES | unset  | up     | up       |
| FastEthernet0/3    | unassigned | YES | unset  | down   | down     |
| FastEthernet0/4    | unassigned | YES | unset  | up     | up       |
| FastEthernet0/5    | unassigned | YES | unset  | down   | down     |
| FastEthernet0/6    | unassigned | YES | unset  | up     | up       |
| 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     |
| FastEthernet0/13   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/14   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/15   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/16   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/17   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/18   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/19   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/20   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/21   | unassigned | YES | unset  | up     | up       |
| FastEthernet0/22   | unassigned | YES | unset  | down   | down     |
| FastEthernet0/23   | unassigned | YES | unset  | down   | down     |
| FastEthernet0/24   | unassigned | YES | unset  | up     | up       |
| GigabitEthernet0/1 | unassigned | YES | unset  | down   | down     |
| GigabitEthernet0/2 | unassigned | YES | unset  | down   | down     |

The **show ip interface brief** is a very useful command. It shows us all the interfaces and their status. This switch has 24x FastEthernet interfaces and 2x Gigabit Interfaces.

The keyword **status** tells us whether the interface is up or down. This is the physical status so it means whether there is a cable connected to the interface or not. The keyword **protocol** tells us if the interface is operational or not. It's possible that the status shows an interface as up but that the protocol is down because of a security violation.

If we want we can take a closer look at one of the interfaces:

```
Godzilla#show interfaces fastEthernet 0/2
FastEthernet0/2 is up, line protocol is up (connected)
  Hardware is Fast Ethernet, address is 0019.569d.5704 (bia 0019.569d.5704)
  MTU 1900 bytes, BW 100000 Kbit, DLY 100 usec,
    reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 100Mb/s, media type is 10/100BaseTX
  input flow-control is off, output flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input never, output 00:00:01, 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 5000 bits/sec, 2 packets/sec
    0 packets input, 0 bytes, 0 no buffer
    Received 0 broadcasts (0 multicasts)
    0 runs, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog, 0 multicast, 0 pause input
    0 input packets with dribble condition detected
  3777 packets output, 1296328 bytes, 0 underruns
  0 output errors, 0 collisions, 1 interface resets
  0 babbles, 0 late collision, 0 deferred
  0 lost carrier, 0 no carrier, 0 PAUSE output
  0 output buffer failures, 0 output buffers swapped out
```

Use the **show interface** command and specify the interface that you want to look at. Above you can see an example of the FastEthernet 0/2 interface. Some of the things that we see are the status, the speed (100Mbit) and the duplex settings (full-duplex). You can also see the number of incoming and outgoing packets.

So now you have an idea how the CLI works, let's continue by creating a basic configuration for our device.

Most of the things we want to configure on a Cisco switch or router have to be done from the configuration mode:

```
Switch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#
```

Use the **configure terminal** command to get into the configuration mode. You can recognize the configuration mode because it now says **(config)#**.

If you try to run a **show** command from the configuration mode you will get an error like this:

```
Switch(config)#show interfaces fastEthernet 0/2
                ^
% Invalid input detected at '^' marker.
```

This is because you are running a “global” command from the “configuration mode”. It might be annoying to switch between “global” mode and “configuration mode” all the time so there is a workaround for this:

```
Switch(config)#do show interfaces fastEthernet 0/2
FastEthernet0/2 is up, line protocol is up (connected)
...
```

Type **do** in front of the show command and it will work anyway.

Let’s give my switch another name. If you have a large network it’s useful to give all of your devices a unique name:

```
Switch(config)#hostname Godzilla
Godzilla(config)#
```

Use the **hostname** command to change it to whatever you like.

If we want to change the configuration of an interface we need to access the **interface configuration**. You can do it like this:

```
Godzilla(config)#interface fastEthernet 0/2
Godzilla(config-if)#
```

Type the **interface** command and the interface number you want to configure. You can see we are in the interface configuration because it says **(config-if)#**. If we want we can change the duplex and/or speed settings:

```
Godzilla(config-if)#duplex full
Godzilla(config-if)#speed 100
```

Use the **duplex** and **speed** command to change them. In my example I changed duplex to full and speed to 100Mbit.

If you have many interfaces it might be useful to configure a description so you know which interface connects to which device:

```
Godzilla(config)#interface fastEthernet 0/2
Godzilla(config-if)#description Connects to Rene's Computer
```

By typing **interface** I can access the configuration for a specific interface. You can recognize this because the terminal now says **(config-if)#**. The **description** command lets us set a description.

If you want to configure a lot of interface it might be time-consuming to configure them one at a time.

We can also select a range of interfaces and configure all of them at the same, here's how to do it:

```
Godzilla(config)#interface range fa0/3 - 10
Godzilla(config-if-range)#
```

The **interface range** commands lets us select multiple interfaces. I used it to select interface FastEthernet 0/3,4,5,6,7,8,9 and 10.

Whenever you want to go back from the interface configuration to the global configuration mode you can do it like this:

```
Godzilla(config-if-range)#exit
Godzilla(config)#
```

Just type **exit** and you'll be back in the global configuration mode.

Right now everyone can connect to our switch and configure whatever you like. It's a good idea to protect it by setting some passwords. One of the things we can do is protect the console port:

```
Godzilla(config)#line console 0
Godzilla(config-line)#password mypassword
Godzilla(config-line)#login
```

First I use the **password** command to set a password. I also need to supply the **login** command otherwise the switch won't ask for the password. Now every time I connect the blue Cisco console cable this will happen:

```
Godzilla con0 is now available

Press RETURN to get started.

User Access Verification

Password:
```

Before I get to the user mode I have to type in a console password. This will ensure that not just anyone can connect a console cable and configure our switch.

I can also protect the privileged (enable) mode. Right now it works like this:

```
Godzilla>enable
Godzilla#
```

We type in "enable" and you have full access to the switch. It's wise to configure our switch so it will prompt for a password every time someone wants to access the privileged mode. We can do it like this:

```
Godzilla(config)#enable password mypassword
```

Use the **enable password** command to set a password. Now whenever I want to access the privilege mode this will happen:

```
Godzilla>enable
Password:
Godzilla#
```

Besides setting passwords it might be a good idea to configure a banner with a warning message:

```
Godzilla(config)#banner login % Authorized Users Only! %
```

The **banner** command lets us configure a banner. You need to use a symbol to tell the switch when the banner begins and ends. I used the % symbol but you can use any symbol you like. Now whenever someone wants to log into our switch this is what they will see:

```
Godzilla con0 is now available

Press RETURN to get started.

Authorized Users Only!
```

Above you see the banner that I configured.

Right now we are still connected to the switch using the blue console cable. We can also connect to it remotely using telnet or SSH. We will have to configure an IP address on our device first if we want this.

This is how you do it on a switch:

```
Godzilla(config)#interface vlan 1
Godzilla(config-if)#ip address 192.168.1.1 255.255.255.0
Godzilla(config-if)#no shutdown
```

The VLAN 1 interface can be used for management. I need to type in an IP address and subnet mask. This interface is disabled by default so I need to type **no shutdown** to activate it.

If you have a router you can configure an IP address like this:

```
Router(config)#interface fastEthernet 0/0
Router(config-if)#ip address 192.168.1.2 255.255.255.0
```

On a router you have to configure an IP address on one of the interfaces. I'll use the FastEthernet 0/0 interface.

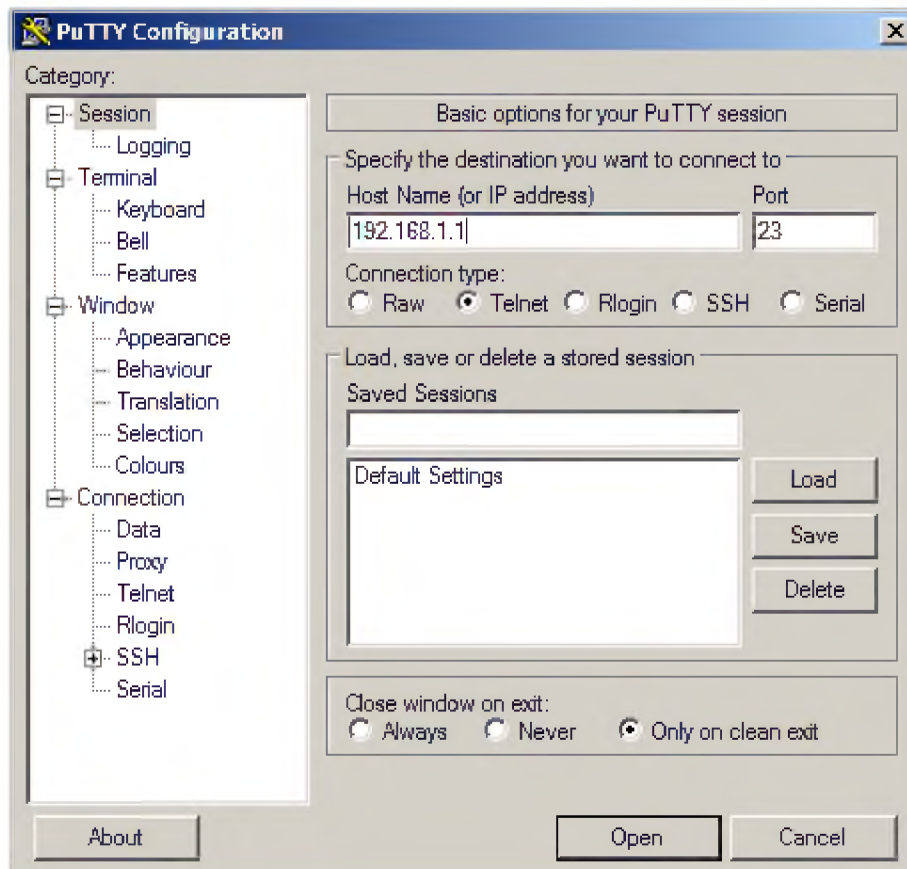
Let's configure telnet so that we can access the device remotely:

```
Godzilla(config)#line vty 0 4
Godzilla(config-line)#password mypassword
Godzilla(config-line)#login
```

A switch or router has a number of virtual lines that you can use for remote access. These are called VTY (Virtual Terminal) lines. I can configure these using the **line vty** command. In my example I'm selecting VTY line 0 up to 4 so that's 5 virtual lines total.

I have configured a password and the **login** command is required otherwise the switch won't ask for the password.

Now you can connect a UTP cable from your computer to the switch and use putty to telnet to the switch:



Just select **telnet** and type in the IP address of your switch. Click on Open and it will connect to it.

Telnet is convenient and easy to configure but it's also **insecure** because everything is sent in clear-text. It's better to configure **SSH**. SSH can also be used to connect remotely to your switch (or router) but all traffic will be encrypted.



*Not all IOS versions offer SSH by default. Check your IOS version to see if it's possible to configure SSH.*

Here's how to configure SSH:

```
Godzilla(config)#username rene password mypassword
```

SSH works with usernames. I'll create an account for myself and a password.

```
Godzilla(config)#ip domain-name gns3vault.local
```

We need to configure a domain name because SSH requires certificates. You can pick anything you like.

Now we can generate the keys that SSH requires:

```
Godzilla(config)#crypto key generate rsa
The name for the keys will be: Godzilla.gns3vault.local
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.
```

**How many bits in the modulus [512]: 1024**

% Generating 1024 bit RSA keys, keys will be non-exportable...[OK]

```
Godzilla(config)#
```

```
Jan 25 17:23:27.109: %SSH-5-ENABLED: SSH 1.99 has been enabled
```

Use **crypto key generate** to generate some RSA keys for SSH. The key should be at least 1024 bits. By default it will enable SSH version 1.99 but for security reasons it's better to use version 2:

```
Godzilla(config)#ip ssh version 2
```

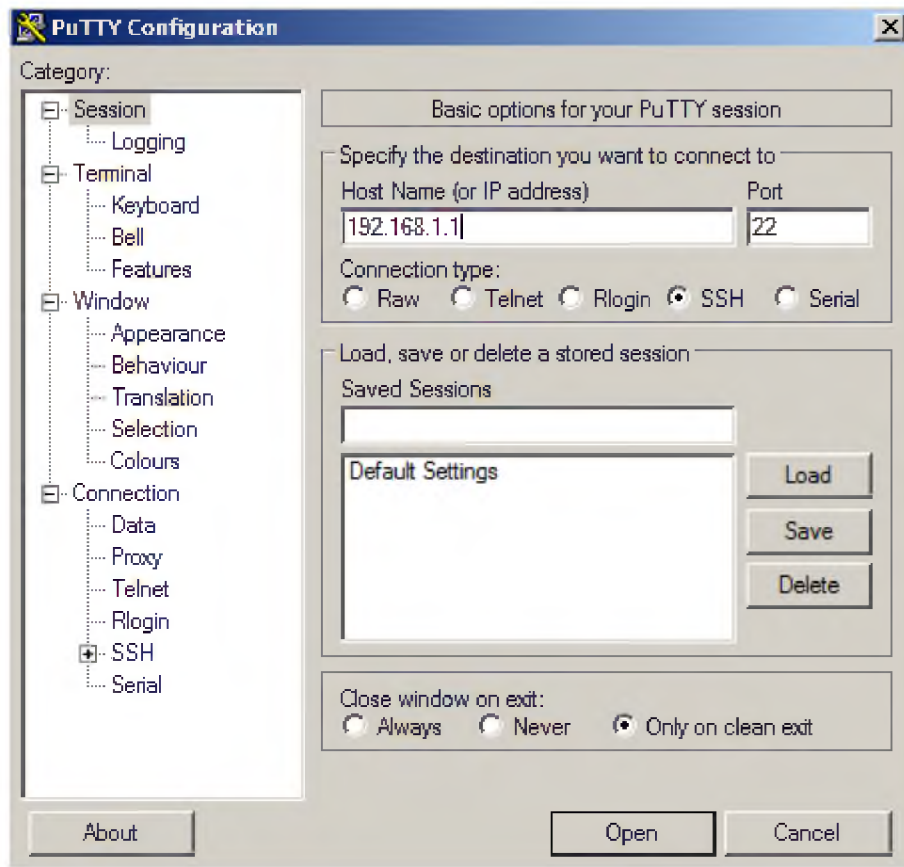
Use **ip ssh version 2** to switch to version 2. Last step is to configure the VTY lines:

```
Godzilla(config)#line vty 0 4
Godzilla(config-line)#login local
Godzilla(config-line)#transport input ssh
```

First we use **login local** to tell the switch to use the local database with the username that I configured. We also require the **transport input** command so that we only allow SSH and no telnet.



We can test our configuration with putty:



Click on the SSH button and type in the IP address of the device. Click on Open and you'll be able to connect.

Everything that you configure on a switch or router is stored in a configuration file called the **running-configuration**.

You can take a look at the running configuration like this:

```
Godzilla#show running-config
Building configuration...

Current configuration : 1587 bytes
!
! Last configuration change at 16:58:25 UTC Fri Jan 25 2013
! NVRAM config last updated at 15:51:32 UTC Fri Jan 25 2013
!
version 12.2
no service pad
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Godzilla
!
boot-start-marker
boot-end-marker
!
enable password mypassword
!
username rene password 7 011E1F145A1815182E5E4A
!
!
spanning-tree mode pvst
spanning-tree extend system-id
!
vlan internal allocation policy ascending
interface FastEthernet0/1
!
interface FastEthernet0/2
  description Connects to Rene's Computer
!
interface FastEthernet0/3
!
interface Vlan1
  ip address 192.168.1.1 255.255.255.0
!
ip default-gateway 192.168.1.254
ip classless
ip http server
ip http secure-server
!
control-plane
!
banner login ^C Authorized Users Only! ^C
!
line con 0
  password mypassword
  login
line vty 0 4
  password mypassword
  login local
  transport input ssh
line vty 5 15
  login
!
end
```

Use the **show running-config** command to take a look at the running configuration. This is the configuration that is active at the moment.

If you want to remove something from the running-config you can use the **no** keyword in front of it. For example:

```
Godzilla(config)#no hostname Godzilla
Switch(config)#
```

Typing **no hostname Godzilla** would remove this line from the running-config.

The running-config is active in **RAM** which means that if you power off your device, your configuration is **gone**.

Of course we can save our running-config in a permanent location; this is how we do it:

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

We need to use the **copy** command to copy the running-config to the startup-config. The startup-config is saved in **NVRAM**. Whenever you power on your device, it will look for the startup-config in the NVRAM and copy it to the running-config in our RAM.

If you want to remove your configuration we can delete the startup-config:

```
Godzilla#erase startup-config
Erasing the nvram filesystem will remove all configuration files! Continue?
[confirm]
[OK]
Erase of nvram: complete
```

Type **erase startup-config** to delete it from the NVRAM. You will have to reload your switch or router before this will take effect:

```
Godzilla#reload
Proceed with reload? [confirm]
```

You can do this with the **reload** command.

If you looked closely at the output of the show running-config command you could see that all passwords are there in **clear-text**. This doesn't sound like a very good idea right? Anyone that has access to our configuration file will have the passwords. There is a command that lets us encrypt all the passwords in the configuration.

Here's how to do it:

```
Godzilla(config)#service password-encryption
```

The **service password-encryption** command will encrypt all passwords in the configuration.

Let's take a look at the difference:

```
Godzilla#show running-config
!
enable password 7 0941571918160405041E00
!
line con 0
password 7 12141C0713181F13253920
login
line vty 0 4
password 7 12141C0713181F13253920
login local
transport input ssh
```

I didn't include everything from the running-config, just the passwords to keep it readable. You can see that the passwords have been encrypted and that there's a "7" in front of the password. This encryption type is called **type 7** that's why you see it.

Now this looks great but in reality it's a bad idea to use this form of encryption since it's **really weak**. There are a couple of websites on the Internet that let you decipher these encrypted passwords with a couple of mouse clicks, here's an example:

<http://www.ibeast.com/content/tools/CiscoPassword/index.asp>

Just copy and paste the encrypted password from the running-config and a few seconds later you'll have the decrypted version...OUCH!

Of course Cisco has a solution for this. Instead of the poor type 7 encryption we can use MD5 hashes for most of our passwords. This is far more secure so let me show you how to do this for your "enable" password:

```
Godzilla(config)#enable secret mypassword
```

Instead of the keyword "password" you should use **secret**. This will create a MD5 hash of the password and save it in the running-config.

Let's take a look:

```
Godzilla#show running-config
Building configuration...

Current configuration : 1673 bytes
!
! Last configuration change at 17:12:56 UTC Fri Jan 25 2013
! NVRAM config last updated at 15:51:32 UTC Fri Jan 25 2013
!
version 12.2
no service pad
service timestamps debug datetime msec
service timestamps log datetime msec
service password-encryption
!
hostname Godzilla
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$RpKB$.1DX18JBZpgNogeS0mAs40
```

Above you see the MD5 hash of the password, not the actual password that is encrypted.

It might become annoying to browse through the entire running-config everytime you want to check just one item. Cisco IOS has a couple of "operators" that we can use to make our lives easier:

```
Godzilla#show running-config | include secret
enable secret 5 $1$RpKB$.1DX18JBZpgNogeS0mAs40
```

Instead of just typing "show running-config" and hitting enter I can use the **| include** operator so it shows me only the lines that have the word "secret" in them.

```
Godzilla#show running-config | begin line con 0
line con 0
 password 7 12141C0713181F13253920
 login
line vty 0 4
 password 7 12141C0713181F13253920
 login
line vty 5 15
 login
!
end
```

I can also use **| begin** and it will not start at the beginning of the config but at the section that I request. Above I'm using it to show the "line con 0" configuration and everything below.

Any other useful commands? One of the annoying things of the CLI is that whenever you type in a wrong command you'll see something like this:

```
Godzilla#clockk
Translating "clockk"...domain server (255.255.255.255)
% Unknown command or computer name, or unable to find computer address
```

By accident I type "clockk" but this command doesn't exist. What Cisco IOS thinks is that you typed in the hostname of a device you want to telnet to. As a result it will do a DNS lookup for the hostname "clockk" but of course it will never get a response. This can take 1 or 2 seconds and you can't abort it. We can solve this by using the following command:

```
Godzilla(config)#no ip domain-lookup
```

The **no ip domain-lookup** command will tell our switch that it shouldn't try any DNS lookups. Now whenever you type in a wrong command you don't have to wait for a DNS lookup that will never be successful.

Sometimes the CLI will show you notification messages like this one:

```
Godzilla(config)#hostn%LINK-5-CHANGED: Interface FastEthernet0/1, changed
state to down
```

It can be useful to see these kind of messages but the annoying part is that when you are typing a command, the CLI will output these notifications on top of whatever you are typing. You can see it in my example above, I was trying the hostname command while suddenly an interface went down. Now I can't see what I was typing...

There's a command to prevent this:

```
Godzilla(config)#line console 0
Godzilla(config-line)#logging synchronous
```

```
Godzilla(config)#line vty 0 4
Godzilla(config-line)#logging synchronous
```

Use the **logging synchronous** command to keep the last line readable. I have to do this for the console and the VTY lines (telnet or SSH) separately. Let me show you the difference:

```
%LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to down
Godzilla(config)#hostname GodzillaTheSecond
```

Above you see that the command line is now at the bottom and the notification appeared above it.

When you are taking a break from playing with your device you'll notice that Cisco IOS will kick you out of the CLI after a while and you'll have to login again.

We can prevent this:

```
Godzilla(config)#line console 0
Godzilla(config-line)#exec-timeout 0 0
```

Setting it to 0 with the **exec-timeout** command means the console will never kick you out. This is useful for our lab environment but in a production network I wouldn't recommend this for security reasons.

Besides the CLI we can use the GUI to configure our switches or routers.



*CCP is no longer on the CCNA exam so if you want, you can skip the upcoming part. I decided to leave it in the book so you can see what the GUI looks like...*

If you want to use CCP you have two options:

- You can install CCP on the flash memory of your router.
- You can run it from your PC.

You can download CCP from the Cisco website:

<http://software.cisco.com/download/release.html?mdfid=281795035&softwareid=282159854&release=2.7&relind=AVAILABLE&rellifecycle=&reltype=latest>

I downloaded the "PC based" version and release 2.6. You also need to make sure you are using the latest version of java and the adobe flash player.

The following part will be configured on a router, not on a switch!

If you want to use the GUI you first have to prepare your router:

```
Router>enable
Router#configure terminal
Router(config)#interface fastEthernet 0/0
Router(config-if)#ip address 192.168.1.2 255.255.255.0
Router(config-if)#no shutdown
```

First I will configure an IP address on the FastEthernet 0/0 interface. Unlike a switch we can configure an IP address on each interface of a router. Secondly I need to enable the HTTP server:

```
Router(config)#ip http server
```

First you need to enable the HTTP in the router. You can do this with the **ip http server** command.



*Enabling HTTP server is the "quick and dirty" way to prepare the router for CCP. For a lab environment this is fine. If you plan to use CCP in a production network*

*it's better to use HTTPS. HTTP sends everything in clear-text while HTTPS is encrypted.*

Let's create a username:

```
Router(config)#username CCP secret MYROUTER
```

The command above will create a username called "CCP" and I'm using password "MYROUTER". Note that I'm using "secret" so not the actual password but a MD5 hash will be stored.



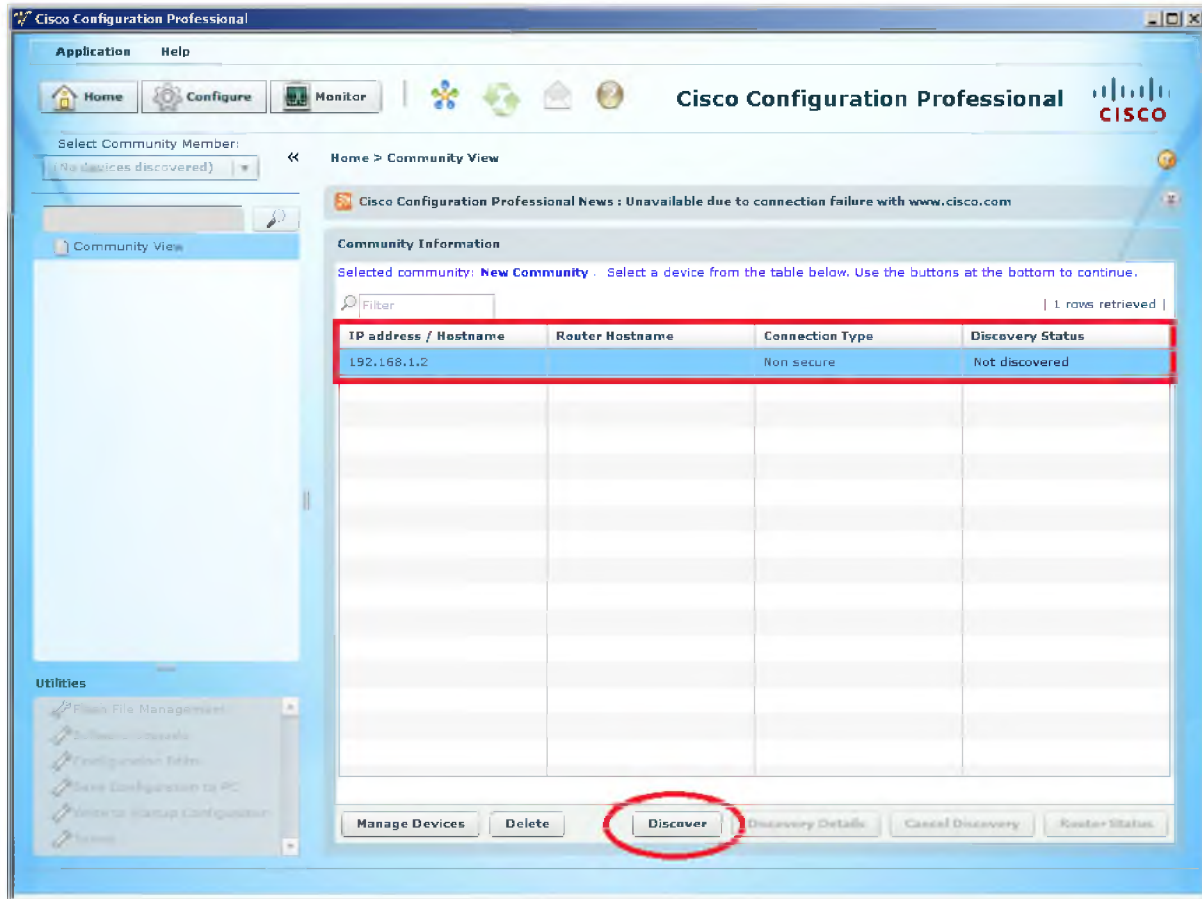
After the installation of CCP you will find a shortcut on your desktop. Click on it and if your java version and flash player are up-to-date it will launch CCP.

CCP will greet you with the following screen when you start it for the first time:

A screenshot of the 'Select / Manage Community' window in Cisco Configuration Professional. The window has a title bar with a question mark and a close button. Below the title bar is a toolbar with icons for file operations. The main area has a 'New Community' button and a list of communities. Below this is a section titled 'Enter information for up to 10 devices for the selected community'. It contains a table with four columns: 'IP Address/Hostname', 'Username', 'Password', and 'Connect Securely'. The first row is pre-filled with '192.168.1.2', 'CCP', and a masked password. The other rows are empty. At the bottom, there is a 'Discover all devices' checkbox and 'OK' and 'Cancel' buttons.

Here you are supposed to configure the routers that you want to manage. I typed in the IP address of my router and the username/password. Click OK and you will return to the main screen:

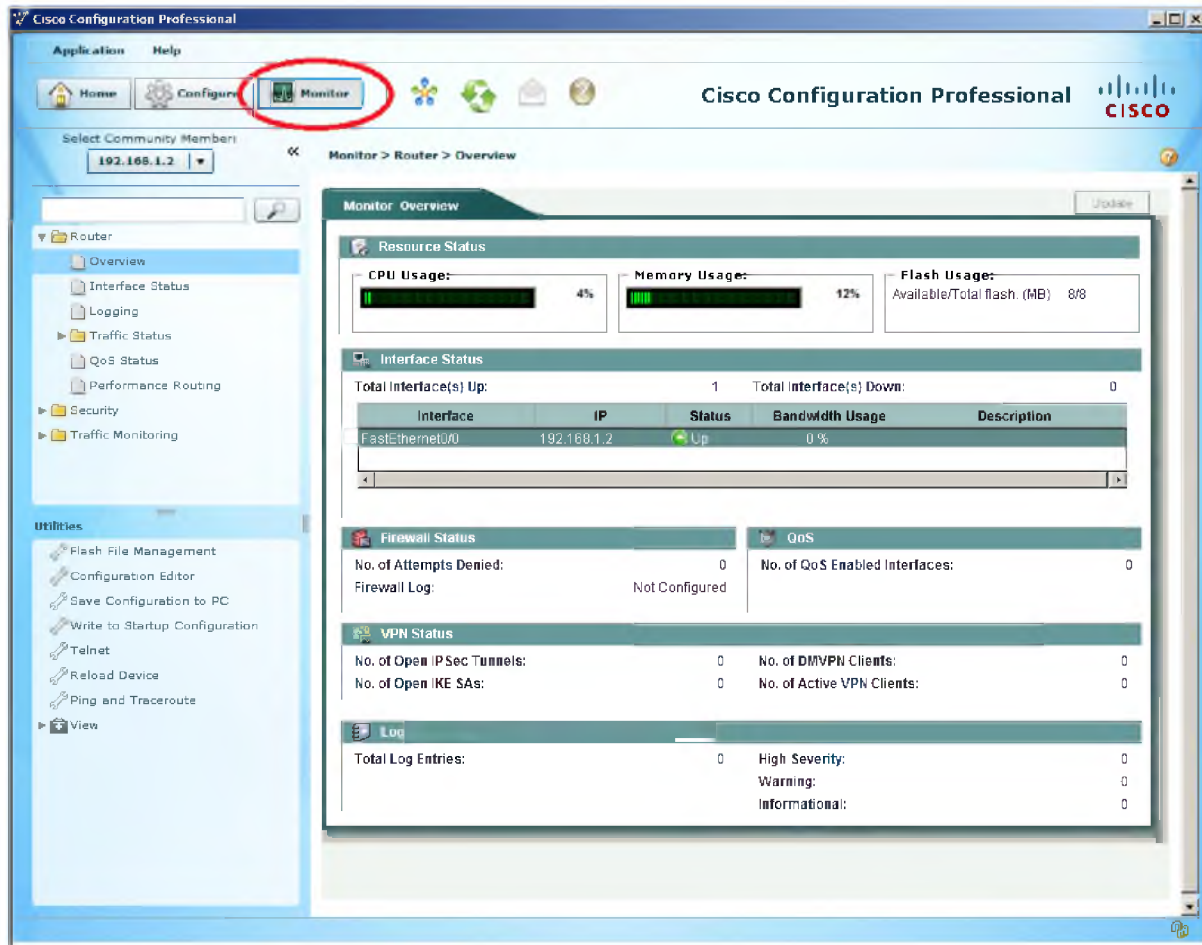




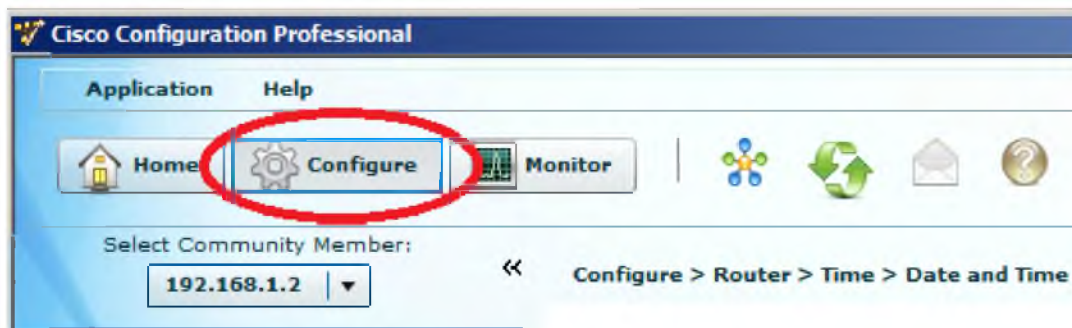
The router now shows up at the main screen but CCP hasn't communicated yet with the router. Click on the **discover** button and CCP will check if the router is present. Now we can monitor or configure the router...



After the discovery you can select the IP address of your router at the **select your community member** button. You can then choose to **monitor** or **configure** your router. Let's click on monitor!

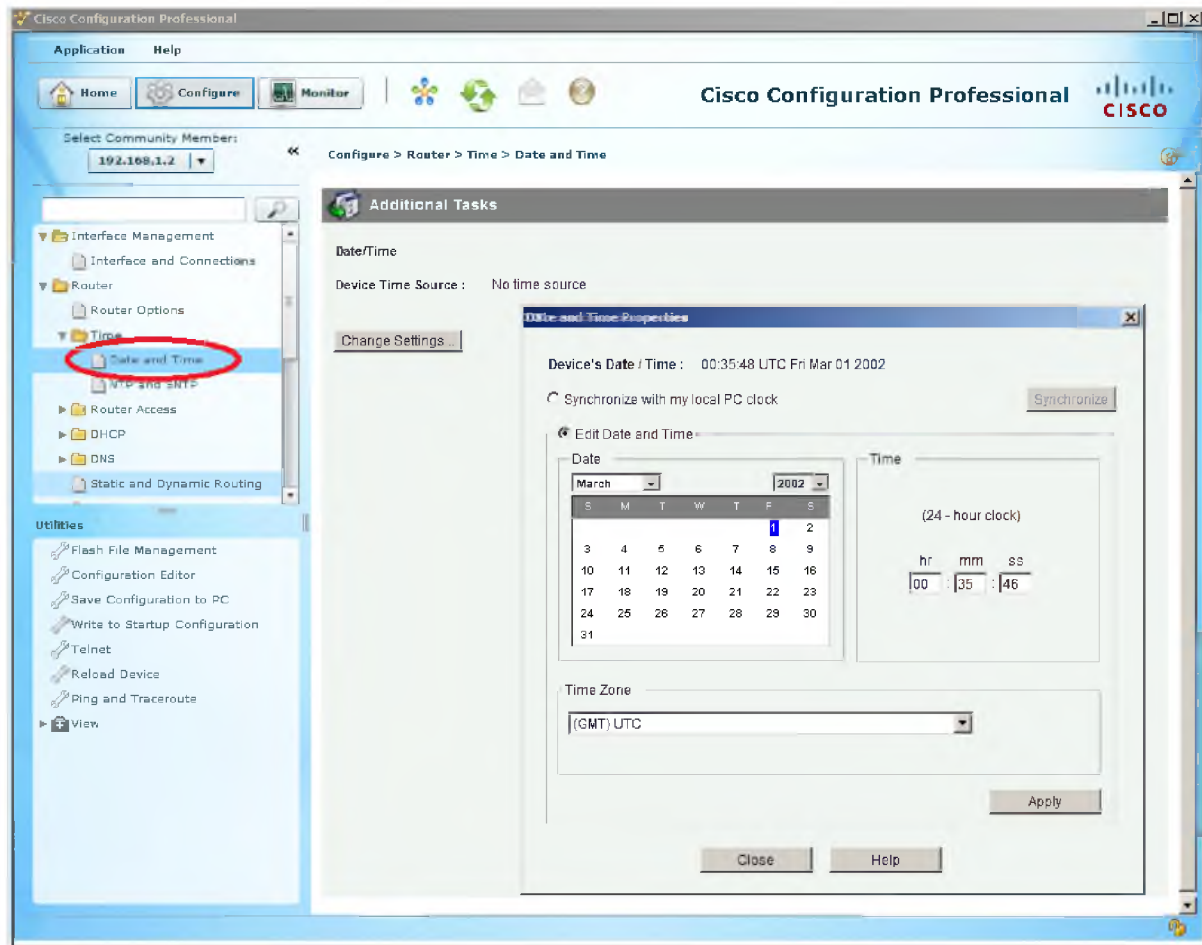


When you click on monitor you will see an overview of the CPU and memory usage, your interface statuses, available flash memory and some other things. Let's see if we can configure our router using CCP:



When you click on configure you will be able to make changes to your router.

For example I can configure the clock using CCP:



Just click on the **Time** dropdown and select **Date and Time** to configure the clock.

Here's another example for SSH:



If you want some exercise with CCP. See if you can create a basic configuration for your router using CCP instead of the command-line. In the rest of the book I will only use the CLI, even in the Cisco exams the focus is on the CLI, not the GUI.



*Right now you might think "CCP looks pretty good" and configuring the clock or SSH looks easier with the GUI than the CLI. This is probably true but when we get to more complex configurations, the CLI will be your friend. If you don't know how to configure something CCP can be useful. Use one of its wizards and then do a "show run-config" on the CLI to see what configuration it created for you.*

This is the end of the chapter and you have now seen the basics of how to configure a router or switch. In the upcoming chapters I will show you plenty of commands to use. You will notice that the more you work with the CLI, the faster you become.

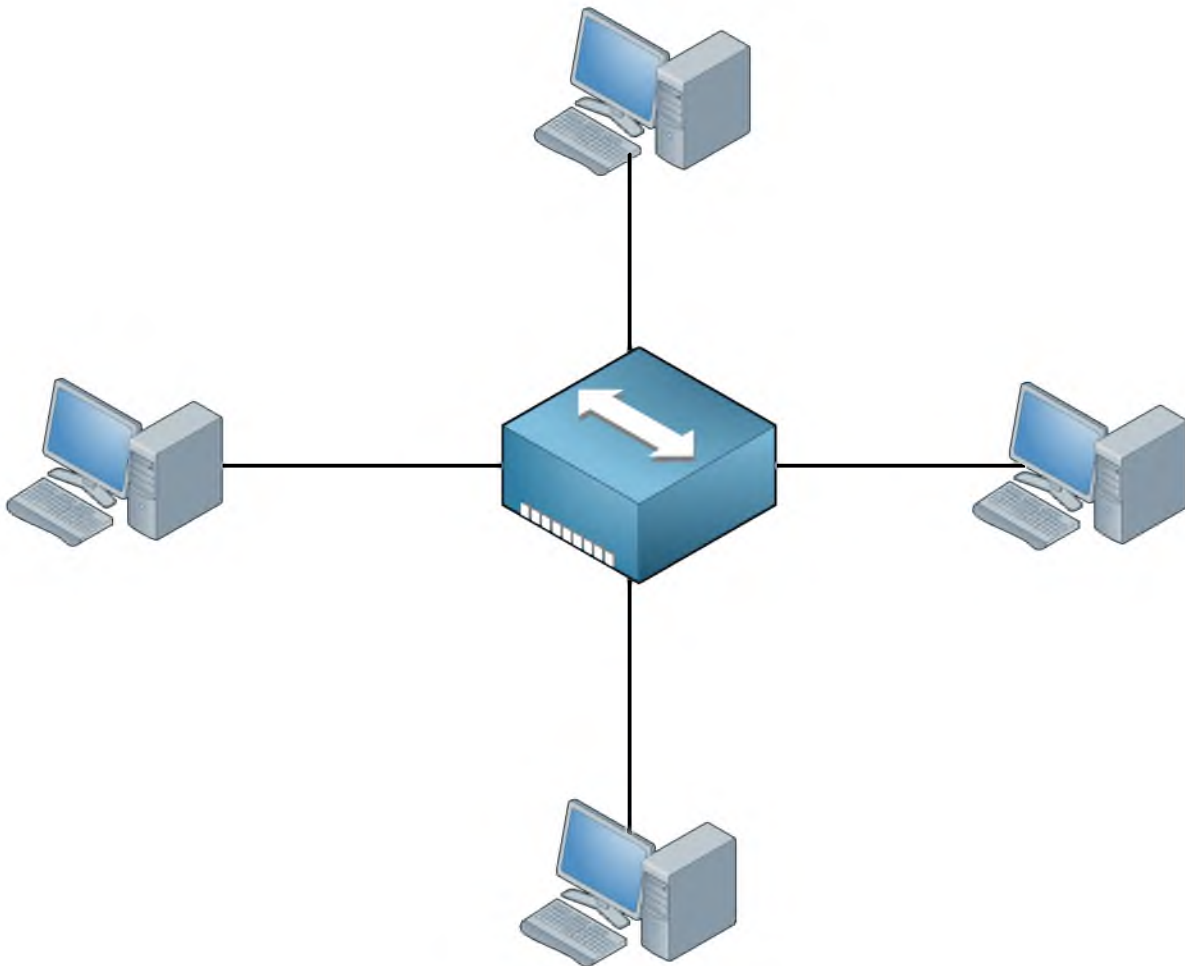
## 8. Hubs, Bridges and Switches

In the beginning of the book I talked a little bit about collisions and hubs. In this chapter we'll talk about those topics a bit more and the difference between hubs, bridges and switches.

A hub is nothing more than a **physical repeater**, if it receives an electrical signal on one interface it will repeat it by sending it on all its interfaces except the one it originated from. There is no intelligence in a hub and it only operates on the physical layer of the OSI model (layer 1 device).

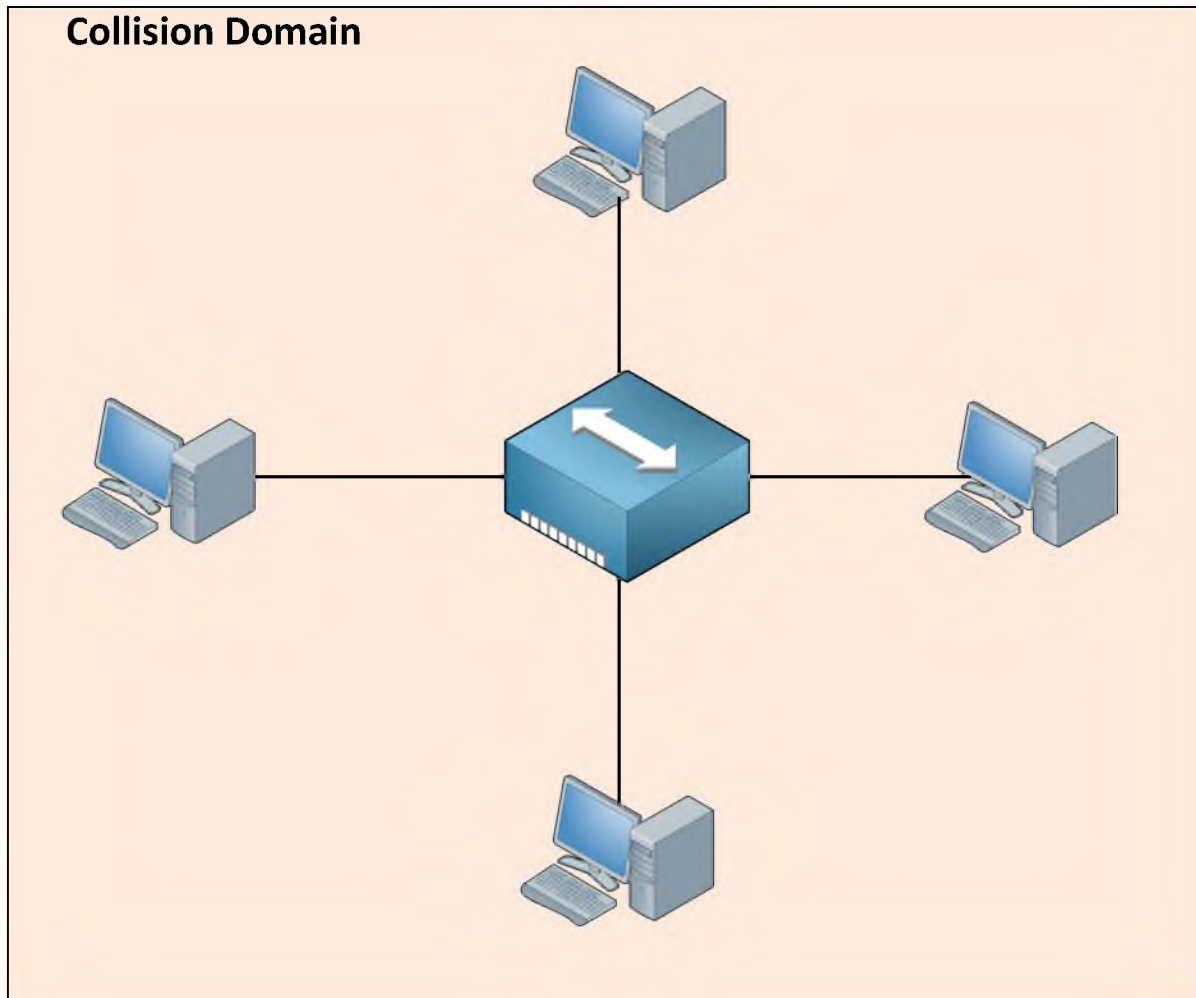
Since we are sharing the physical medium, computers are running in half-duplex and we can get collisions. If we get a collision we can solve this by using the CSMA/CD protocol.

The more computers in your network, the bigger the chance you get collisions. More collisions means your throughput will go down.



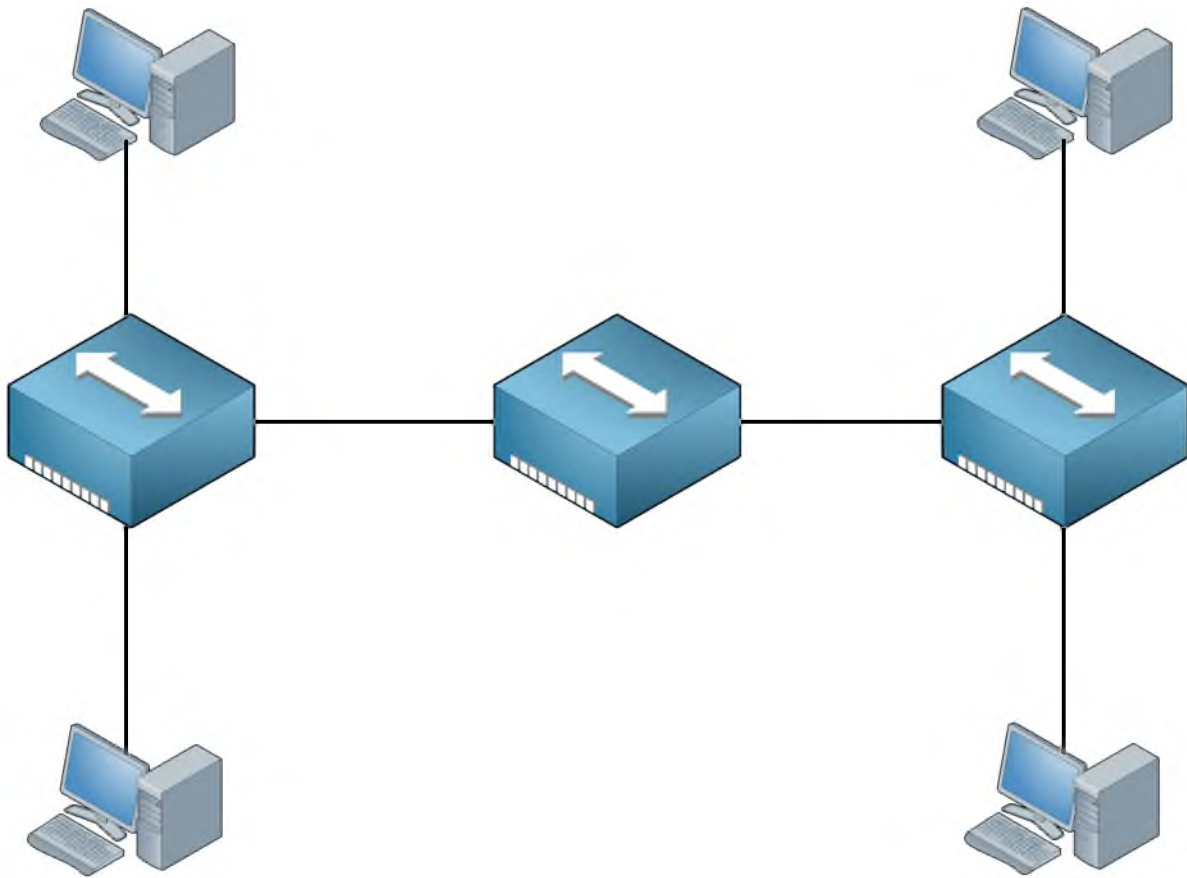
In this example we have a hub in the middle, pay attention to the icon I'm using since this is the "original" Cisco icon they use for hubs. If one of our computers sends some data, the hub will just repeat the electric signal on all other ports which means everyone will receive

this data whether they need it or not. The network is running half-duplex which means we can get collisions here. Since we can get collisions everywhere because of the hub, we call this a single "collision domain".

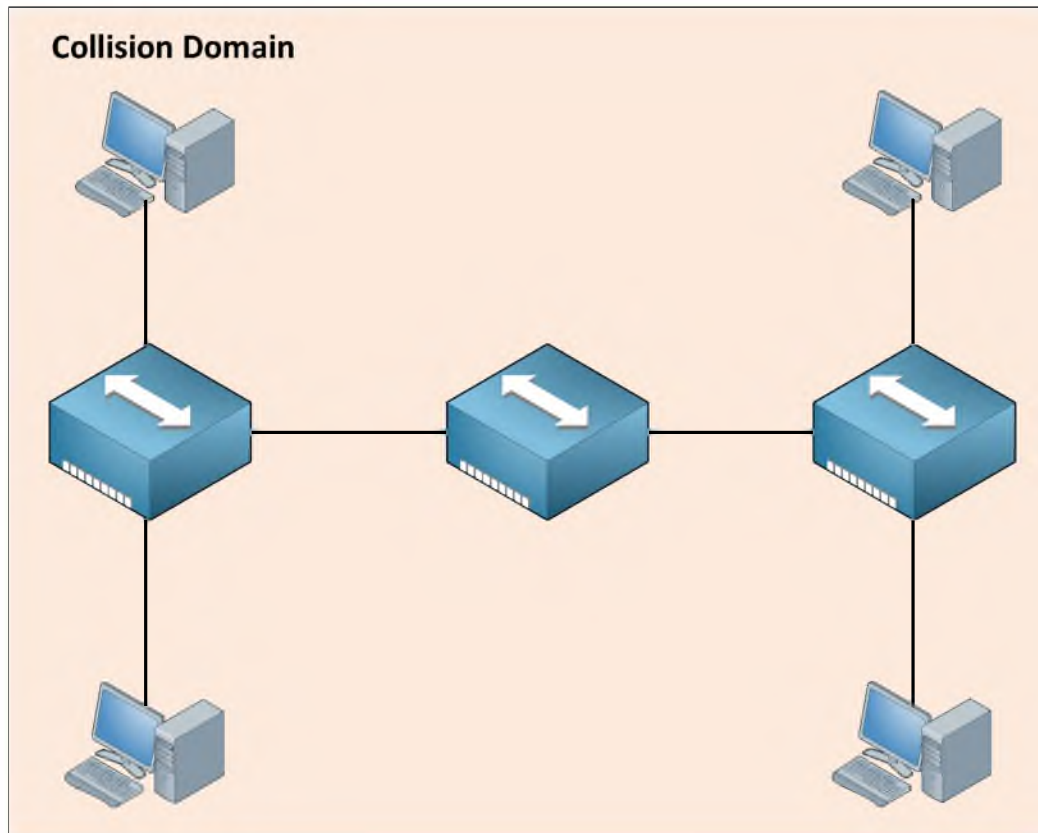


As networks grew larger we also got more collisions, effectively decreasing our throughput.





If you look at the example above, where do you think we will encounter collisions? It's all hubs so we get collisions everywhere! It's still one big collision domain.



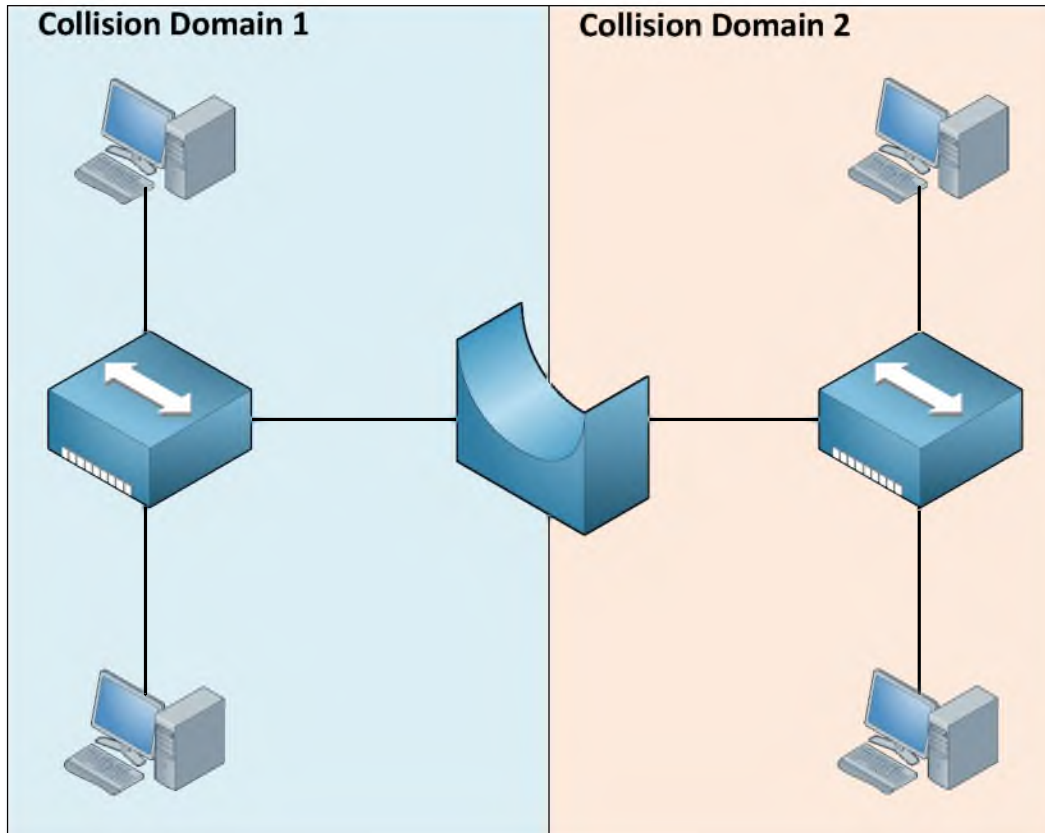
This is where some smart people started to think, there had to be a way to decrease these collisions so throughput wouldn't be affected. The answer was a device which had more intelligence than the hub, thus the bridge was born.

A bridge has "intelligence" and operates at the data link layer (layer 2) of the OSI model, let's see what it can do:

- Make decisions where to send Ethernet frames by looking at the MAC addresses.
- **Forward** Ethernet frames on ports where they are needed.
- **Filter** Ethernet frames (discard them).
- **Flood** Ethernet frames (send them everywhere).
- They only have a few ports.
- They are slow.



Let's take the previous picture and replace the hub in the middle with a bridge:



You can see we now have 2 collision domains. The bridge has intelligence and will not forward Ethernet frames if it's not required. If the computer on the top left would send an Ethernet frame meant for the computer at the bottom left, the bridge will receive this Ethernet frame on its left interface but won't forward it to the other computers. That's great so bridges break up collision domains.

Enough history lessons now, we don't use hubs or bridges nowadays. We do use switches however!

## *A switch is a bridge on steroids!*

- Switches have many ports.
- Switches can have different port speeds like FastEthernet or Gigabit.
- Fast Internet switching.
- Large buffers.
- Different switching modes:
  - **Cut-through**
  - **Store-and-forward**
  - **Fragment-free**

Basically a bridge and switch is the same thing, it's just that the switch is the evolved version of the bridge. We have dedicated chips called ASICs (Application Specified Integrated Circuit) that take care of switching which makes them lightning-fast.

Switches come in many sizes, the smaller ones like the Cisco 2960:



Courtesy of Cisco Systems, Inc. Unauthorized use not permitted

Or the really large switches like the 6500 series:



Courtesy of Cisco Systems, Inc. Unauthorized use not permitted

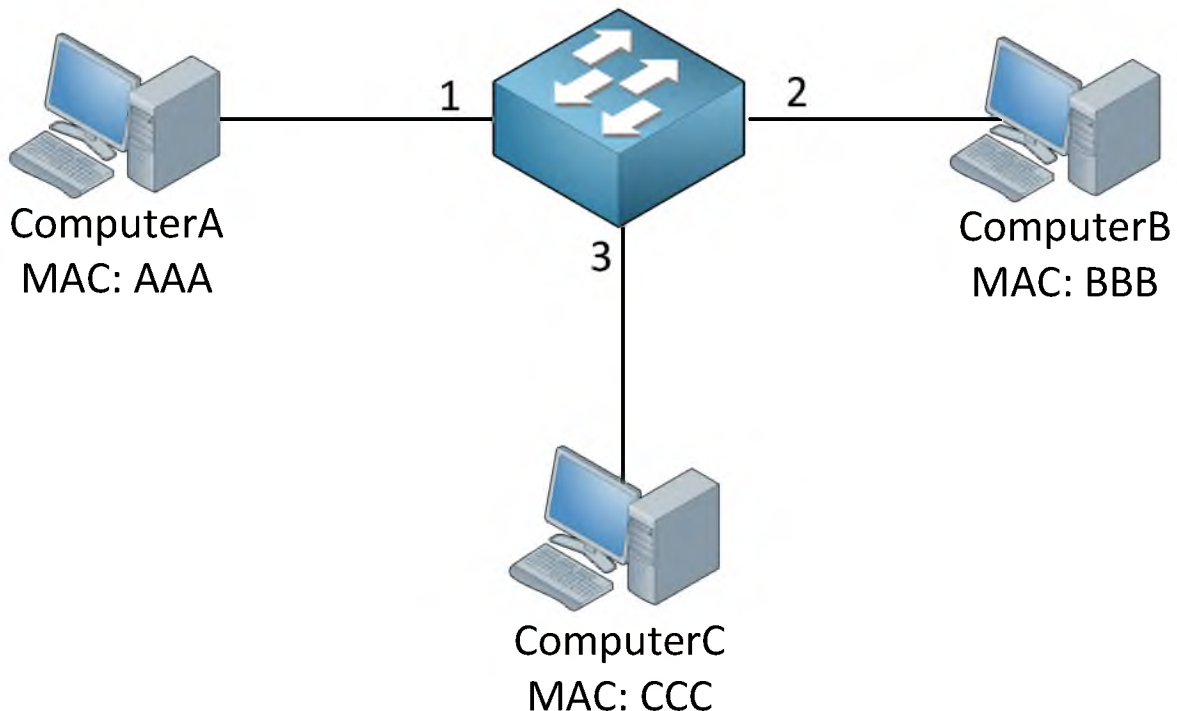
Managed switches like the ones from Cisco have many more features but the “core” of switching is the same of bridging. Switches generally have 3 different switching modes:

- Cut-through switching: The switch will start forwarding the frame before the whole frame has entered the switch. The switch only needs to know the destination MAC address so as soon as it reads it it can start forwarding. This is fast but less reliable if you have corrupt frames.
- Store-and-forward: The switch will receive the complete frame, check if it's errors free and then forward it. If it's corrupt it will be discarded.

- Fragment-free: The switch will check if the first 64 bytes are OK, basically this is a trade-off between cut-through and store-and-forward switching.

Nowadays all Cisco catalyst switches use store-and-forward except for the high-end Nexus switches which can also do an adapted version of cut-through (you can forget about that for your CCNA).

How does a bridge or switch work? I told you that it has some intelligence compared to a hub and that it operates on the data link layer of the OSI-model (layer 2) but I didn't explain yet how it works. Let's look at an example and see what's going on:

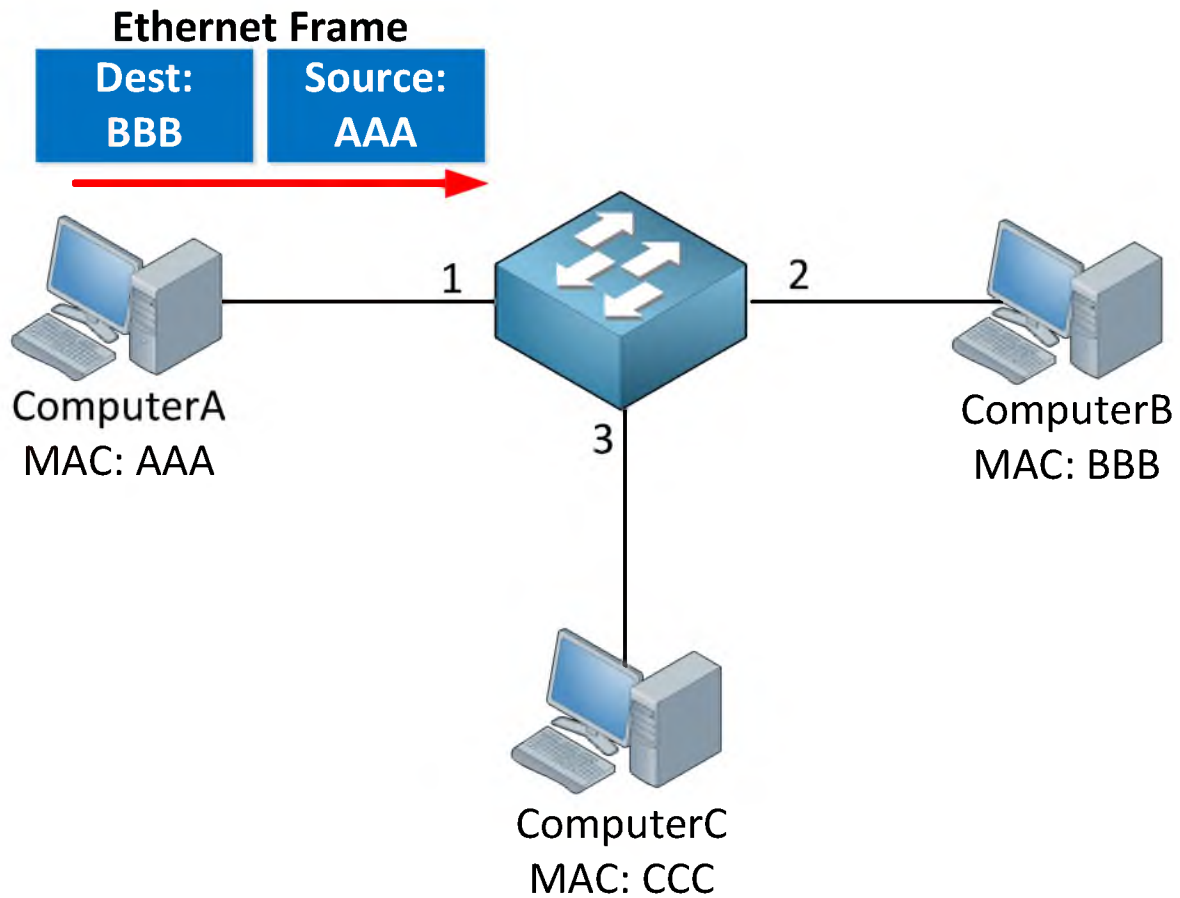


There's a switch in the middle and we have 3 computers. All computers have a MAC address but I've simplified them. Our switch has a MAC address table and it will learn where all the MAC addresses are in the network.

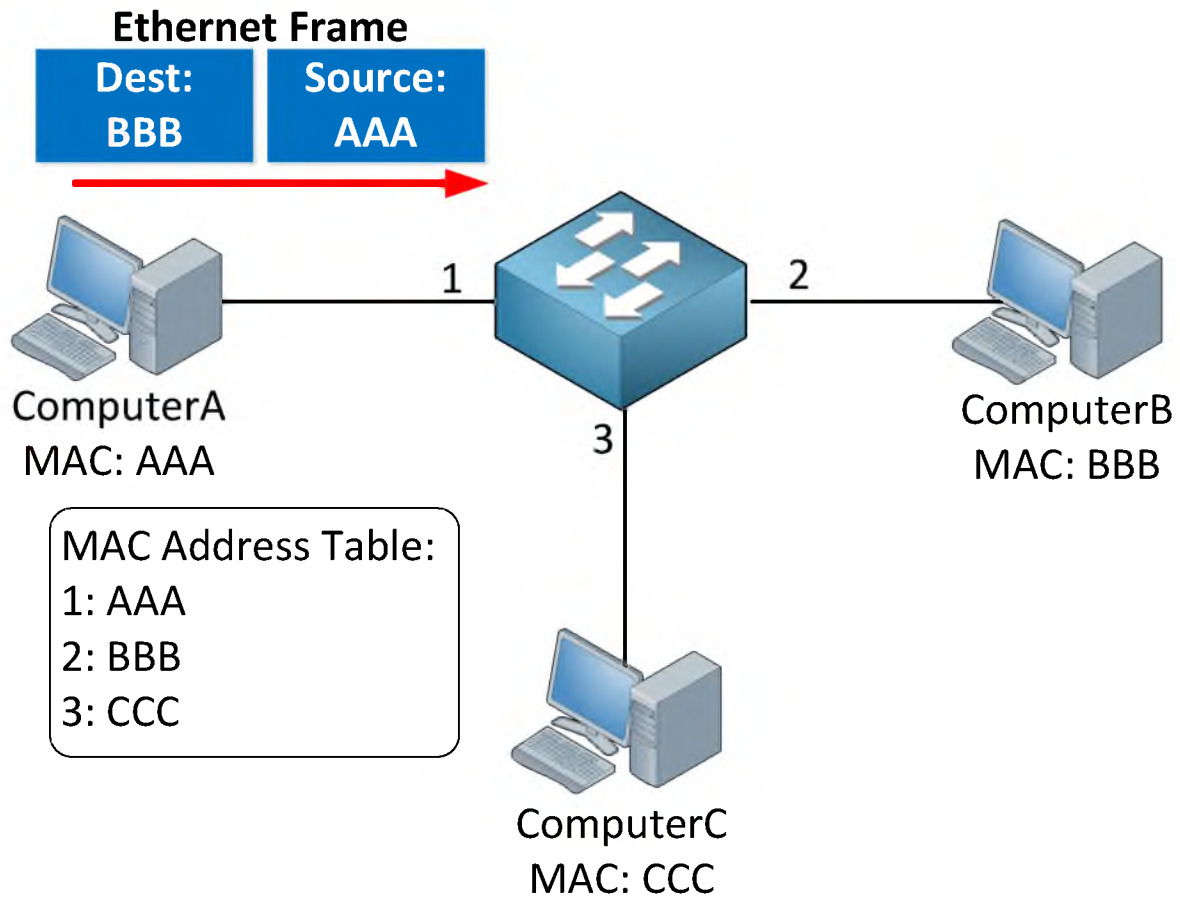
Question for you: how many collision domains do we have here?

Since we are running full-duplex we can't get any collisions in a switched network. **Every interface on a switch is a separate collision domain!** So why do we call each interface a separate collision domain if we can't get any collisions? Well if you connect a hub to one of our switch interfaces we can still get collisions there...

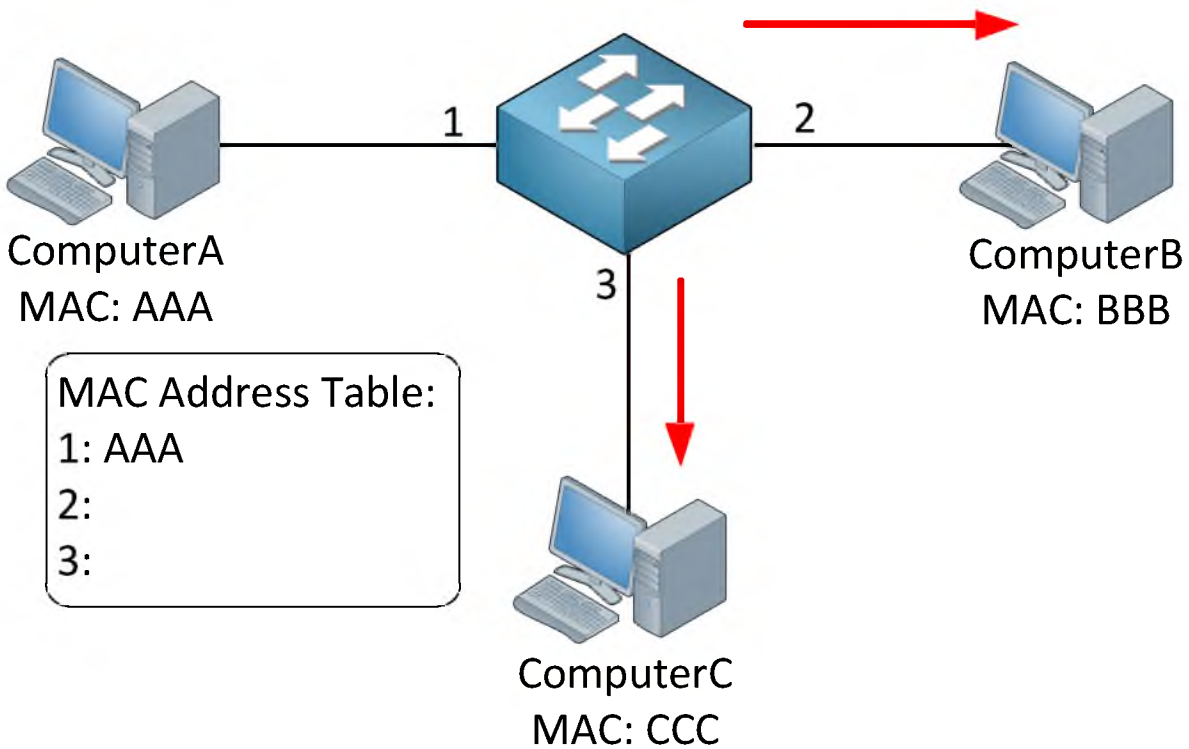
Since we are running full-duplex and we can't get any collisions anymore, our CSMA/CD protocol we talked about before is **disabled**.



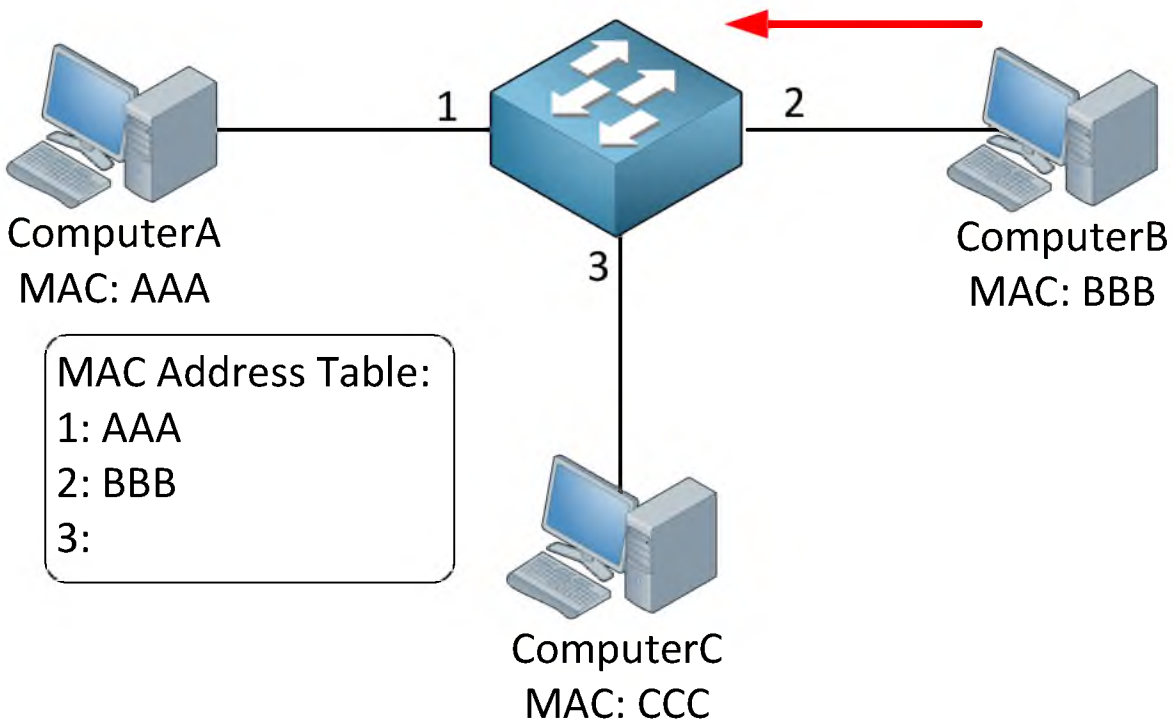
Computer A is going to send some data meant for computer B, thus it will create an Ethernet frame which has a source MAC address (AAA) and a destination MAC address (BBB).



Our switch will build a MAC address table and only **learns from source MAC addresses**. At this moment it just learned that the MAC address of computer A is on interface 1. It will now add this information in its MAC address table.



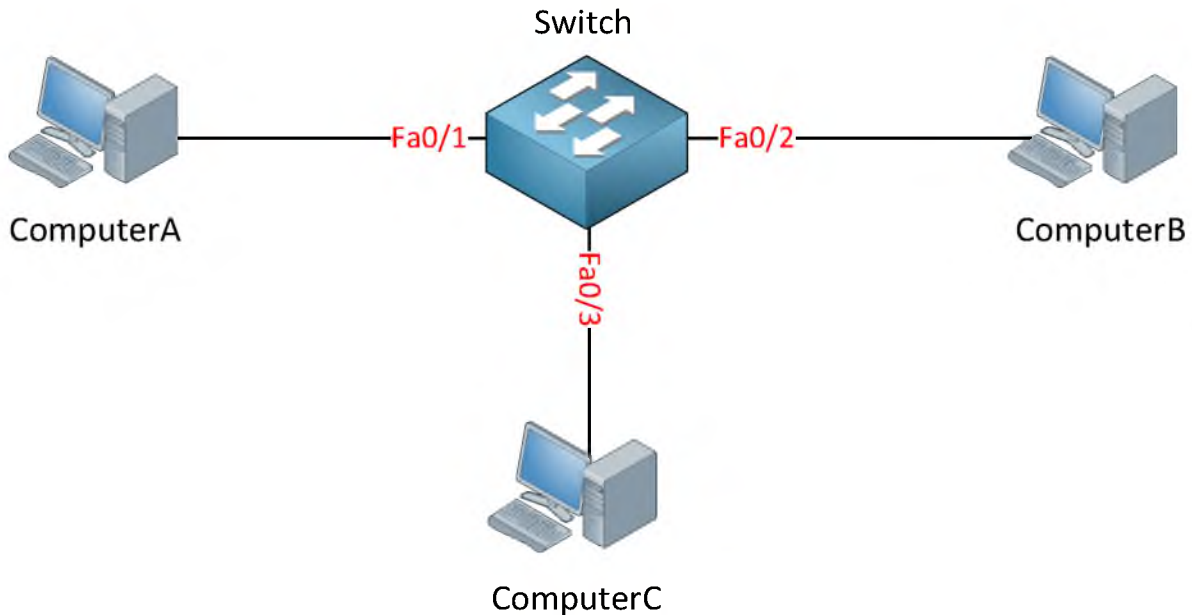
As you can see our switch currently has no information where computer B is located. There's only one option left....**flood** this frame out of all its interfaces except the one where it came from. computer B and computer C will receive this Ethernet frame.



Since computer B sees its MAC address as the destination of this Ethernet frame it knows it's meant for him, computer C will discard it. Computer B is going to respond to computer A, build an Ethernet frame and send it towards our switch. At this moment the switch will learn the MAC address of computer B.

That's the end of our story, the switch now knows both MAC addresses and the next time it can "switch" instead of flooding Ethernet frames. Computer C will never see any frames between computer A and B except for the first one which was flooded.

Let me show you what this looks like on a real Cisco switch:



This is the topology I'll use, it's the same as the previous example but I have added some interface numbers.

```

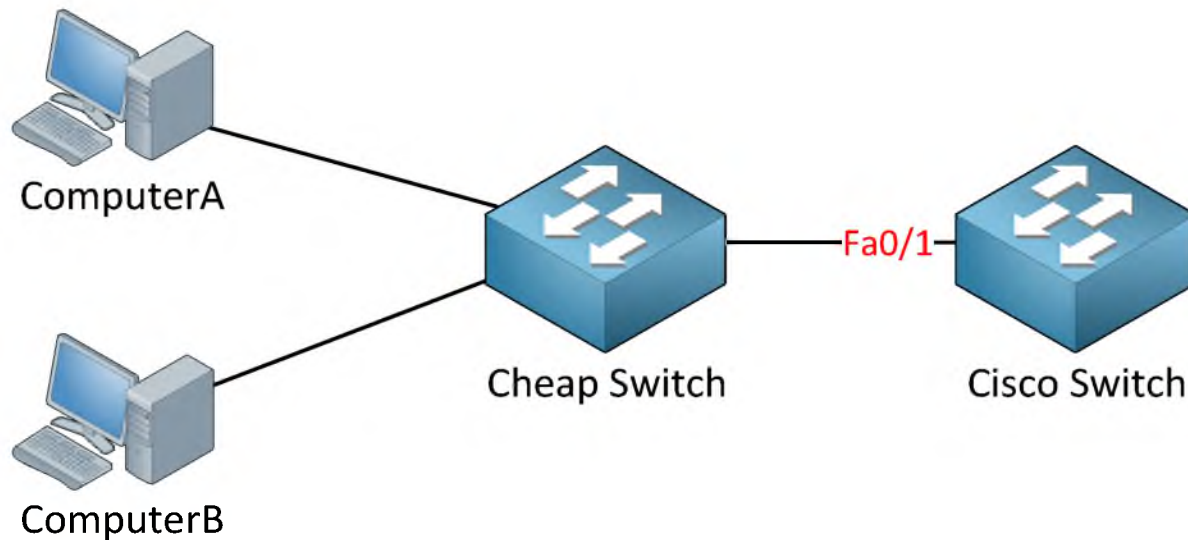
Switch#show mac address-table dynamic
      Mac Address Table
-----
Vlan  Mac Address      Type      Ports
----  -
1      000c.2928.5c6c      DYNAMIC   Fa0/1
1      000c.29e2.03ba      DYNAMIC   Fa0/2
1      000c.2944.0343      DYNAMIC   Fa0/3
  
```

Use the **show mac address-table dynamic** command to see all the MAC addresses that the switch has learned. You can see that it has learned the MAC addresses of Computer A, B and C.

By default there is no limit to the number of MAC addresses a switch can learn on an interface and all MAC addresses are allowed. If we want we can change this behavior with **port-security**.



Let's take a look at the following situation:



In the topology above someone connected a cheap switch that they brought from home to the FastEthernet 0/1 interface of our Cisco switch. Sometimes people like to bring an extra switch from home to the office. As a result our Cisco switch will learn the MAC address of ComputerA and ComputerB on its FastEthernet 0/1 interface.

Of course we don't want people to bring their own switches and connect it to our network so we want to prevent this from happening. This is how we can do it:

```
Switch(config)#interface fa0/1
Switch(config-if)#switchport port-security
Switch(config-if)#switchport port-security maximum 1
```

Use the **switchport port-security** command to enable port-security. I have configured port-security so only one MAC address is allowed. Once the switch sees another MAC address on the interface it will be in **violation** and something will happen. I'll show you what happens in a bit...

Besides setting a maximum on the number of MAC addresses we can also use port security to **filter** MAC addresses. You can use this to only allow certain MAC addresses. In the example above I configured port security so it only allows MAC address `aaaa.bbbb.cccc`. This is not the MAC address of my computer so it's perfect to demonstrate a violation.

```
Switch(config)#interface fa0/1
Switch(config-if)#switchport port-security mac-address aaaa.bbbb.cccc
```

Use the **switchport port-security mac-address** command to define the MAC address that you want to allow. Now we'll generate some traffic to cause a violation:

```
C:\Documents and Settings\ComputerA>ping 1.2.3.4
```

I'm pinging to some bogus IP address...there is nothing that has IP address 1.2.3.4; I just want to generate some traffic.



Here's what you will see:

```
SwitchA#
%PM-4-ERR_DISABLE: psecure-violation error detected on Fa0/1, putting Fa0/1
in err-disable state
%PORT_SECURITY-2-PSECURE_VIOLATION: Security violation occurred, caused by
MAC address 0090.cc0e.5023 on port FastEthernet0/1.
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed
state to down
%LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to down
```

Banzai! We have a security violation and as a result the port goes in **err-disable state**. As you can see it is now down. Let's take a closer look at port-security:

```
Switch#show port-security interface fa0/1
Port Security           : Enabled
Port Status             : Secure-shutdown
Violation Mode          : Shutdown
Aging Time              : 0 mins
Aging Type              : Absolute
SecureStatic Address Aging : Disabled
Maximum MAC Addresses   : 1
Total MAC Addresses     : 1
Configured MAC Addresses : 1
Sticky MAC Addresses    : 0
Last Source Address:Vlan : 0090.cc0e.5023:1
Security Violation Count : 1
```

Here is a useful command to check your port security configuration. Use **show port-security interface** to see the port security details per interface. You can see the violation mode is shutdown and that the last violation was caused by MAC address 0090.cc0e.5023 (ComputerA). The **aging time** is 0 mins which means it will stay in err-disable state forever.

```
Switch#show interfaces fa0/1
FastEthernet0/1 is down, line protocol is down (err-disabled)
```

Shutting the interface after a security violation is a good idea (security-wise) but the problem is that the interface will **stay in err-disable state**. This probably means another call to the helpdesk and *you* bringing the interface back to the land of the living! Let's activate it again:

```
Switch(config)#interface fa0/1
Switch(config-if)#shutdown
Switch(config-if)#no shutdown
```

To get the interface out of err-disable state you need to type "shutdown" followed by "no shutdown". Only typing "no shutdown" is **not enough!**

It might be easier if the interface could recover itself after a certain time:

```
Switch(config)#errdisable recovery cause psecure-violation
Switch(config)#interface fa0/1
Switch(config-if)#switchport port-security aging time 10
```

You can change the aging time from 0 to whatever value you like with the **switchport port-security aging time** command.

After 10 minutes it will automatically recover from err-disable state. Make sure you solve the problem though because otherwise it will just have another violation and end up in err-disable state again. Make sure you don't forget to enable automatic recovery with the **errdisable recovery cause psecure-violation** command.

Instead of typing in the MAC address ourselves we can also make the switch learn a MAC address for port-security:

```
Switch(config-if)#no switchport port-security mac-address aaaa.bbbb.cccc
Switch(config-if)#switchport port-security mac-address sticky
```

The **sticky** keyword will make sure that the switch uses the first MAC address that it learns on the interface for port-security. Let's verify it:

```
Switch#show run interface fa0/1
Building configuration...

Current configuration : 228 bytes
!
interface FastEthernet0/1
 switchport mode access
 switchport port-security
 switchport port-security aging time 10
 switchport port-security mac-address sticky
 switchport port-security mac-address sticky 000c.2928.5c6c
```

You can see that it will save the MAC address of ComputerA in the running-configuration by itself.

Shutting the interface in case of a violation might be a bit too much. There are other options, here's what you can do:

```
Switch(config-if)#switchport port-security violation ?
protect      Security violation protect mode
restrict     Security violation restrict mode
shutdown     Security violation shutdown mode
```

There are other options like **protect** and **restrict**.

- **Protect:** Ethernet frames from MAC addresses that are not allowed will be dropped but you won't receive any logging information.
- **Restrict:** Ethernet frames from MAC addresses that are not allowed will be dropped but you will see logging information and a SNMP trap is sent.

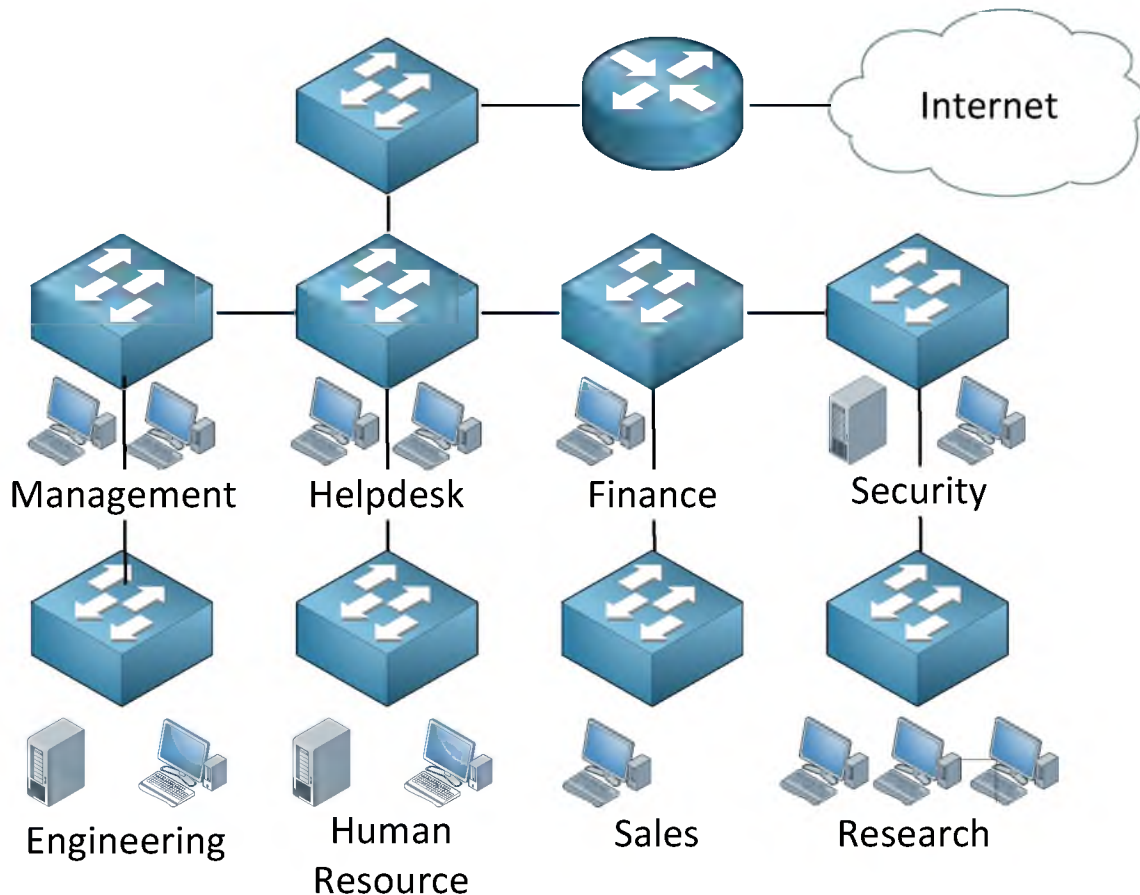
- **Shutdown:** Ethernet frames from MAC addresses that are not allowed will cause the interface to go to err-disable state. You will see logging information and a SNMP trap is sent. For recovery you have two options:
  - Manual: The default aging time is 0 mins so you'll have to enable the interface yourself.
  - Automatic: Configure the aging time to another value.

That's all I wanted to show you about port-security.

Are you having fun yet? There's more to switching...we'll talk about VLANS (Virtual LANs), Trunks and spanning tree later in the next chapter!

## 9. Virtual LANs (VLANs), Trunks and VTP

We talked about switches before and now we will further enhance your knowledge and introduce VLANs, trunks and VTP (VLAN Trunking Protocol). Let's start off by showing you a picture of a network:



Look at this picture for a minute, we have many departments and every department has its own switch. Users are grouped physically together and are connected to their switch. What do you think of it? Does this look like a good network design? If you are unsure let me ask you some questions to think about:

- What happens when a computer connected to the Research switch sends a broadcast like an ARP request?
- What happens when the Helpdesk switch fails?
- Will our users at the Human Resource switch have fast network connectivity?
- How can we implement security in this network?

Now tell me explain you why this is a bad network design. If any of our computers sends a broadcast what will our switches do? They flood it! This means that a single broadcast frame will be flooded on this entire network. This also happens when a switch hasn't learned about a certain MAC address, the frame will be flooded.

If our helpdesk switch would fail this means that users from Human Resource are “isolated” from the rest and unable to access other departments or the internet, this applies to other switches as well. Everyone has to go through the Helpdesk switch in order to reach the Internet which means we are sharing bandwidth, probably not a very good idea performance-wise.

Last but not least, what about security? We could implement port-security and filter on MAC addresses but that’s not a very secure method since MAC addresses are very easy to spoof. VLANs are one way to solve our problems.

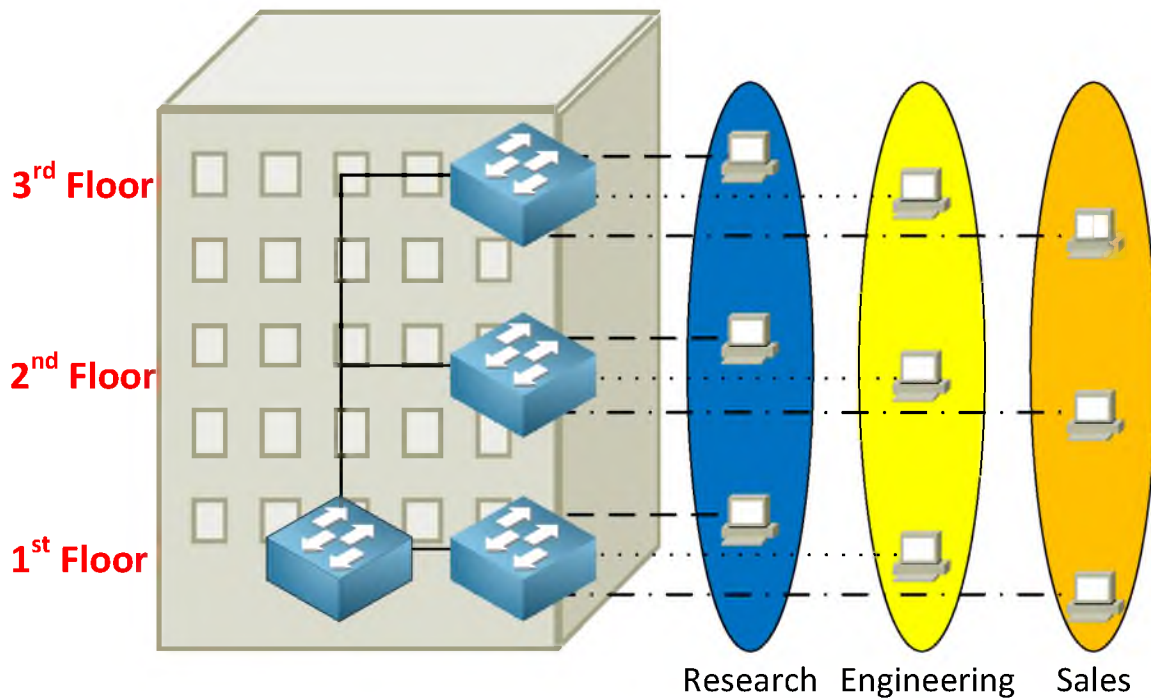
Two more questions I’d like to ask you to refresh your knowledge:

- How many collision domains do we have here?
- How many broadcast domains do we have here?

Remember our collision domains when we talked about hubs, bridges and switches? I told you that each port on a switch is a separate collision domain so in this picture we have a LOT of collision domains...more than 20.

What about broadcast domains? We didn’t talk about this before but I think you can answer it. If a computer from the sales switch would send a broadcast frame we know that all other switches will forward it. Did you spot the router on top of the picture? What about it...do you think a router will forward a broadcast frame?

The answer is that routers don’t forward broadcast frames so they effectively “limit” our broadcast domain. Of course on the right side of our router where we have an Internet connection this would be another broadcast domain...so we have 2 broadcast domains here.



When you work with switches you have to keep in mind there’s a big difference between physical and logical topology. Physical is just the way our cables are connected while logical is how we have setup things ‘virtually’. In the example above we have 4 switches and I

have created 3 VLANs called Research, Engineering and Sales. A VLAN is a Virtual LAN so it's like having a "switch inside a switch".

What are the advantages of using VLANs?

- A VLAN is a single broadcast domain which means that if a user in the research VLAN would send a broadcast frame only users in the same VLAN will receive it.
- Users are only able to communicate within the same VLAN (more on this later).
- Users don't have to be grouped physically together, as you can see we have users in the Engineering VLAN sitting on the 1<sup>st</sup>, 2<sup>nd</sup> and 3<sup>rd</sup> floor.

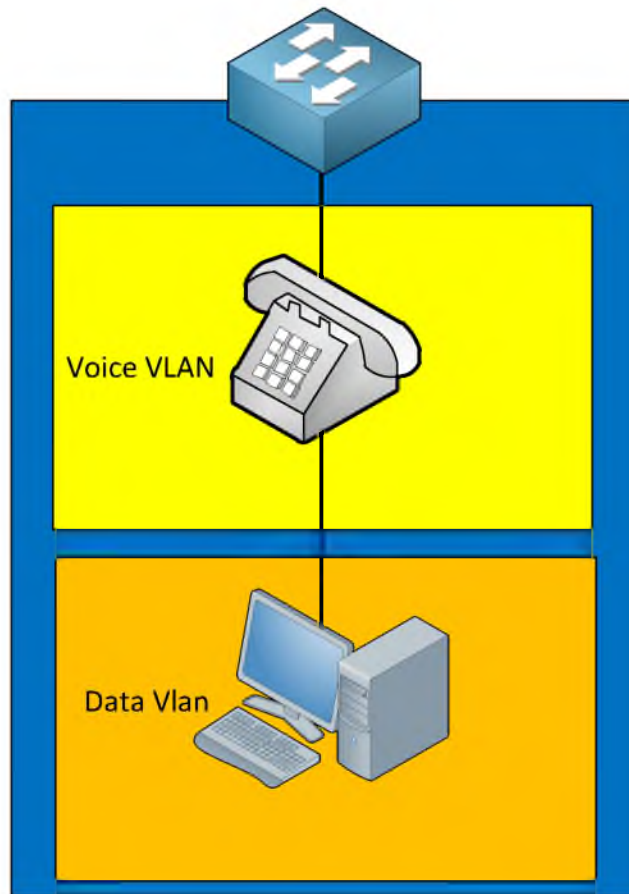
In my example I grouped different users in different VLANs but you can also use VLANs to separate different traffic types. Perhaps you want to have all printers in one VLAN, all servers in a VLAN and all the computers in another. What about VoIP? Put all your Voice over IP phones in a separate VLAN so its traffic is separated from other data.

If you have ever seen a Cisco IP Phone you might have noticed it has two interfaces:



There's one to connect your IP Phone to the switch and another one to connect your computer to the phone. This will save you a port on your switch and the need to get another cable to your switch, but there's also something else that's pretty neat.

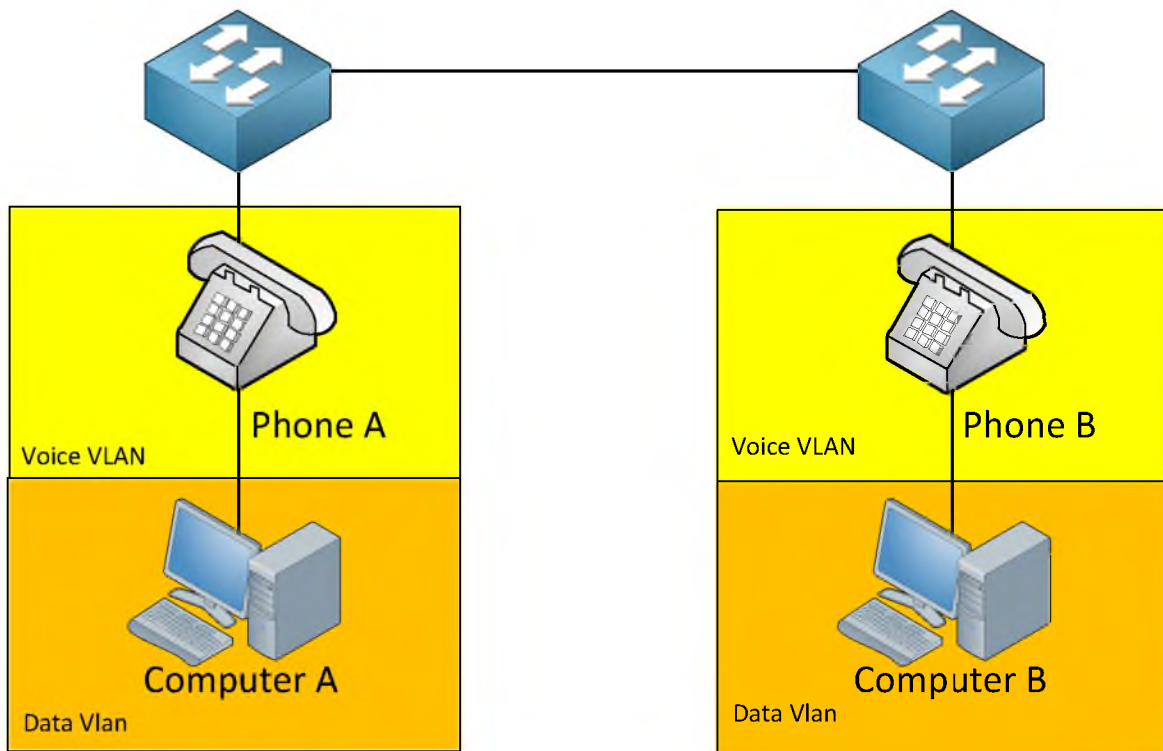
Your IP Phone will be in the Voice VLAN and the computer will be in the Data VLAN, so you only have one cable from the switch to your phone but traffic will be nicely separated by using VLANs.



This also gives us the possibility to use Quality of Service (QOS). This is a topic which is not on the CCNA but very important if you deal with VoIP, it will let you set priorities for certain traffic types. VoIP is very sensitive to delay and jitter (jitter is a variation in delay) and in order to make things smooth you need to make sure VoIP traffic will always be prioritized in case of congestion of your network, this is what QOS takes care of.



Now you have seen some of the advantages that VLANs can give us but this leaves us with one big question, take a look at the following picture:



So we have two switches and there's a voice VLAN and a data VLAN. What happens when Phone A wants to call Phone B? Or if computer A wants to send something to computer B?

Our switches will forward traffic but how do they know to which VLAN our traffic belongs?

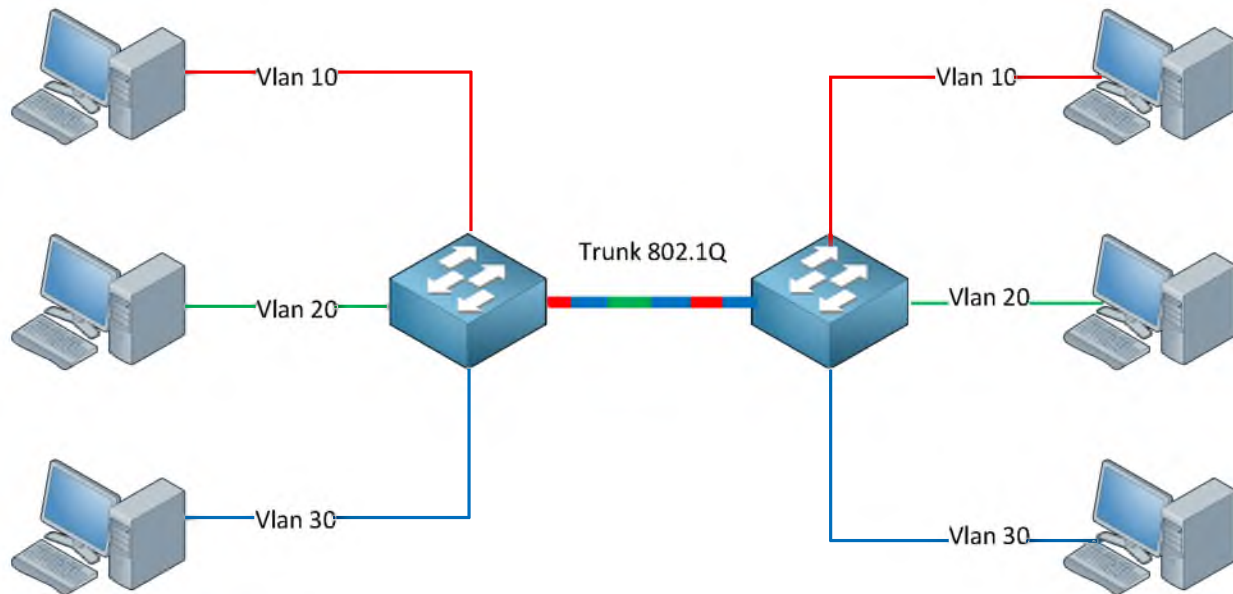
Let's take another look at an Ethernet frame:



Do you see any field where we can specify to which VLAN our Ethernet frame belongs? Well there isn't! That's why we need another protocol to help us.

Between switches we are going to create a **trunk**. A trunk connection is simply said nothing more but a normal link but it carries all VLAN traffic.



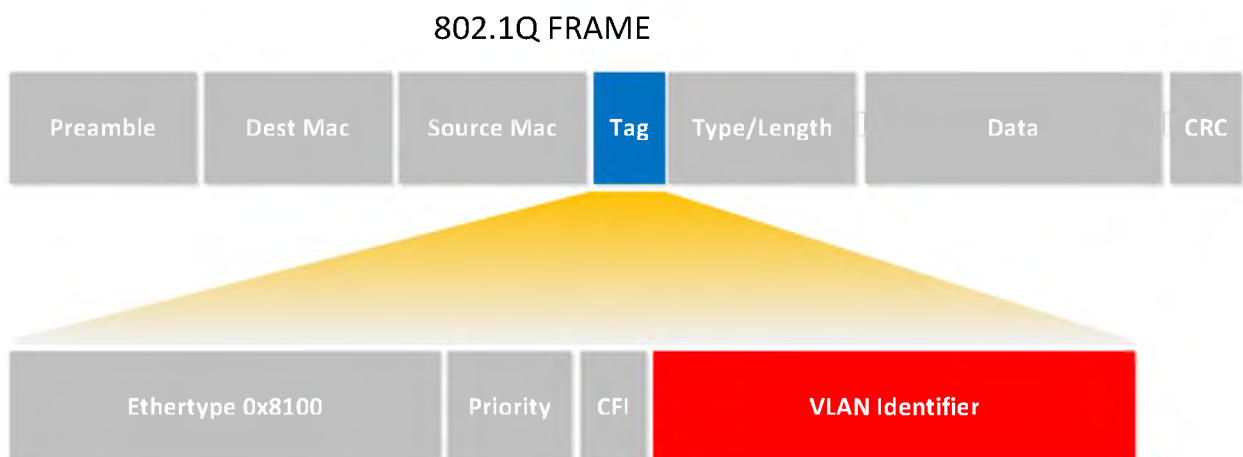


As you can see we have computers on both sides and they are in different VLANs, by using trunks we can make sure all VLAN traffic can be sent between the switches. Because our regular Ethernet frames don't have anything to show to which VLAN they belong we will need another protocol.

There are two trunking protocols:

- **802.1Q:** This is the most common trunking protocol. It's a standard and supported by many vendors.
- **ISL:** This is the Cisco trunking protocol. Not all switches support it.

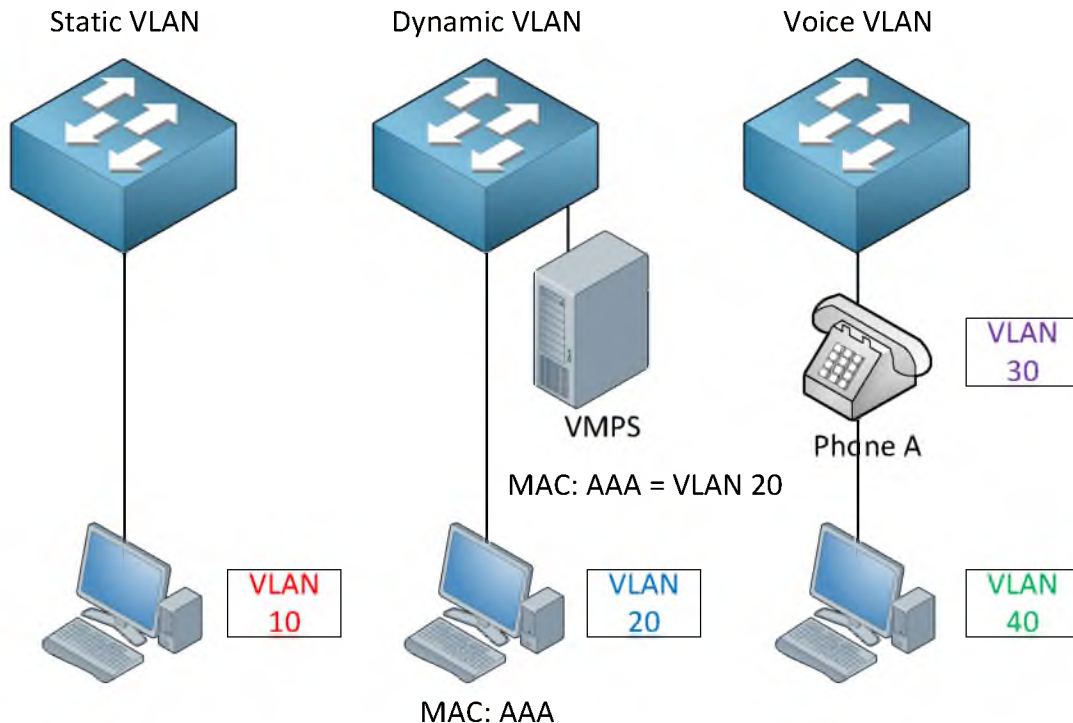
Let's take a look at 802.1Q:



Here's an example of an 802.1Q Ethernet frame. As you can see it's the same as a normal Ethernet frame but we have added a tag in the middle (that's the blue field). In our tag you will find a "VLAN identifier" which is the VLAN to which this Ethernet frame belongs. This is how switches know to which VLAN our traffic belongs. That's not too bad right? There's also

a field called "Priority" which is how we can give a different priority to the different types of traffic.

VLANs and Trunks...if you are following me so far you understand the basics, very good! Let's take a look at the options we have to configure VLANs:



Static VLAN is the most common method; you just configure the VLAN yourself on the interface. Dynamic VLAN is where you have a VMPS server (VLAN Management Policy Server) which has a database of MAC address – VLAN information. It will check the MAC address of the computer and assign you the VLAN that it found in its database. Is this a good idea? Probably not since MAC addresses are easy to spoof. The third option is the voice VLAN which has to be configured separately on a Cisco switch. The link between the switch and the phone is actually a trunk!

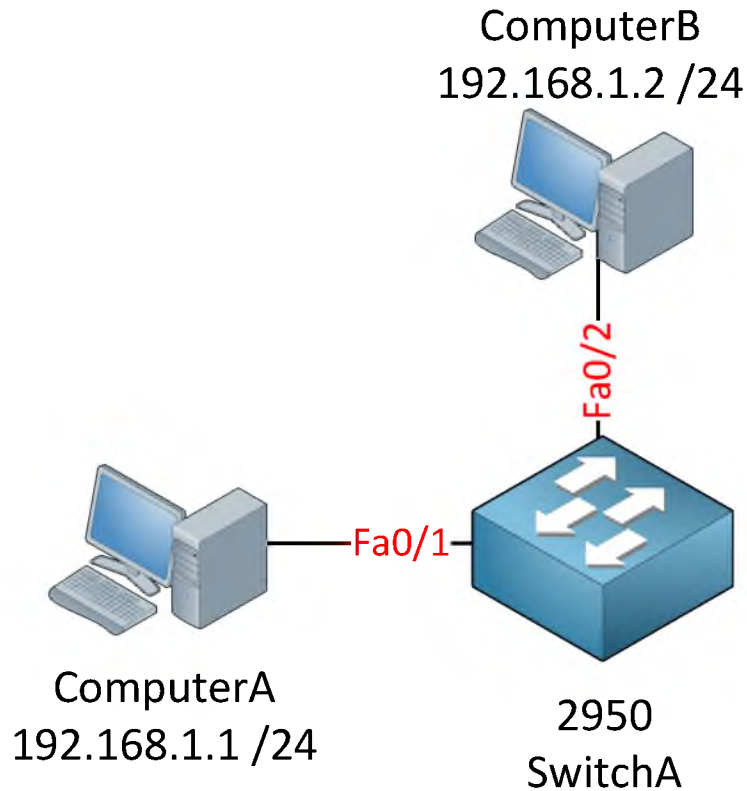
There is a 4<sup>th</sup> method which is popular nowadays, you can use 802.1X and a RADIUS server to authenticate users and dynamically assign users to a VLAN. This gets even more interesting by adding NAC (Network Admission Control) to it. If your laptop doesn't have all the latest Windows Updates and Anti-virus definitions you will be assigned to a special quarantine VLAN where you can only update your MACHINE, once you are updated you will be moved to the correct VLAN. In the wireless chapter I talk a bit more about 802.1X and Radius.

Back to our trunks...every VLAN that goes across the trunk will be tagged using the 802.1Q protocol but there is one exception. The **native VLAN** is the only VLAN that will not be tagged. That's right it will be using regular Ethernet frames. So what do we use the native VLAN for?

- Management protocols like CDP (Cisco Discovery Protocol) use the native VLAN.
- Remote management to your Cisco switch uses the native VLAN.

- The default native VLAN is VLAN 1.

Now you have an idea what VLANs, trunks and VTP are about. Let's take a look what this all looks like on our real switches:



Let's start with a simple example. ComputerA and ComputerB are connected to SwitchA.

First we will look at the default VLAN configuration on SwitchA:

```
SwitchA#show vlan
```

| VLAN | Name               | Status    | Ports   |
|------|--------------------|-----------|---|
| 1    | default            | active    | Fa0/1, Fa0/2, Fa0/3, Fa0/4<br>Fa0/5, Fa0/6, Fa0/7, Fa0/8<br>Fa0/9, Fa0/10, Fa0/12<br>Fa0/13, Fa0/14, Fa0/22<br>Fa0/23, Fa0/24, Gi0/1, Gi0/2 |
| 1002 | fddi-default       | act/unsup |   |
| 1003 | token-ring-default | act/unsup |   |
| 1004 | fddinet-default    | act/unsup |   |
| 1005 | trnet-default      | act/unsup |   |

Interesting...VLAN 1 is the default LAN and you can see that all active interfaces are assigned to VLAN 1.



*VLAN information is not saved in the running-config or startup-config but in a separate file called `vlan.dat` on your flash memory. If you want to delete the VLAN information you should delete this file by typing `delete flash:vlan.dat`. I configured an IP address on ComputerA and ComputerB so they are in the same subnet.*

Let's see if ComputerA and ComputerB can reach each other:

```
C:\Documents and Settings\ComputerA>ping 192.168.1.2
```

```
Pinging 192.168.1.2 with 32 bytes of data:
```

```
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
```

```
Ping statistics for 192.168.1.2:
```

```
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

```
Approximate round trip times in milli-seconds:
```

```
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Even with the default switch configuration ComputerA is able to reach ComputerB. Let's see if I can create a new VLAN for ComputerA and ComputerB:

```
SwitchA(config)#vlan 50
SwitchA(config-vlan)#name Computers
SwitchA(config-vlan)#exit
```

This is how you create a new VLAN. If you want you can give it a name but this is optional. I'm calling my VLAN "Computers".

```
SwitchA#show vlan
```

| VLAN | Name      | Status | Ports  |
|------|-----------|--------|--|
| 1    | default   | active | Fa0/1, Fa0/2, Fa0/3, Fa0/4<br>Fa0/5, Fa0/6, Fa0/7, Fa0/8<br>Fa0/9, Fa0/10, Fa0/11, Fa0/12<br>Fa0/13, Fa0/14, Fa0/15,<br>Fa0/23, Fa0/24, Gi0/1, Gi0/2 |
| 50   | Computers | active |  |

VLAN 50 was created on SwitchA and you can see that it's active. However no ports are currently in VLAN 50. Let's see if we can change this...

```
SwitchA(config)#interface fa0/1
SwitchA(config-if)#switchport mode access
SwitchA(config-if)#switchport access vlan 50
```

```
SwitchA(config)interface fa0/2
SwitchA(config-if)#switchport mode access
SwitchA(config-if)#switchport access vlan 50
```

First I will configure the switchport in **access mode** with the **switchport mode access command**. By using the **switchport access vlan** command we can move our interfaces to another VLAN.

```
SwitchA#show vlan
```

| VLAN | Name      | Status | Ports  |
|------|-----------|--------|--|
| 1    | default   | active | Fa0/3, Fa0/4<br>Fa0/5, Fa0/6, Fa0/7, Fa0/8<br>Fa0/9, Fa0/10,, Fa0/12<br>Fa0/13, Fa0/14, Fa0/15,<br>Fa0/23, Fa0/24, Gi0/2 |
| 50   | Computers | active | Fa0/1, Fa0/2   |

Excellent! Both computers are now in VLAN 50. Let's verify our configuration by checking if they can ping each other:

```
C:\Documents and Settings\ComputerA>ping 192.168.1.2
```

```
Pinging 192.168.1.2 with 32 bytes of data:
```

```
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
```

```
Ping statistics for 192.168.1.2:
```

```
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

```
Approximate round trip times in milli-seconds:
```

```
Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Our computers are able to reach each other within VLAN 50. Besides pinging each other we can also use another show command to verify our configuration:

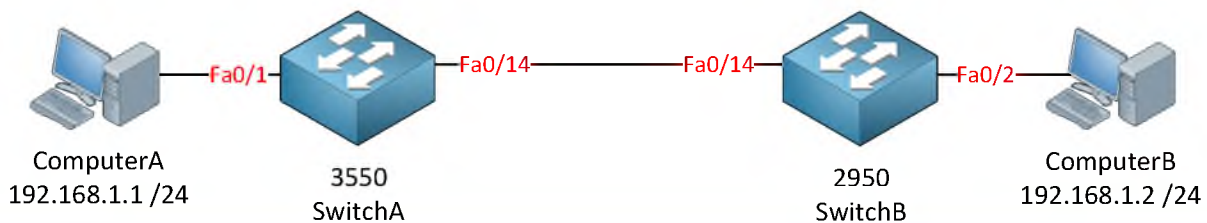
```
SwitchA#show interfaces fa0/1 switchport
```

```
Name: Fa0/1
Switchport: Enabled
Administrative Mode: static access
Operational Mode: static access
Administrative Trunking Encapsulation: negotiate
Operational Trunking Encapsulation: native
Negotiation of Trunking: Off
Access Mode VLAN: 50 (Computers)
Trunking Native Mode VLAN: 1 (default)
```

```
SwitchA#show interfaces fa0/2 switchport
Name: Fa0/1
Switchport: Enabled
Administrative Mode: static access
Operational Mode: static access
Administrative Trunking Encapsulation: negotiate
Operational Trunking Encapsulation: native
Negotiation of Trunking: Off
Access Mode VLAN: 50 (Computers)
Trunking Native Mode VLAN: 1 (default)
```

By using the "show interfaces switchport" command we can see that the **operational mode** is "static access" which means it's in access mode. We can also verify that the interface is assigned to VLAN 50.

Let's continue our VLAN adventure by adding SwitchB to the topology. I also moved ComputerB from SwitchA to SwitchB.



```
SwitchB(config)#vlan 50
SwitchB(config-vlan)#name Computers
SwitchB(config-vlan)#exit
```

```
SwitchB(config)#interface fa0/2
SwitchB(config-if)#switchport access vlan 50
```

I just created VLAN 50 on SwitchB and the interface connected to ComputerB is assigned to VLAN 50.

Next step is to create a trunk between SwitchA and SwitchB:

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode trunk
Command rejected: An interface whose trunk encapsulation is "Auto" can not be configured to "trunk" mode.
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport mode trunk
Command rejected: An interface whose trunk encapsulation is "Auto" can not be configured to "trunk" mode.
```

I try to change the interface to trunk mode with the **switchport mode trunk** command. Depending on the switch model you might see the same error as me. If we want to change the interface to trunk mode we need to change the trunk encapsulation type.

Let's see what options we have:

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

Aha...so this is where you can choose between 802.1Q and ISL. By default our switch will negotiate about the trunk encapsulation type.

```
SwitchA(config-if)#switchport trunk encapsulation dot1q
```

```
SwitchB(config-if)#switchport trunk encapsulation dot1q
```

Let's change it to 802.1Q by using the **switchport trunk encapsulation** command.

```
SwitchA#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
Administrative Trunking Encapsulation: dot1q
```

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
Administrative Trunking Encapsulation: dot1q
```

As you can see the trunk encapsulation is now 802.1Q.

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode trunk
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport mode trunk
```

Now I can successfully change the switchport mode to trunk.

```
SwitchA#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
Administrative Trunking Encapsulation: dot1q
Operational Trunking Encapsulation: dot1q
```

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
```



```
Administrative Mode: trunk
Operational Mode: trunk
Administrative Trunking Encapsulation: dot1q
Operational Trunking Encapsulation: dot1q
```

We can confirm we have a trunk because the operational mode is "dot1q".

Let's try if ComputerA and ComputerB can reach each other:

```
C:\Documents and Settings\ComputerA>ping 192.168.1.2
```

Pinging 192.168.1.2 with 32 bytes of data:

```
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
```

Ping statistics for 192.168.1.2:

**Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),**

Approximate round trip times in milli-seconds:

Minimum = 0ms, Maximum = 0ms, Average = 0ms

Excellent! ComputerA and ComputerB can reach each other! Does this mean we are done? Not quite yet...there's more I want to show to you:

```
SwitchB#show vlan
```

| VLAN | Name      | Status | Ports  |
|------|-----------|--------|--|
| 1    | default   | active | Fa0/1, Fa0/3, Fa0/4, Fa0/5<br>Fa0/6, Fa0/7, Fa0/8, Fa0/9<br>Fa0/10, Fa0/11, Fa0/12,<br>Fa0/13<br>Fa0/15, Fa0/22, Fa0/23,<br>Fa0/24<br>Gi0/1, Gi0/2 |
| 50   | Computers | active | <b>Fa0/2</b>   |

First of all, if we use the show vlan command we don't see the Fa0/14 interface. This is completely normal because the show vlan command **only shows interfaces in access mode and no trunk interfaces**.

```
SwitchB#show interface fa0/14 trunk
```

| Port   | Mode | Encapsulation | Status   | Native vlan |
|--------|------|---------------|----------|-------------|
| Fa0/14 | on   | 802.1q        | trunking | 1           |

| Port   | Vlans allowed on trunk |
|--------|------------------------|
| Fa0/14 | 1-4094                 |

| Port   | Vlans allowed and active in management domain |
|--------|---|
| Fa0/14 | 1,50  |



```
Port          Vlans in spanning tree forwarding state and not pruned
Fa0/14        50
```

The **show interface trunk** command is very useful. You can see if an interface is in trunk mode, which trunk encapsulation protocol it is using (802.1Q or ISL) and what the native VLAN is. We can also see that VLAN 1 – 4094 are allowed on this trunk.

For security reasons you might want to limit the VLANs that are allowed on the trunk, we can do this with the **allowed** keyword:

```
SwitchB(config-if)#switchport trunk allowed vlan remove 1-4094
SwitchB(config-if)#switchport trunk allowed vlan add 1-50
```

This will remove VLAN 1-4094 from the trunk and only allows VLAN 1-50.

```
SwitchB#show interface fa0/14 trunk

Port          Mode          Encapsulation  Status        Native vlan
Fa0/14        on             802.1q         trunking      1

Port          Vlans allowed on trunk
Fa0/14        1-50

Port          Vlans allowed and active in management domain
Fa0/14        1,50

Port          Vlans in spanning tree forwarding state and not pruned
Fa0/14        50
```

As you can see only VLAN 1-50 is now allowed.

We can also see that currently only VLAN 1 (native VLAN) and VLAN 50 are active. Last but not least you can see something which VLANs are in the forwarding state for spanning-tree (more on spanning-tree later!).

Before we continue with the configuration of VTP I want to show you one more thing about access and trunk interfaces:

```
SwitchB#show interface fa0/2 switchport
Name: Fa0/2
Switchport: Enabled
Administrative Mode: static access
Operational Mode: static access
```

An interface can be in access mode or in trunk mode. The interface above is connected to ComputerB and you can see that the operational mode is "static access" which means it's in access mode.

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
```

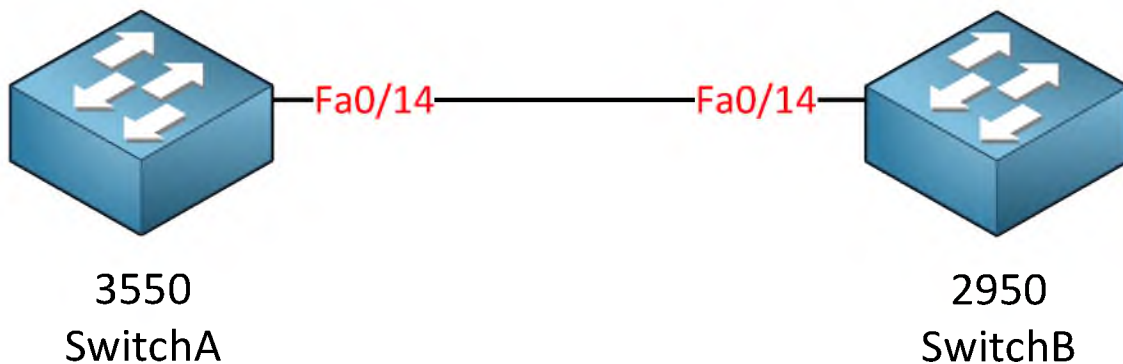
This is our trunk interface which is connected to SwitchA. You can see the operational mode is trunk mode.

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

If I go to the interface configuration to change the switchport mode you can see I have more options than access or trunk mode. There is also a **dynamic** method. Don't worry about the other options for now.

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

We can choose between **dynamic auto** and **dynamic desirable**. Our switch will automatically find out if the interface should become an access or trunk port. So what's the difference between dynamic auto and dynamic desirable? Let's find out!



I'm going to play with the switchport mode on SwitchA and SwitchB and we'll see what the result will be.

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode dynamic auto
```

```
SwitchA(config)#interface fa0/14
SwitchB(config-if)#switchport mode dynamic auto
```

First I'll change both interfaces to dynamic auto.

```
SwitchA(config-if)#do show interface f0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
```

```
SwitchB(config-if)#do show interface f0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
```

Our administrative mode is dynamic auto and as a result we now have an access port.

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode dynamic desirable
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport mode dynamic desirable
```

```
SwitchA#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
```

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
```

Once we change both interfaces to dynamic desirable we end up with a trunk link. What do you think will happen if we mix the switchport types? Maybe dynamic auto on one side and dynamic desirable on the other side? Let's find out!

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode dynamic desirable
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport mode dynamic auto
```

```
SwitchA#show interfaces f0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
```

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic auto
```

```
Operational Mode: trunk
```

It seems our switch has a strong desire to become a trunk. Let's see what happens with other combinations!

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode dynamic auto
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport mode trunk
```

```
SwitchA#show interfaces f0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: trunk
```

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
```

Dynamic auto will prefer to become an access port but if the other interface has been configured as trunk we will end up with a trunk.

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode dynamic auto
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport mode access
```

```
SwitchA#show interfaces f0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
```

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: static access
Operational Mode: static access
```

Configuring one side as dynamic auto and the other one as access and the result will be an access port.

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode dynamic desirable
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport mode trunk
```

```
SwitchA#show interfaces f0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
```

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
```

Dynamic desirable and trunk mode offers us a working trunk.

What do you think will happen if I set one interface in access mode and the other one as trunk? Doesn't sound like a good idea but let's push our luck:

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport mode access
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport mode trunk
```

```
SwitchA#show interfaces f0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: static access
Operational Mode: trunk
```

```
SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
```

```
SwitchA#
%SPANTREE-7-RECV_1Q_NON_TRUNK: Received 802.1Q BPDU on non trunk
FastEthernet0/14 VLAN1.
%SPANTREE-7-BLOCK_PORT_TYPE: Blocking FastEthernet0/14 on VLAN0001.
Inconsistent port type.
%SPANTREE-2-UNBLOCK_CONSIST_PORT: Unblocking FastEthernet0/14 on VLAN0001.
Port consistency restored.
```

As soon as I change the switchport mode I see these spanning-tree error messages on SwitchA. Spanning-tree is a protocol that runs on switches that prevents loops in our network. Don't worry about it now as you will learn about it in the next chapter.

Let me give you an overview of the different switchport modes and the result:

|                   | Trunk   | Access  | Dynamic Auto | Dynamic Desirable |
|-------------------|---------|---------|--------------|-------------------|
| Trunk             | Trunk   | Limited | Trunk        | Trunk             |
| Access            | Limited | Access  | Access       | Access            |
| Dynamic Auto      | Trunk   | Access  | Access       | Trunk             |
| Dynamic Desirable | Trunk   | Access  | Trunk        | Trunk             |

Make sure you know the result of these combinations if you plan to do the CCNA exam. I always like to think that the switch has a strong "desire" to become a trunk. Its wish will always be granted unless the other side has been configured as access port. The "A" in dynamic auto stands for "Access", it would like to become an access port but only if the other side also is configured as dynamic auto or access mode.

I recommend **never to use** the "dynamic" types. I want my interfaces to be in trunk OR access mode and I like to make the decision myself. Keep in mind that dynamic auto is the **default** on most modern switches which means it's possible to form a trunk with any interface on your switch automatically. Some of the older switches use dynamic desirable as the default. This is a security issue you should deal with! If I walk into your company building I could connect my laptop to any wall jack, boot up GNS3, form a trunk to your switch and I'll have access to all your VLANs...doesn't sound like a good idea right?

This is what I recommend for trunk interfaces:

```
Switch(config-if)#switchport mode trunk
Switch(config-if)#switchport nonegotiate
```

The negotiation of the switchport status by using dynamic auto or dynamic desirable is called **DTP (Dynamic Trunking Protocol)**. You can disable it completely by using the **switchport nonegotiate** command.

This is what I would do for access interfaces:

```
Switch(config-if)#switchport mode access
Switch(config-if)#switchport nonegotiate
```

For security reasons it's a good idea to disable the negotiation of the switchport status and to configure the interface in access mode yourself.

One last thing about VLANs and trunks before we continue with VTP. I recommend changing the native VLAN to something else.

```
SwitchB#show interfaces fa0/14 trunk
```

| Port   | Mode | Encapsulation | Status   | Native vlan |
|--------|------|---------------|----------|-------------|
| Fa0/14 | on   | 802.1q        | trunking | 1           |

You can see that VLAN 1 is the default native VLAN on Cisco switches. Management protocols use this native VLAN so for security reasons it might be a good idea to change it into something else:

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport trunk native vlan 10
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport trunk native vlan 10
```

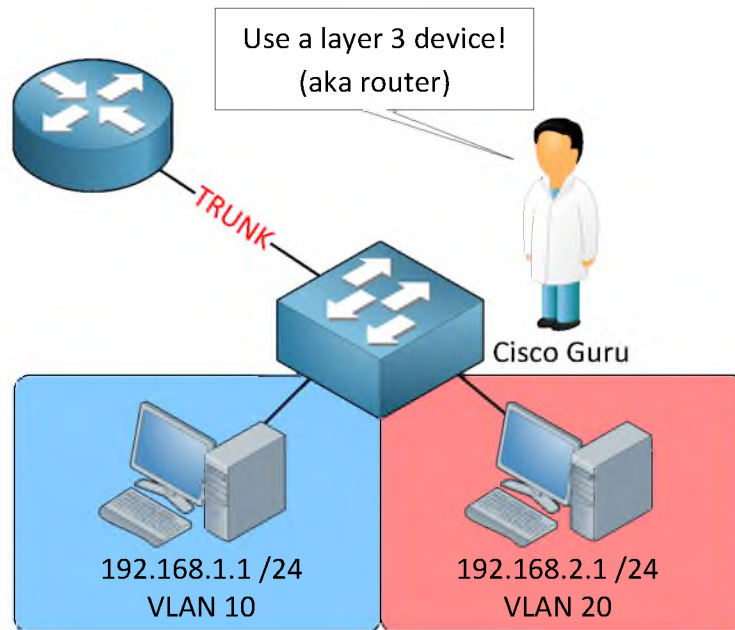
This is how we change the native VLAN.

```
SwitchB#show interfaces fa0/14 trunk
```

| Port   | Mode | Encapsulation | Status   | Native vlan |
|--------|------|---------------|----------|-------------|
| Fa0/14 | on   | 802.1q        | trunking | 10          |

You can see the native VLAN is now VLAN 10.

Question for you: Will the computers in **different VLANs** be able to communicate with each other? We separated them for a good reason but are they totally isolated?



Computers in different VLANs are still able to communicate with each other but you'll need a router! In the example above I have two computers in two different VLANs.

Each VLAN has a different IP subnet. If the computer in VLAN 10 wants to communicate with the computer in VLAN 20 we'll have to **route between the two networks**. Traffic will go from computer VLAN 10 towards the router and from the router back towards the computer in VLAN 20. This is what we call a **router on a stick**.

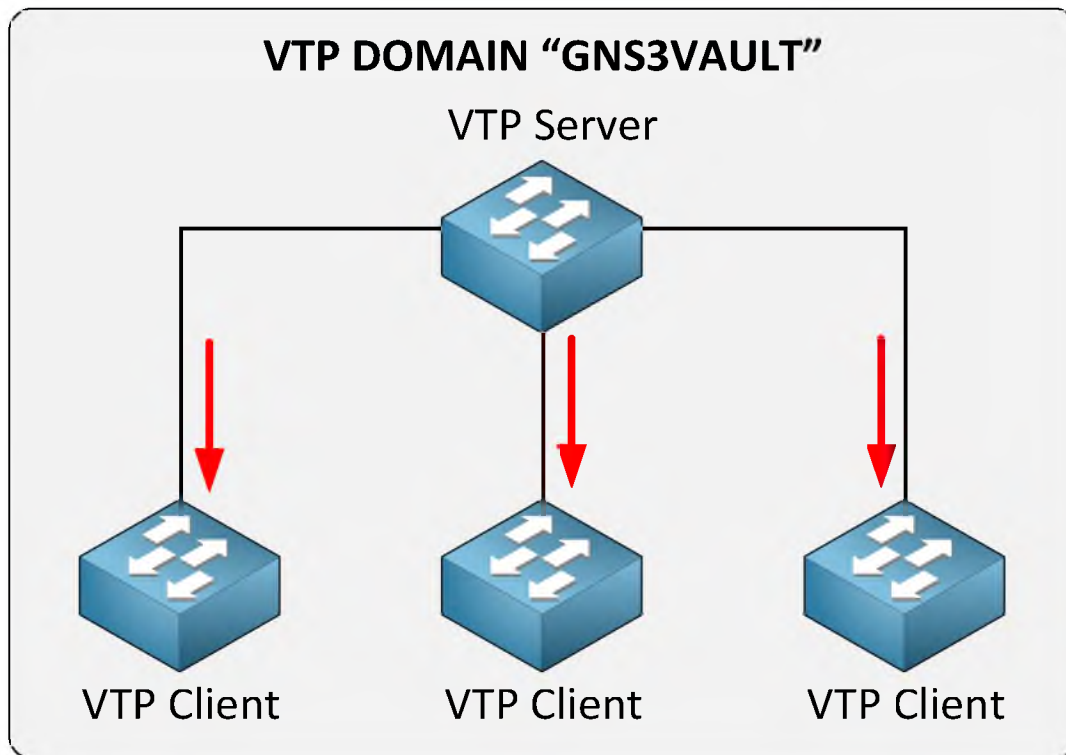
If you look at this picture you might be wondering if this makes any sense, our traffic goes from the switch to the router and from the router back to the switch. This is why nowadays you can also buy **layer 3 switches**. The routing functionality has been placed in the switch!

Later in the book when we talk about routing I will show you how to configure the router on a stick.

This is all that I have for you about VLANs and trunking. We still have to look at VTP (VLAN Trunking Protocol) which can help you to synchronize VLANs between switches.

Let's say you have a network with 20 switches and 50 VLANs. Normally you would have to configure each switch separately and create those VLANs on each and every switch. That's a time consuming task so there is something to help us called **VTP** (VLAN Trunking Protocol). VTP will let you create VLANs on one switch and all the other switches will synchronize themselves.



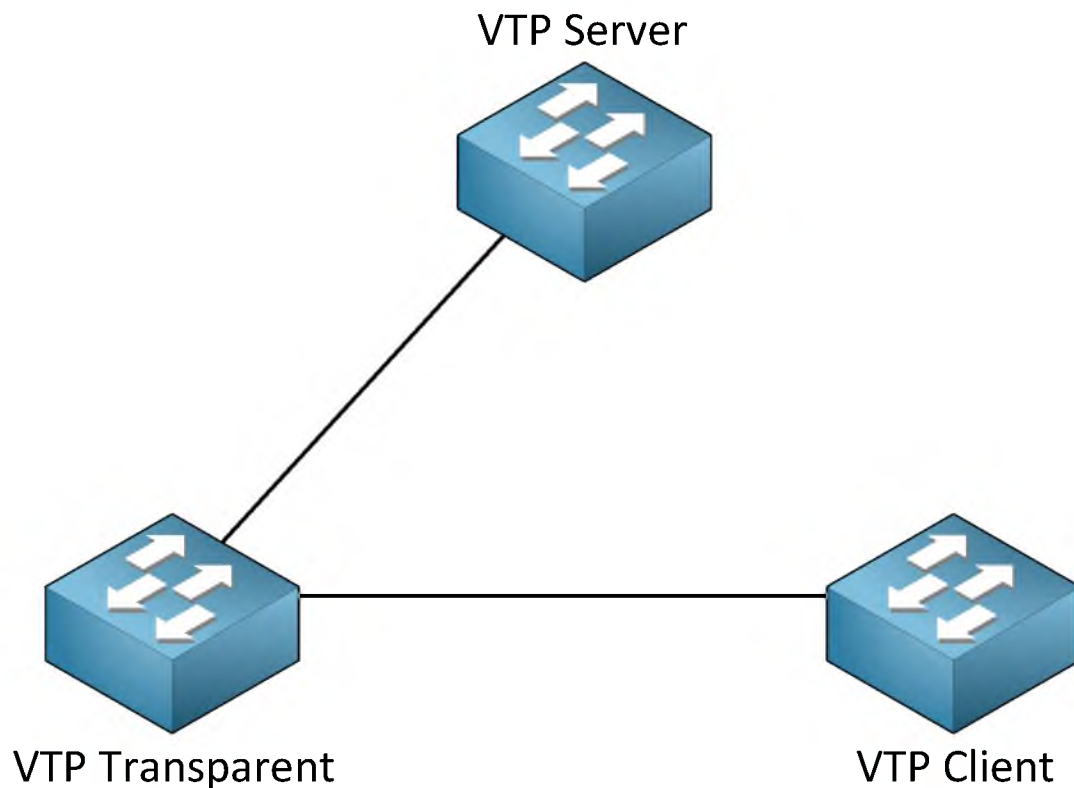


We have one VTP server which is the switch where you create / modify or delete VLANs. The other switches are VTP clients. The VTP configuration has a revision number which will increase when you make a change. Every time you make a change on the VTP server this will be synchronized to the VTP clients. Oh and by the way you can have multiple VTP servers since it also functions as a VTP client so you can make changes on multiple switches in your network. In order to make VTP work you need to setup a VTP domain name which is something you can just make up, as long as you configure it to be the same on all your switches.

This is the short version of what I just described:

1. VTP adds / modifies / deletes VLANs.
2. For every change the revision number will increase.
3. The latest advertisement will be sent to all VTP clients.
4. VTP clients will synchronize themselves with the latest information.

Besides the VTP server and VTP client there's also a VTP transparent which is a bit different, let me show you an example:



Our VTP Transparent will forward advertisements but will **not synchronize** itself. You can create VLANs locally though which is impossible on the VTP client. Let's say you create VLAN 20 on our VTP server, this is what will happen:

1. You create VLAN 20 on the VTP server.
2. The revision number will increase.
3. The VTP server will forward the latest advertisement which will reach the VTP transparent switch.
4. The VTP transparent will not synchronize itself but will forward the advertisement to the VTP client.
5. The VTP client will synchronize itself with the latest information

Here's an overview of the 3 VTP modes:

|                                   | VTP Server | VTP Client | VTP Transparent |
|-----------------------------------|------------|------------|-----------------|
| <b>Create/Modify/Delete VLANs</b> | Yes        | No         | Only local      |
| <b>Synchronizes itself</b>        | Yes        | Yes        | No              |
| <b>Forwards advertisements</b>    | Yes        | Yes        | Yes             |

Should you use VTP? It might sound useful but VTP has a big security risk...the problem with VTP is that a VTP server is also a VTP Client and any VTP client will synchronize itself with the highest revision number.

The following situation can happen with VTP:

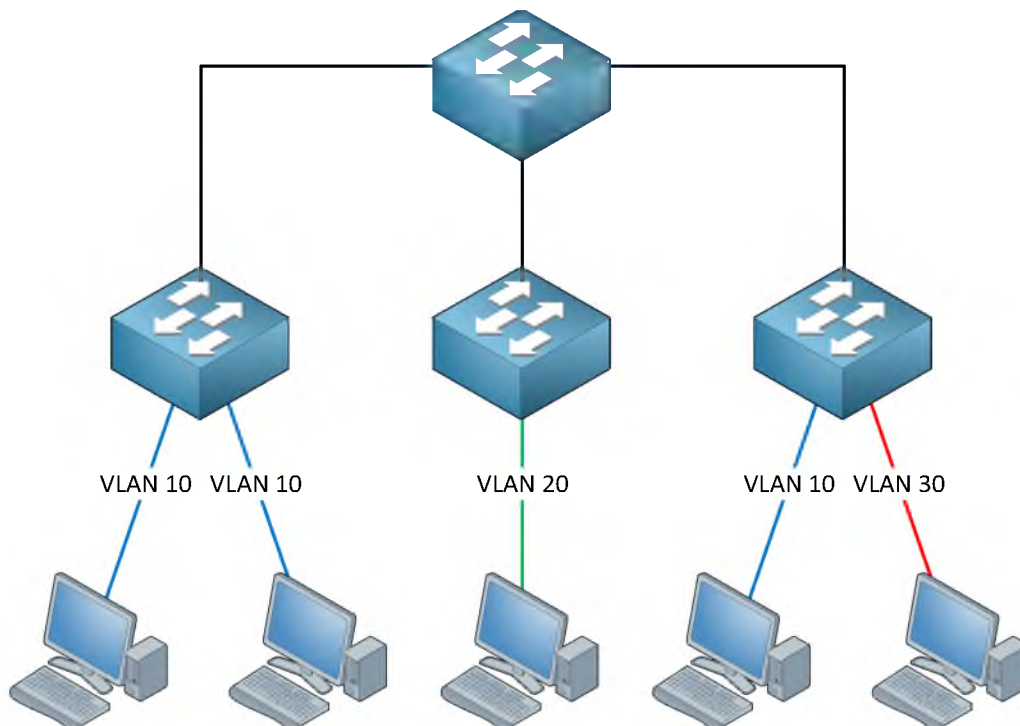
You have a network with a single VTP server and a couple of VTP client switches, everything is working fine but one day you want to test some stuff and decide to take one of the VTP clients out of the network and put it in a lab environment.

1. You take the VTP client switch out of the network.
2. You configure it so it's no longer a VTP Client but a VTP server.
3. You play around with VTP, create some VLANs, modify some.
4. Every time you make a change the revision number increases.
5. You are done playing...you delete all VLANs.
6. You configure the switch from VTP Server to VTP Client.
7. You connect your switch to your production network.

What do you think the result will be? The revision number of VTP on the switch we played with is higher than the revision number on the switches of our production network. The VTP client will advertise its information to the other switches, they synchronize to the latest information and POOF all your VLANs are gone! A VTP client can **overwrite** a VTP server if the revision number is higher because a VTP server is also a VTP client.

Yes I know this sounds silly but this is the way it works...very dangerous since you'll lose all your VLAN information. Your interfaces won't go back to VLAN 1 by default but will float around in no man's land...

One more thing about VTP, let me give you another picture:

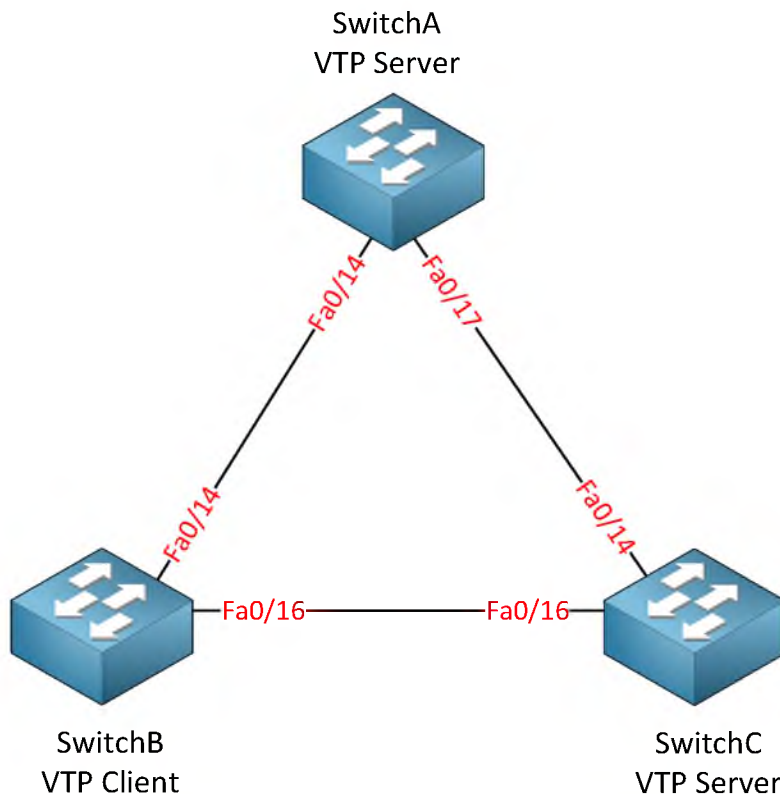


You see we have computers in VLAN 10, 20 and 30. The links between the switches are trunks using the 802.1Q protocol and carrying all VLAN traffic. One of our computers in VLAN 10 sends a broadcast frame, where do you think this broadcast frame will go?

Broadcast frames have to be flooded by our switches and since our trunks are carrying all VLANs, this broadcast will go everywhere.

However if you look at the switch in the middle do you see any computer in VLAN 10? Nope there's only VLAN 20 there which means this broadcast is wasted bandwidth. By enabling **VTP pruning** we'll make sure there is no unnecessary VLAN traffic on trunks when there's nobody in a particular VLAN. Depending on your switch model VTP pruning is either turned on or off by default.

Let's take a look at the configuration of VTP. I will be using three switches for this task. I erased the VLAN database and the startup-configuration on all switches.



```

SwitchA#show vtp status
VTP Version                : running VTP1 (VTP2 capable)
Configuration Revision      : 0
Maximum VLANs supported locally : 1005
Number of existing VLANs    : 5
VTP Operating Mode          : Server
VTP Domain Name             :
VTP Pruning Mode            : Disabled
VTP V2 Mode                 : Disabled
VTP Traps Generation        : Disabled
MD5 digest                  : 0x57 0xCD 0x40 0x65 0x63 0x59 0x47 0xBD
Configuration last modified by 0.0.0.0 at 0-0-00 00:00:00
Local updater ID is 0.0.0.0 (no valid interface found)
  
```

```
SwitchB#show vtp status
VTP Version                : running VTP1 (VTP2 capable)
Configuration Revision      : 0
Maximum VLANs supported locally : 1005
Number of existing VLANs    : 5
VTP Operating Mode          : Server
VTP Domain Name             :
VTP Pruning Mode            : Disabled
VTP V2 Mode                 : Disabled
VTP Traps Generation        : Disabled
MD5 digest                  : 0x57 0xCD 0x40 0x65 0x63 0x59 0x47 0xBD
Configuration last modified by 0.0.0.0 at 0-0-00 00:00:00
Local updater ID is 0.0.0.0 (no valid interface found)
```

```
SwitchC#show vtp status
VTP Version                : 2
Configuration Revision      : 0
Maximum VLANs supported locally : 1005
Number of existing VLANs    : 5
VTP Operating Mode          : Server
VTP Domain Name             :
VTP Pruning Mode            : Disabled
VTP V2 Mode                 : Disabled
VTP Traps Generation        : Disabled
MD5 digest                  : 0x57 0xCD 0x40 0x65 0x63 0x59 0x47 0xBD
Configuration last modified by 0.0.0.0 at 0-0-00 00:00:00
Local updater ID is 0.0.0.0 (no valid interface found)
```

Depending on the switch model you will see a similar output if you use the **show vtp status** command. There's a couple of interesting things to see here:

- Configuration revision 0: Each time we add or remove VLANs this number will change. It's 0 at the moment since I haven't created or removed any VLANs.
- VTP Operating mode: the default is VTP server.
- VTP Pruning: this will help to prevent unnecessary traffic on your trunk links, more in this later.
- VTP V2 Mode: The switch is capable of running VTP version 2 but it's currently running VTP version 1.

```
SwitchA(config)#vlan 10
SwitchA(config-vlan)#name Printers
```

Let's create a VLAN on SwitchA and we'll see if anything changes...

```
SwitchA#show vlan
```

| VLAN | Name     | Status | Ports  |
|------|----------|--------|--|
| 1    | default  | active | Fa0/1, Fa0/2, Fa0/3, Fa0/4<br>Fa0/5, Fa0/6, Fa0/7, Fa0/8<br>Fa0/9, Fa0/10, Fa0/11, Fa0/12<br>Fa0/13, Fa0/14, Fa0/22<br>Fa0/23, Fa0/24, Gi0/2 |
| 10   | Printers | active |  |

My new VLAN shows up in the VLAN database, so far so good...

```
SwitchA#show vtp status
VTP Version           : running VTP1 (VTP2 capable)
Configuration Revision : 1
```

You can see that the configuration revision has increased by one.

```
SwitchB#show vtp status
VTP Version           : running VTP1 (VTP2 capable)
Configuration Revision : 0
```

```
SwitchC#show vtp status
VTP Version           : 2
Configuration Revision : 0
```

Unfortunately nothing has changed on SwitchB and SwitchC. This is because we need to configure a **VTP domain-name** before it starts working.

```
SwitchB#debug sw-vlan vtp events
vtp events debugging is on
```

```
SwitchC#debug sw-vlan vtp events
vtp events debugging is on
```

Before I change the domain-name I'm going to enable a debug using the **debug sw-vlan vtp events** command. This way we can see in real-time what is going on.

```
SwitchA(config)#vtp domain GNS3VAULT
Changing VTP domain name from NULL to GNS3VAULT
```

```
SwitchB#
VTP LOG RUNTIME: Summary packet received in NULL domain state
VTP LOG RUNTIME: Summary packet received, domain = GNS3VAULT, rev = 1,
followers = 1, length 77, trunk Fa0/16
VTP LOG RUNTIME: Transitioning from NULL to GNS3VAULT domain
VTP LOG RUNTIME: Summary packet rev 1 greater than domain GNS3VAULT rev 0
```

You will see the following debug information on SwitchB and SwitchC; there are two interesting things we can see here:

- The switch receives a VTP packet from domain "GNS3VAULT" and decides to change its own domain-name from "NULL" (nothing) to "GNS3VAULT". It will only change the domain-name if it doesn't have a domain-name.
- The switch sees that the VTP packet has a higher revision number (1) than what it currently has (0) and as a result it will synchronize itself.

```
SwitchB#no debug all
All possible debugging has been turned off
```

```
SwitchC#no debug all
All possible debugging has been turned off
```

Make sure to disable the debug output before you get flooded with information.

```
SwitchB#show vtp status
VTP Version           : running VTP1 (VTP2 capable)
Configuration Revision : 1
```

```
SwitchC#show vtp status
VTP Version           : 2
Configuration Revision : 1
```

The revision number on SwitchB and SwitchC is now "1".

```
SwitchB#show vlan

VLAN Name                Status    Ports
----
1    default                active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                           Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                           Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                           Fa0/13, Fa0/14, Fa0/15,
                                           Fa0/23, Fa0/24, Gi0/1, Gi0/2
10   Printers                active
```

```
SwitchC#show vlan

VLAN Name                Status    Ports
----
1    default                active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                           Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                           Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                           Fa0/20, Fa0/22, Fa0/23,
                                           Gi0/1, Gi0/2
10   Printers                active
```

The show vlan command tells us that SwitchB and SwitchC have learned VLAN 10 through VTP.

Since all switches are in VTP Server mode I can create VLANs on any switch and they should all synchronize:

```
SwitchB(config)#vlan 20
SwitchB(config-vlan)#name Servers
```

```
SwitchC(config)#vlan 30
SwitchC(config-vlan)#name Management
```

Let's create VLAN 20 on SwitchB and VLAN 30 on SwitchC.

```
SwitchA#show vlan
```

| VLAN | Name       | Status | Ports |
|------|------------|--------|-------|
| 10   | Printers   | active |       |
| 20   | Servers    | active |       |
| 30   | Management | active |       |

```
SwitchB#show vlan
```

| VLAN | Name       | Status | Ports |
|------|------------|--------|-------|
| 10   | Printers   | active |       |
| 20   | Servers    | active |       |
| 30   | Management | active |       |

```
SwitchC#show vlan
```

| VLAN | Name       | Status | Ports |
|------|------------|--------|-------|
| 10   | Printers   | active |       |
| 20   | Servers    | active |       |
| 30   | Management | active |       |

As you can see all switches know about the VLANs. What about the revision number? Did it change?

```
SwitchA#show vtp status
```

```
VTP Version : running VTP1 (VTP2 capable)
Configuration Revision : 3
```

```
SwitchB#show vtp status
```

```
VTP Version : running VTP1 (VTP2 capable)
Configuration Revision : 3
```

```
SwitchC#show vtp status
```

```
VTP Version : 2
Configuration Revision : 3
```



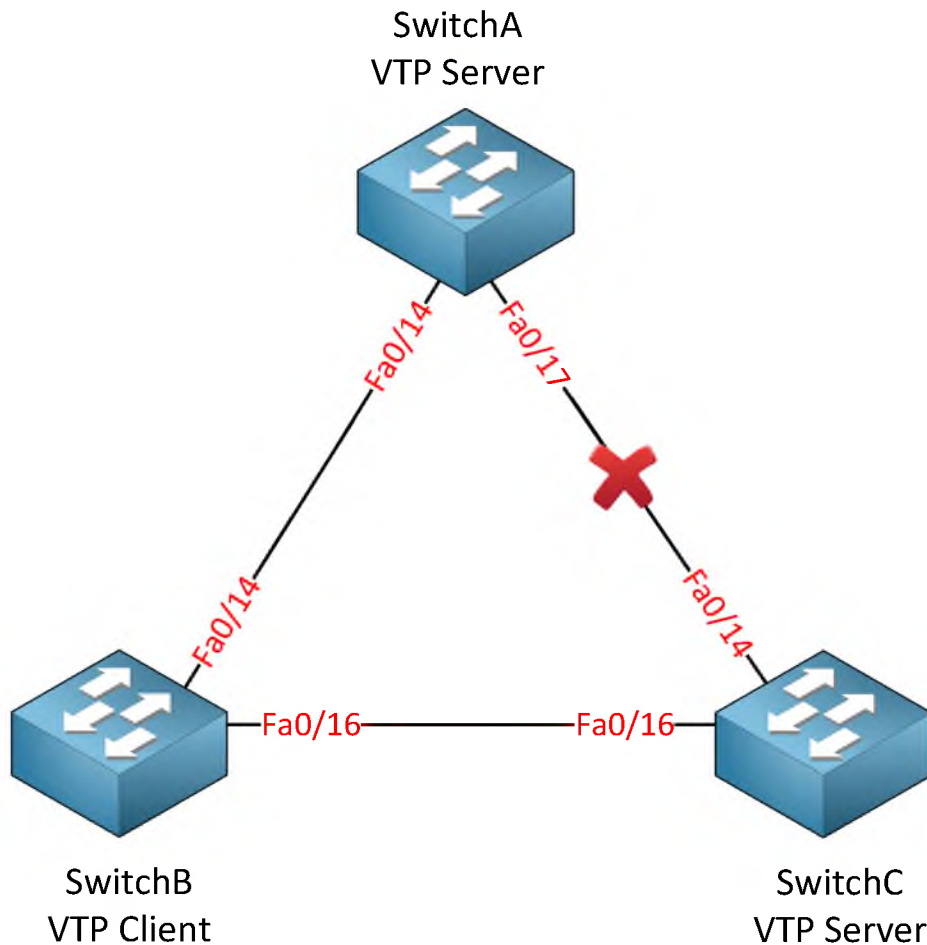
Each time I create another VLAN the revision number increases by one. Let's change the VTP mode on SwitchB to see what it does.

```
SwitchB(config)#vtp mode client  
Setting device to VTP CLIENT mode.
```

```
SwitchB#show vtp status  
VTP Version           : running VTP1 (VTP2 capable)  
Configuration Revision : 3  
Maximum VLANs supported locally : 1005  
Number of existing VLANs : 7  
VTP Operating Mode     : Client
```

It's now running in VTP Client mode.

Right now SwitchA and SwitchC are in VTP Server mode. SwitchB is running VTP Client mode. I have disconnected the link between SwitchA and SwitchC so there is no direct connection between them.



I'll create another VLAN on SwitchA so we can see if SwitchB and SwitchC will learn it.

```
SwitchA(config)#vlan 40
```

```
SwitchA(config-vlan) #name Engineering
```

I'll call the new VLAN "Engineering".

```
SwitchB#show vlan
```

| VLAN | Name        | Status | Ports |
|------|-------------|--------|-------|
| 10   | Printers    | active |       |
| 20   | Servers     | active |       |
| 30   | Management  | active |       |
| 40   | Engineering | active |       |

SwitchB learns about VLAN 40 through SwitchA.

```
SwitchC#show vlan
```

| VLAN | Name        | Status | Ports |
|------|-------------|--------|-------|
| 10   | Printers    | active |       |
| 20   | Servers     | active |       |
| 30   | Management  | active |       |
| 40   | Engineering | active |       |

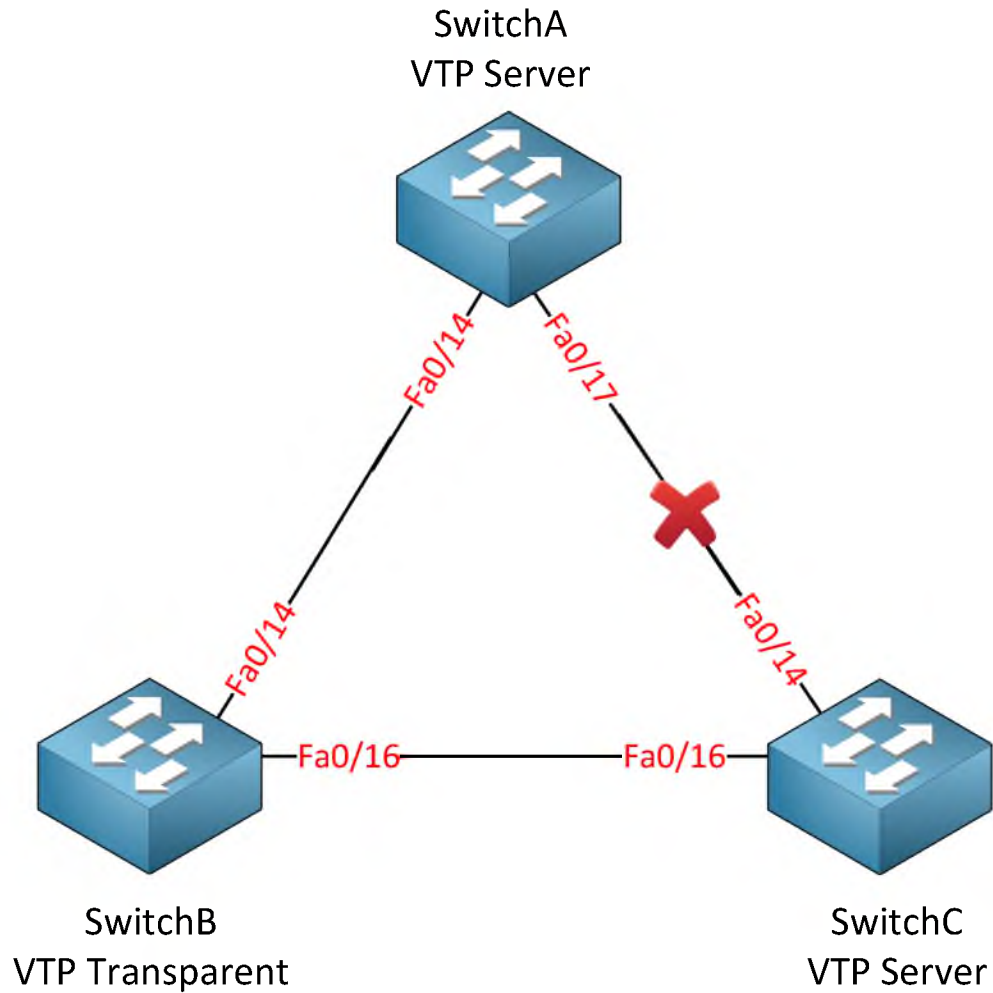
SwitchC learns about VLAN 40 through SwitchB. SwitchB as a VTP client will synchronize itself but it will also forward VTP advertisements.

```
SwitchB(config) #vlan 50
```

```
%VTP VLAN configuration not allowed when device is in CLIENT mode.
```

A switch running in VTP Client mode is unable to create VLANs so that's why I get this error if I try to create one.

What about the VTP Transparent mode? That's the last one we have to try...



I'll change SwitchB to VTP Transparent mode and the link between SwitchA and SwitchC is still disconnected.

```
SwitchB(config)#vtp mode transparent
Setting device to VTP TRANSPARENT mode.
```

This is how we change SwitchB to VTP Transparent mode.

```
SwitchA(config)#vlan 50
SwitchA(config-vlan)#name Research
```

Let's create VLAN 50 for this experiment on SwitchA.

```
SwitchA#show vlan
```

| VLAN | Name        | Status | Ports |
|------|-------------|--------|-------|
| 10   | Printers    | active |       |
| 20   | Servers     | active |       |
| 30   | Management  | active |       |
| 40   | Engineering | active |       |
| 50   | Research    | active |       |

It shows up on SwitchA as expected.

```
SwitchB#show vlan
```

| VLAN | Name        | Status | Ports |
|------|-------------|--------|-------|
| 10   | Printers    | active |       |
| 20   | Servers     | active |       |
| 30   | Management  | active |       |
| 40   | Engineering | active |       |

It doesn't show up on SwitchB because it's in VTP transparent mode and doesn't synchronize itself.

```
SwitchC#show vlan
```

| VLAN | Name        | Status | Ports |
|------|-------------|--------|-------|
| 10   | Printers    | active |       |
| 20   | Servers     | active |       |
| 30   | Management  | active |       |
| 40   | Engineering | active |       |
| 50   | Research    | active |       |

It does show up on SwitchC! A switch in VTP Transparent mode will **not synchronize itself** but it will **forward VTP advertisements** to other switches so they can synchronize themselves.

What will happen if I create a VLAN on SwitchB? Let's find out!

```
SwitchB(config)#vlan 60
SwitchB(config-vlan)#name Cameras
```

```
SwitchB#show vlan
```

| VLAN | Name        | Status | Ports |
|------|-------------|--------|-------|
| 10   | Printers    | active |       |
| 20   | Servers     | active |       |
| 30   | Management  | active |       |
| 40   | Engineering | active |       |
| 50   | Research    | active |       |
| 60   | Cameras     | active |       |

We can create this new VLAN on SwitchB without any trouble. It's in VTP Transparent mode so we can do this.

```
SwitchA#show vlan
```

| VLAN | Name        | Status | Ports |
|------|-------------|--------|-------|
| 10   | Printers    | active |       |
| 20   | Servers     | active |       |
| 30   | Management  | active |       |
| 40   | Engineering | active |       |
| 50   | Research    | active |       |

```
SwitchC#show vlan
```

| VLAN | Name        | Status | Ports |
|------|-------------|--------|-------|
| 10   | Printers    | active |       |
| 20   | Servers     | active |       |
| 30   | Management  | active |       |
| 40   | Engineering | active |       |
| 50   | Research    | active |       |

VLAN 60 doesn't show up on SwitchA and SwitchC because SwitchB is in VTP Transparent mode. SwitchB will not advertise its VLANs because they are only **known locally**.

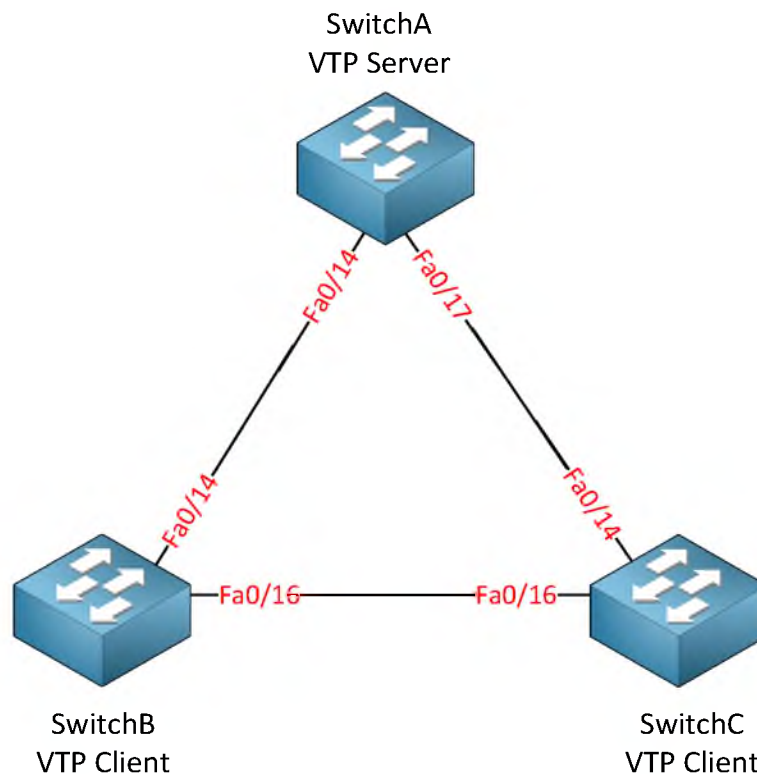
Is there anything else you need to know about VTP Transparent mode?

```
SwitchB#show running-config
Building configuration...

vlan 10
  name Printers
!
vlan 20
  name Servers
!
vlan 30
  name Management
!
vlan 40
  name Engineering
!
vlan 60
  name Cameras
```

There's a difference between VTP Transparent mode VS Server/Client mode. If you look at the running-config you will see that VTP Transparent stores all VLAN information in the running-config. VTP Server and Client mode store their information in the VLAN database (vlan.dat on your flash memory).

If you understand the difference between VTP Server, Client and Transparent mode...good! There's one more thing I want to show to you about VTP.



I'm going to demonstrate the danger of VTP as I explained before. A VTP client can overwrite a VTP server if the revision number of the VTP client is higher. I'm using the same topology but this time SwitchA is the VTP Server and SwitchB and SwitchC are VTP Clients.

```
SwitchA(config)#vtp mode server
Device mode already VTP SERVER.
SwitchA(config)#vtp domain GNS3VAULT
Changing VTP domain name from NULL to GNS3VAULT
```

```
SwitchB(config)#vtp mode client
Setting device to VTP CLIENT mode.
```

```
SwitchC(config)#vtp mode client
Setting device to VTP CLIENT mode.
```

First I change the domain-name and configure the correct VTP modes.

```
SwitchA(config)#vlan 10
SwitchA(config-vlan)#name Printers
SwitchA(config)#vlan 20
SwitchA(config-vlan)#name Servers
SwitchA(config)#vlan 30
SwitchA(config-vlan)#name Management
```

Next step is to create a couple of VLANS.

```
SwitchA(config)#interface fa0/1
SwitchA(config-if)#switchport mode access
SwitchA(config-if)#switchport access vlan 10
```

I will configure one (random) interface so it's in VLAN 10.

```
SwitchA#show vtp status
VTP Version                : running VTP1 (VTP2 capable)
Configuration Revision      : 4
```

```
SwitchB#show vtp status
VTP Version                : running VTP1 (VTP2 capable)
Configuration Revision      : 4
```

```
SwitchC#show vtp status
VTP Version                : 2
Configuration Revision      : 4
```

All switches currently have the same revision number.

SwitchA#show vlan

| VLAN | Name       | Status | Ports                      |
|------|------------|--------|----------------------------|
| --   | --         | --     | --                         |
| 1    | default    | active | Fa0/2, Fa0/3, Fa0/4, Fa0/5 |
| 10   | Printers   | active | <b>Fa0/1</b>               |
| 20   | Servers    | active |                            |
| 30   | Management | active |                            |

SwitchB#show vlan

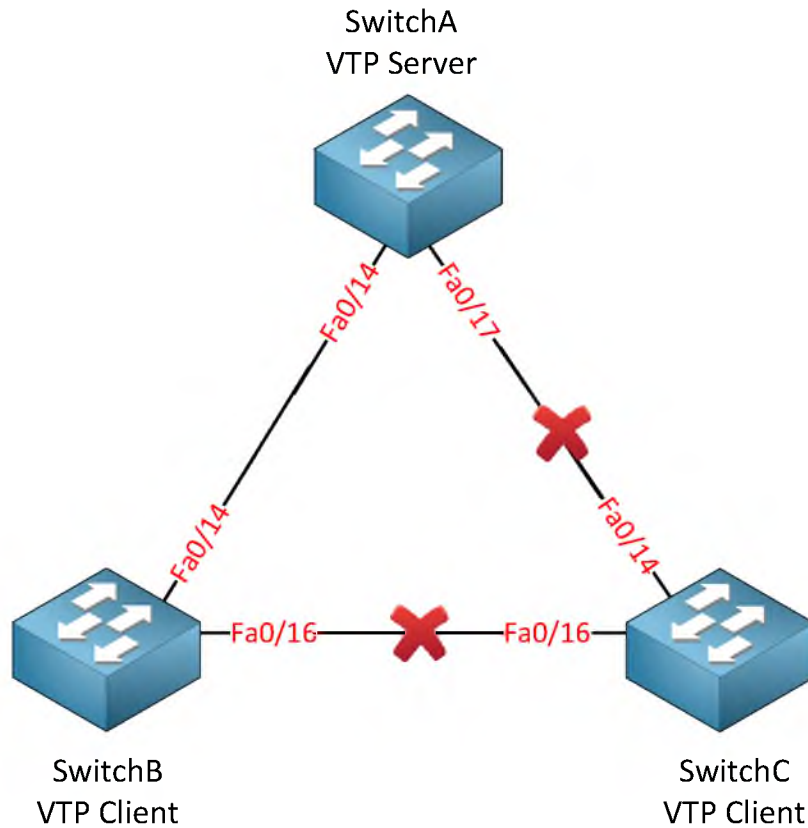
| VLAN | Name       | Status | Ports                      |
|------|------------|--------|----------------------------|
| --   | --         | --     | --                         |
| 1    | default    | active | Fa0/2, Fa0/3, Fa0/4, Fa0/5 |
| 10   | Printers   | active |                            |
| 20   | Servers    | active |                            |
| 30   | Management | active |                            |

SwitchC#show vlan

| VLAN | Name       | Status | Ports                      |
|------|------------|--------|----------------------------|
| --   | --         | --     | --                         |
| 1    | default    | active | Fa0/2, Fa0/3, Fa0/4, Fa0/5 |
| 10   | Printers   | active |                            |
| 20   | Servers    | active |                            |
| 30   | Management | active |                            |

All switches are up-to-date with the latest VLAN information. Note that Fa0/1 on SwitchA is in VLAN 10 at this moment.





Now I'm going to shut down the interfaces on SwitchC connecting SwitchA and SwitchB. This could happen if you want to remove a switch from your production network and temporarily use it in a lab.

```
SwitchC(config)#interface fa0/14
SwitchC(config-if)#shutdown
SwitchC(config)#interface fa0/16
SwitchC(config-if)#shutdown
```

I will change SwitchC from VTP Client mode to VTP Server mode:

```
SwitchC(config)#vtp mode server
Setting device to VTP SERVER mode
```

Easy enough! Let's add some VLANs:

```
SwitchC(config)#vlan 70
SwitchC(config-vlan)#name Lab
SwitchC(config)#vlan 80
SwitchC(config-vlan)#name Experiment
```

```
SwitchC#show vtp status
VTP Version                : 2
Configuration Revision      : 6
```

After adding the VLANs you can see that the VTP revision number has increased.

```
SwitchC(config)#no vlan 10
SwitchC(config)#no vlan 20
SwitchC(config)#no vlan 30
SwitchC(config)#no vlan 70
SwitchC(config)#no vlan 80
```

```
SwitchC(config)#vtp mode client
Setting device to VTP CLIENT mode.
```

After playing with my lab I'm going to erase the VLANs and change the switch back to VTP Client mode so we can return it to the production network.

```
SwitchC#show vtp status
VTP Version                : 2
Configuration Revision      : 11
```

Note that after deleting the VLANs the VTP revision number increased even more.

```
SwitchC(config)#interface fa0/14
SwitchC(config-if)#no shutdown
SwitchC(config)#interface fa0/16
SwitchC(config-if)#no shutdown
```

Let's do a "no shutdown" on the interfaces and return SwitchC to the production network.

```
SwitchA#show vtp status
VTP Version                : running VTP1 (VTP2 capable)
Configuration Revision      : 11
```

```
SwitchB#show vtp status
VTP Version           : running VTP1 (VTP2 capable)
Configuration Revision : 11
```

Ugh...this doesn't look good. SwitchA and SwitchB now have the same revision number as SwitchC. This is the moment where you start to get nervous before you type in the next command...

```
SwitchA#show vlan
```

| VLAN | Name    | Status | Ports                      |
|------|---------|--------|----------------------------|
| 1    | default | active | Fa0/2, Fa0/3, Fa0/4, Fa0/5 |

```
SwitchB#show vlan
```

| VLAN | Name    | Status | Ports                      |
|------|---------|--------|----------------------------|
| 1    | default | active | Fa0/1, Fa0/2, Fa0/3, Fa0/4 |

OUCH! All VLAN information is lost since SwitchA and SwitchB are synchronized to the latest information from SwitchC.



This is the moment where your relaxing Monday morning turns into a horrible day...if you are lucky the support ticket system doesn't work anymore...if you are using VoIP than there's a chance your phones don't work anymore and you just have to wait till a mob of angry users will ram your door because they blame you for not being able to reach Facebook anymore...

Ok maybe I'm exaggerating a bit but you get the idea. If you have a big flat network with lots of switches and VLANs than this would be a disaster. I would advise to use VTP Transparent mode on all your switches and create VLANs locally!

If you do want to use VTP Server / Client mode you need to make sure you **reset the revision number**:

- Changing the **domain-name** will reset the revision number.
- Deleting the vlan.dat file on your flash memory will reset the revision number.



*What happens to interfaces when you delete a VLAN? Take a close look at the show vlan command on SwitchA on the previous page. When you delete a VLAN all interfaces are in 'no-man's land'. They don't return to VLAN 1...you'll have to re-assign them yourself!*

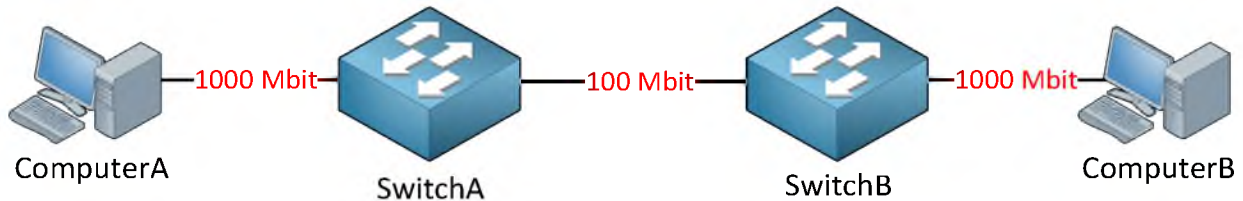
This is the end of the VLAN, Trunks and VTP chapter. If you want to get some practice I recommend you to take a look at the following labs:

<http://gns3vault.com/Switching/vlans-and-trunks.html>

<http://gns3vault.com/Switching/vtp-vlan-trunking-protocol.html>

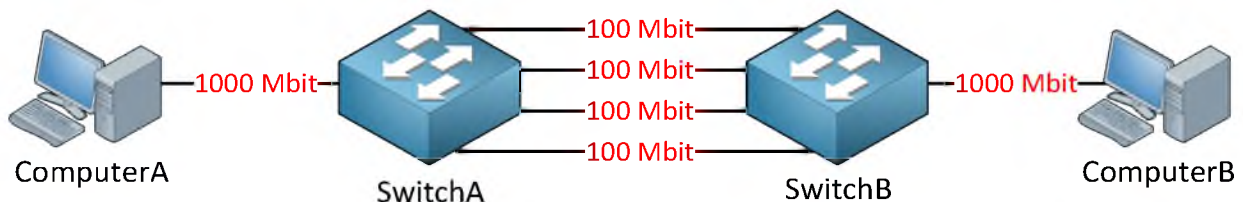
## 10. Etherchannel (Link Aggregation)

In this chapter we'll take a look at **etherchannel** which is also known as **link aggregation** or **port-channel**. Etherchannel is a technology that lets you bundle multiple physical links into a single logical link. We'll take a look at how it works and what the advantages of etherchannel are.



Take a look at the picture above. I have two switches and two computers connected to the switches. The computers are connected with 1000 Mbit interfaces while the link between the switches is only 100 Mbit. If one of the computers would send traffic that exceeds 100 Mbit of bandwidth we'll have congestion and traffic will be dropped. There are two solutions to this problem:

- Replace the link in between the switches with something faster, 1000Mbit or maybe even 10 gigabit if you feel like.
- Add multiple links and bundle them into an etherchannel.



In the picture above I have added a couple of extra links. The problem is that 3 out of 4 of these links will be blocked because of loop prevention. Spanning-tree is a protocol that prevents loops on switched networks (you'll learn about it in the next chapter!).



The cool thing about etherchannel is that it will bundle all physical links into a logical link with the combined bandwidth. By combining 4x 100 Mbit I now have a 400 Mbit link.

Etherchannel will do load balancing among the different links that we have and it takes care of redundancy. Once one of the links fails it will keep working and use the links that we have left. There's a maximum to the number of links you can use: **8 physical interfaces**.

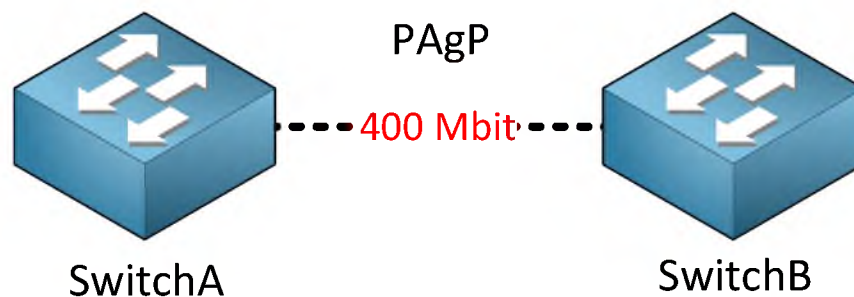
If you want to configure an Etherchannel there are two protocols you can choose from:

- **PAgP (Cisco proprietary)**
- **LACP (IEEE standard)**

These protocols can dynamically configure an etherchannel. It's also possible to configure a static etherchannel without these protocols doing the negotiation of the link for you. If you are going to create an etherchannel you need to make sure that all ports have the same configuration:

- Duplex has to be the same.
- Speed has to be the same.
- Same native AND allowed VLANs.
- Same switchport mode (access or trunk).

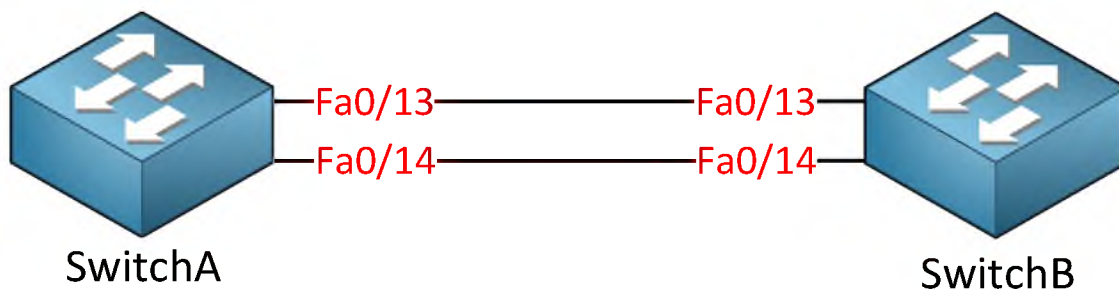
PAgp and LACP will check if the configurations of the interfaces that you use are the same.



If you want to configure PAgP you have a number of options you can choose from, an interface can be configured as:

- **On** (interface becomes member of the etherchannel but does not negotiate).
- **Desirable** (interface will actively ask the other side to become an etherchannel).
- **Auto** (interface will wait passively for the other side to ask to become an etherchannel).
- **Off** (no etherchannel configured on the interface).

Let me show you the configuration of PAgP and how the different options work!



I'll use SwitchA and SwitchB to demonstrate PAgP. We'll use two interfaces to bundle into a single logical link.

```
SwitchA(config)#interface fa0/13
SwitchA(config-if)#channel-group 1 mode ?
    active      Enable LACP unconditionally
    auto        Enable PAgP only if a PAgP device is detected
    desirable   Enable PAgP unconditionally
    on          Enable Etherchannel only
    passive     Enable LACP only if a LACP device is detected
```

First we go to the interface level where we can create a **channel-group**. I'm going to use channel-group number 1. Above you can see the different options that we have for PAgP and LACP.

```
SwitchA(config)#interface fa0/13
SwitchA(config-if)#channel-group 1 mode desirable
Creating a port-channel interface Port-channel 1
SwitchA(config)#interface fa0/14
SwitchA(config-if)#channel-group 1 mode desirable
```

I configure SwitchA for PAgP desirable mode. It will actively ask SwitchB to become an Etherchannel this way.

```
SwitchB(config)#interface fa0/13
SwitchB(config-if)#channel-group 1 mode auto
SwitchB(config)#interface fa0/14
SwitchB(config-if)#channel-group 1 mode auto
```

Here's the configuration of SwitchB. I used the PAgP auto mode so it will respond to requests to become an etherchannel.

```
SwitchA %LINK-3-UPDOWN: Interface Port-channel1, changed state to up
```

```
SwitchB %LINK-3-UPDOWN: Interface Port-channel1, changed state to up
```

You'll see a message on your switches like mine above. The switch will create a port-channel interface.

```
SwitchA(config)#interface port-channel 1
SwitchA(config-if)#switchport trunk encapsulation dot1q
SwitchA(config-if)#switchport mode trunk
```

```
SwitchB(config)#interface port-channel 1
SwitchB(config-if)#switchport trunk encapsulation dot1q
SwitchB(config-if)#switchport mode trunk
```

The port-channel interface can be configured. I've set mine to use 802.1Q encapsulation and to become a trunk.

```
SwitchA#show etherchannel 1 port-channel
      Port-channels in the group:
      -----

Port-channel: Po1
-----

Age of the Port-channel   = 0d:00h:10m:16s
Logical slot/port        = 2/1           Number of ports = 2
GC                        = 0x00010001    HotStandBy port = null
Port state                = Port-channel Ag-Inuse
Protocol                  = PAgP
Port security             = Disabled

Ports in the Port-channel:
```

| Index | Load | Port   | EC state     | No of bits |
|-------|------|--------|--------------|------------|
| 0     | 00   | Fa0/13 | Desirable-S1 | 0          |
| 0     | 00   | Fa0/14 | Desirable-S1 | 0          |

```

Time since last port bundled:    0d:00h:00m:07s    Fa0/14
Time since last port Un-bundled: 0d:00h:04m:08s    Fa0/13

```

Here's one way to verify your configuration. Use the `show etherchannel port-channel` command to check if the port-channel is active or not. You can also see that we are using PAgP. Interface fa0/13 and fa0/14 are both in use for this etherchannel.

```
SwitchA#show etherchannel summary
Flags:  D - down          P - bundled in port-channel
        I - stand-alone  s - suspended
        H - Hot-standby (LACP only)
        R - Layer3       S - Layer2
        U - in use       f - failed to allocate aggregator

        M - not in use, minimum links not met
        u - unsuitable for bundling
        w - waiting to be aggregated
        d - default port

Number of channel-groups in use: 1
Number of aggregators:          1

Group  Port-channel  Protocol    Ports
-----+-----+-----+-----
1      Po1 (SU)      PAgP        Fa0/13 (P)  Fa0/14 (P)
```

If you have many etherchannels you can also use the `show etherchannel summary` command. It will give you a quick overview of all the etherchannels and the interfaces that are in use.



```
SwitchA#show interfaces fa0/14 etherchannel
Port state      = Up Mstr In-Bndl
Channel group = 1          Mode = Desirable-Sl      Gcchange = 0
Port-channel   = Po1       GC   = 0x00010001      Pseudo port-channel = Po1
Port index     = 0         Load = 0x00          Protocol = PAgP

Flags:  S - Device is sending Slow hello.  C - Device is in Consistent state.
        A - Device is in Auto mode.        P - Device learns on physical
port.
        d - PAgP is down.
Timers: H - Hello timer is running.        Q - Quit timer is running.
        S - Switching timer is running.    I - Interface timer is running.

Local information:

Port          Flags State  Timers Interval Count  Priority  Method  Ifindex
Fa0/14        SC      U6/S7   H    30s     1       128      Any     5001

Partner's information:

Port          Partner          Partner          Partner          Partner Group
Name          Device ID          Port          Age  Flags  Cap.
Fa0/14        SwitchB            0019.569d.5700 Fa0/14        19s  SAC   10001

Age of the port in the current state: 0d:00h:02m:37s
```

The third method to verify your etherchannel is to use the **show interfaces etherchannel** command. In my example I am looking at the information of my fa0/14 interface. Besides information of our local switch you can also see the interface of our neighbor switch (SwitchB in my example).

The last thing I want to share with you about PAgP are the different modes you can choose from:

- On
- Desirable
- Auto
- Off

I have configured SwitchA to use desirable and SwitchB to use auto mode. Not all the different combinations work:

|           | On  | Desirable | Auto | Off |
|-----------|-----|-----------|------|-----|
| On        | Yes | No        | No   | No  |
| Desirable | No  | Yes       | Yes  | No  |
| Auto      | No  | Yes       | No   | No  |
| Off       | No  | No        | No   | No  |

Here's an overview with all the different options. Keep in mind that configuring your etherchannel as "on" doesn't use any negotiation so it will fail if the other side is configured for auto or desirable.

LACP is similar to PAgP. You also have different options to choose from when you configure the interface:

- **On** (interface becomes member of the etherchannel but does not negotiate).
- **Active** (interface will actively ask the other side to become an etherchannel).
- **Passive** (interface will wait passively for the other side to ask to become an etherchannel).
- **Off** (no etherchannel configured on the interface).

It's basically the same thing as PAgP but the terminology is different. Let's configure LACP to see what it does.

```
SwitchA(config)#default interface fa0/13
Interface FastEthernet0/13 set to default configuration
SwitchA(config)#default interface fa0/14
Interface FastEthernet0/14 set to default configuration
```

```
SwitchB(config)#default interface fa0/13
Interface FastEthernet0/13 set to default configuration
SwitchB(config)#default interface fa0/14
Interface FastEthernet0/14 set to default configuration
```

```
SwitchA(config)#no interface port-channel1
```

```
SwitchB(config)#no interface port-channel1
```

Don't forget to clean up PAgP before you start playing with LACP.

```
SwitchA(config-if)#interface fa0/13
SwitchA(config-if)#channel-group 1 mode active
Creating a port-channel interface Port-channel 1
SwitchA(config-if)#interface f0/14
SwitchA(config-if)#channel-group 1 mode active
```

I'll configure SwitchA to use LACP active mode.

```
SwitchB(config)#interface fa0/13
SwitchB(config-if)#channel-group 1 mode passive
Creating a port-channel interface Port-channel 1
SwitchB(config-if)#interface fa0/14
SwitchB(config-if)#channel-group 1 mode passive
```

SwitchB will use LACP passive mode.

```
SwitchA#show etherchannel 1 port-channel
Port-channels in the group:
-----

Port-channel: Po1      (Primary Aggregator)

-----

Age of the Port-channel   = 0d:00h:03m:04s
Logical slot/port        = 2/1           Number of ports = 2
HotStandBy port = null
Port state                = Port-channel Ag-Inuse
Protocol                  = LACP
Port security             = Disabled

Ports in the Port-channel:

Index  Load  Port      EC state      No of bits
-----+-----+-----+-----+-----
0      00     Fa0/13    Active        0
0      00     Fa0/14    Active        0

Time since last port bundled: 0d:00h:00m:54s Fa0/14
```

We can use the show etherchannel port-channel command again to verify our configuration again. As you can see the protocol is now LACP and interfaces fa0/13 and fa0/14 are active.

The configuration of PAgP and LACP is similar. Keep in mind that PAgP can only be used between Cisco devices while LACP is a IEEE standard, you can use it to form etherchannels with devices from other vendors.

|         | On  | Active | Passive | Off |
|---------|-----|--------|---------|-----|
| On      | Yes | No     | No      | No  |
| Active  | No  | Yes    | Yes     | No  |
| Passive | No  | Yes    | No      | No  |
| Off     | No  | No     | No      | No  |

Here's an overview with the different modes and combinations for LACP. It's similar to PAgP but now we have the active and passive mode.

Last thing I want to show you about etherchannel is load-balancing:

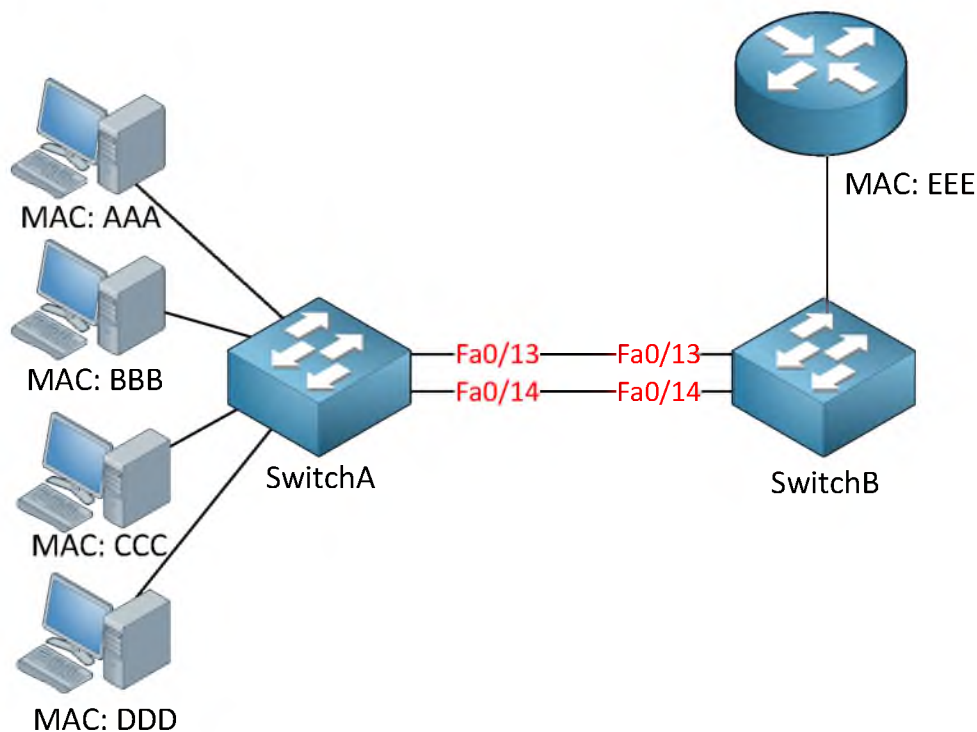
```
SwitchA#show etherchannel load-balance
EtherChannel Load-Balancing Configuration:
src-mac

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

Use the **show etherchannel load-balance** command to see what the default configuration is. As you can see our etherchannel load-balances based on the source MAC address.

```
SwitchA(config)#port-channel load-balance ?
dst-ip      Dst IP Addr
dst-mac     Dst Mac Addr
src-dst-ip  Src XOR Dst IP Addr
src-dst-mac Src XOR Dst Mac Addr
src-ip      Src IP Addr
src-mac     Src Mac Addr
```

You can use the global **port-channel load-balance** command to change this behavior. You can see you can choose between source/destination MAC/IP address or a combination of source/destination.



Why should you care about load balancing? Take a look at the picture above. We have 4 computers and one router on the right side. The default load-balancing mechanism is source MAC address. This means that **ALL** traffic from **one** MAC address will be sent down one and the same physical interface, for example:

- MAC address AAA will be sent using SwitchA's fa0/13 interface.
- MAC address BBB will be sent using SwitchA's fa0/14 interface.
- MAC address AAA will be sent using SwitchA's fa0/13 interface.
- MAC address BBB will be sent using SwitchA's fa0/14 interface.

Since we have multiple computers this is fine, both physical links on SwitchA will be used for our etherchannel so depending on how much traffic the computers send it will be close to a 1:1 ratio.

It's a different story for SwitchB since we only have one router with MAC address EEE. It will pick one of the physical interfaces so **ALL** traffic from the router will be sent down interface fa0/13 **OR** fa0/14. One of the physical links won't be used at all...

```
SwitchB(config)#port-channel load-balance dst-mac
```

If this is the case it's better to change the load balancing mechanism. If we switch it to destination MAC address on SwitchB traffic from our router to the computer will be load-balanced amongst the different physical interfaces because we have multiple computers with different destination MAC addresses.

This is all I have for you on etherchannels. Before you continue to the next chapter why not try to configure some PAgP and LACP etherchannels yourself? Here's a lab for you:

<http://gns3vault.com/Switching/pagp-lacp-etherchannel.html>

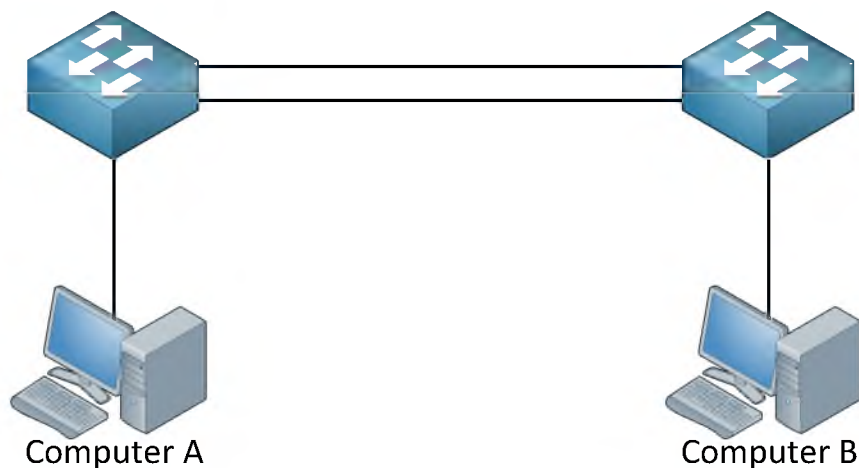
## 11. Spanning-Tree (STP)

I showed you the difference between hubs, bridges and switches. We talked about Virtual LANs (VLANs) and Trunks so there is one final switching topic we need to cover: Spanning-Tree. In our networks we want to have redundancy since we don't want to have a single point of failure. The drawback of redundancy in a switched network is that we introduce loops. Spanning-tree was designed to solve our loop-problem.

How do we get loops? By **having redundancy** in our switched network.



We want to have redundancy in our switched network since there should never be a single point of failure, you don't want your users screaming at you on the phone because a single cable was eaten by mice and the network went down. To solve this "single point of failure" cable we'll add another cable.



Now we have redundancy in our network, the problem we now have is something you should remember:

**Redundancy brings loops in our switched network.**

So why do we have a loop now? Let's imagine the following situation:

1. Computer A sends an ARP request because it's looking for the MAC address of computer B. An ARP request is a broadcast frame.
2. Switch A will forward this broadcast frame on all its links, except the link where the frame originated from.
3. Switch B will receive both broadcast frames.

Now what does switch B do with those broadcast frames?

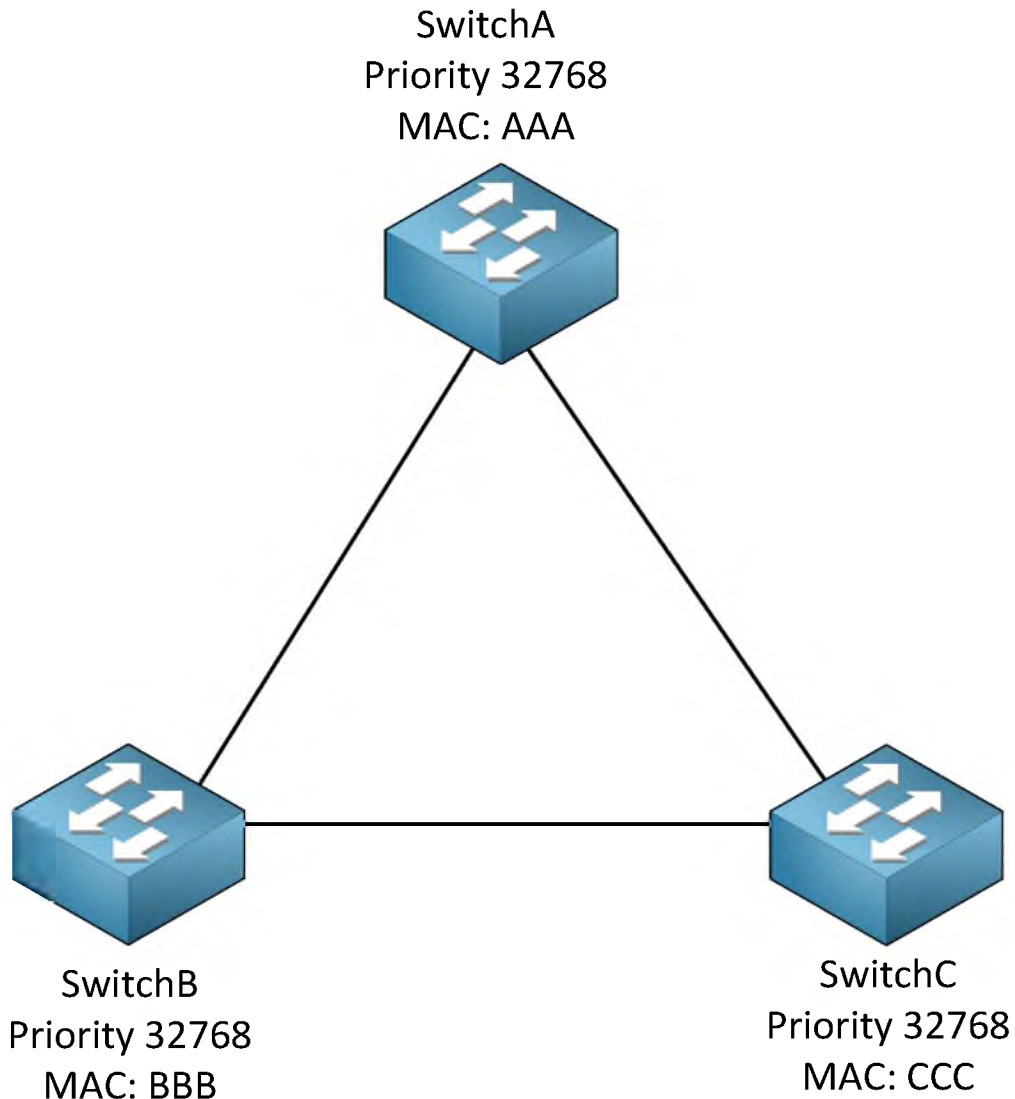
4. It will forward it out of every link except the link where it originated from.
5. This means that the frame that was received on Interface 1 will be forwarded on Interface 2.
6. The frame that was received on Interface 2 will be forwarded on Interface 1.

Do you see where this is going? We have a loop! Both switches will keep forwarding over and over until this will happen:

- You fix the loop by disconnecting one of the cables.
- Your switches will crash because they are overburdened with traffic (uh oh!)
- Ethernet frames don't have a TTL (Time to Live) field so frames will loop forever.

So here it is...spanning tree ready to save the day! All your switches will run spanning tree and this protocol will create a loop-free topology for us by blocking some ports.

Awesome! So how does it work? Let's take a look at the following example:



We have three switches and as you can see we have added redundancy by connecting the switches in a triangle, this also means we have a loop here. I have added the MAC addresses but simplified them for this example:

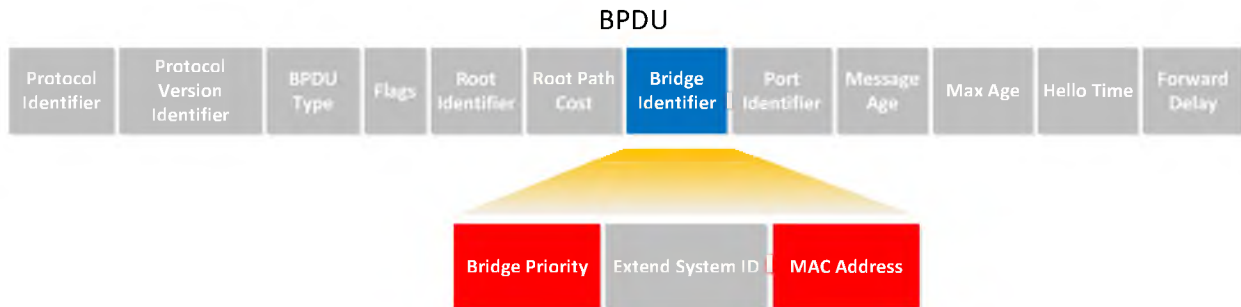
- Switch A: MAC AAA
- Switch B: MAC BBB
- Switch C: MAC CCC

Since spanning tree is enabled, all our switches will send a special frame to each other called a **BPDU**.

**"Bridge Protocol Data Unit"**



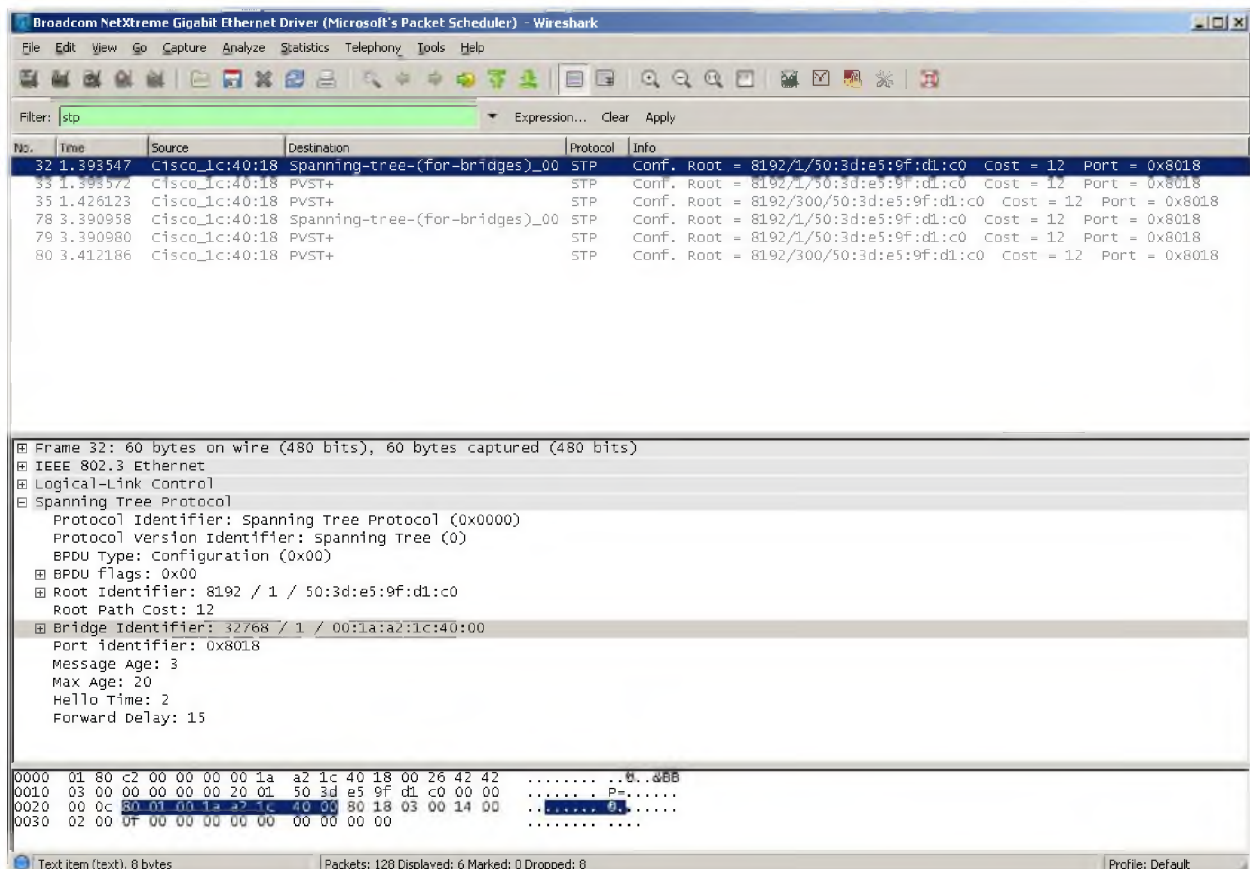
This is what it looks like:



There are a lot of fields in this frame but there are two fields that we are concerned about at this moment:

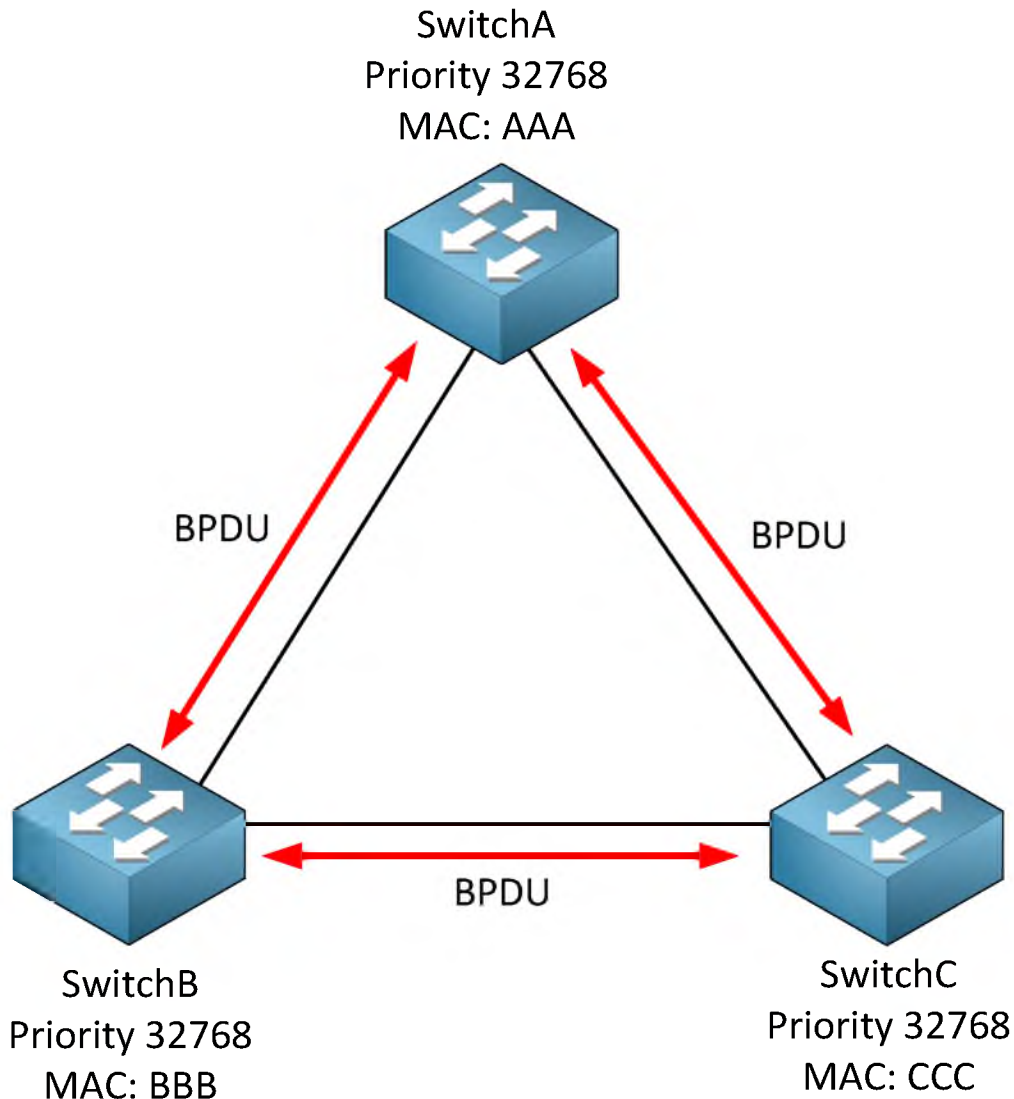
- MAC Address
- Priority

The priority and MAC address make up the **bridge ID**.



If you run Wireshark on a device that is connected to a Cisco switch you can capture a BPDU and see its contents.

This BPDU is being flooded between all of our switches as seen in this example:

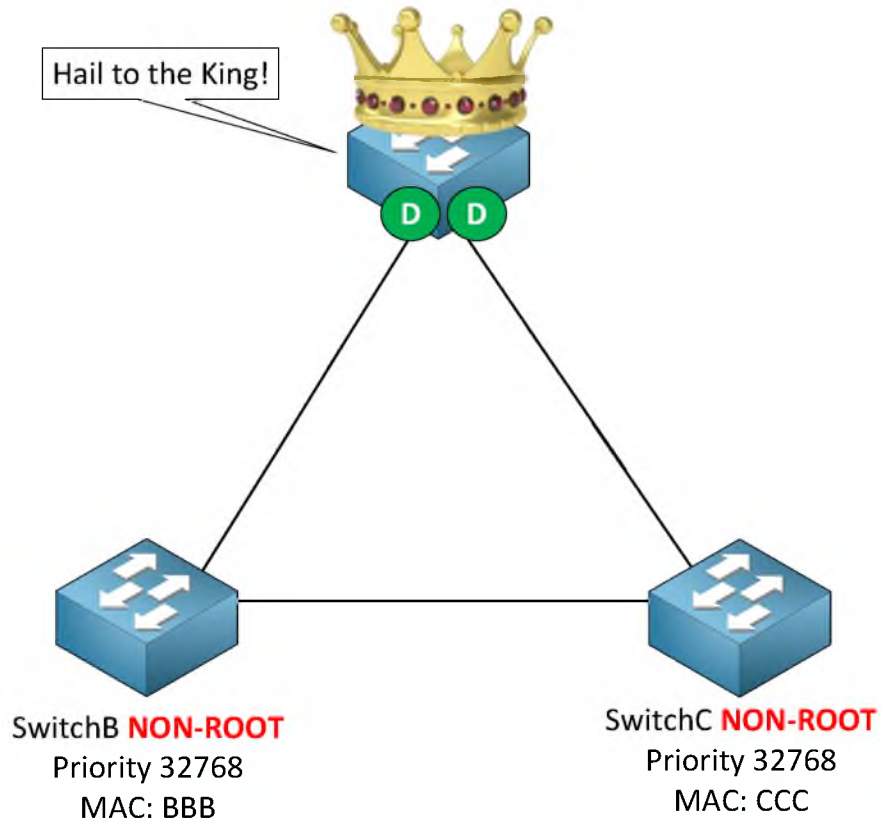


This is the information that spanning tree requires, so let's get going and do a spanning tree calculation!

- First of all spanning tree will elect a root bridge; this root-bridge will be the one that has the **best** "bridge ID".
- The lower the bridge ID, the better it is.
- By default the priority is a value of 32768 but you can change it if you want.

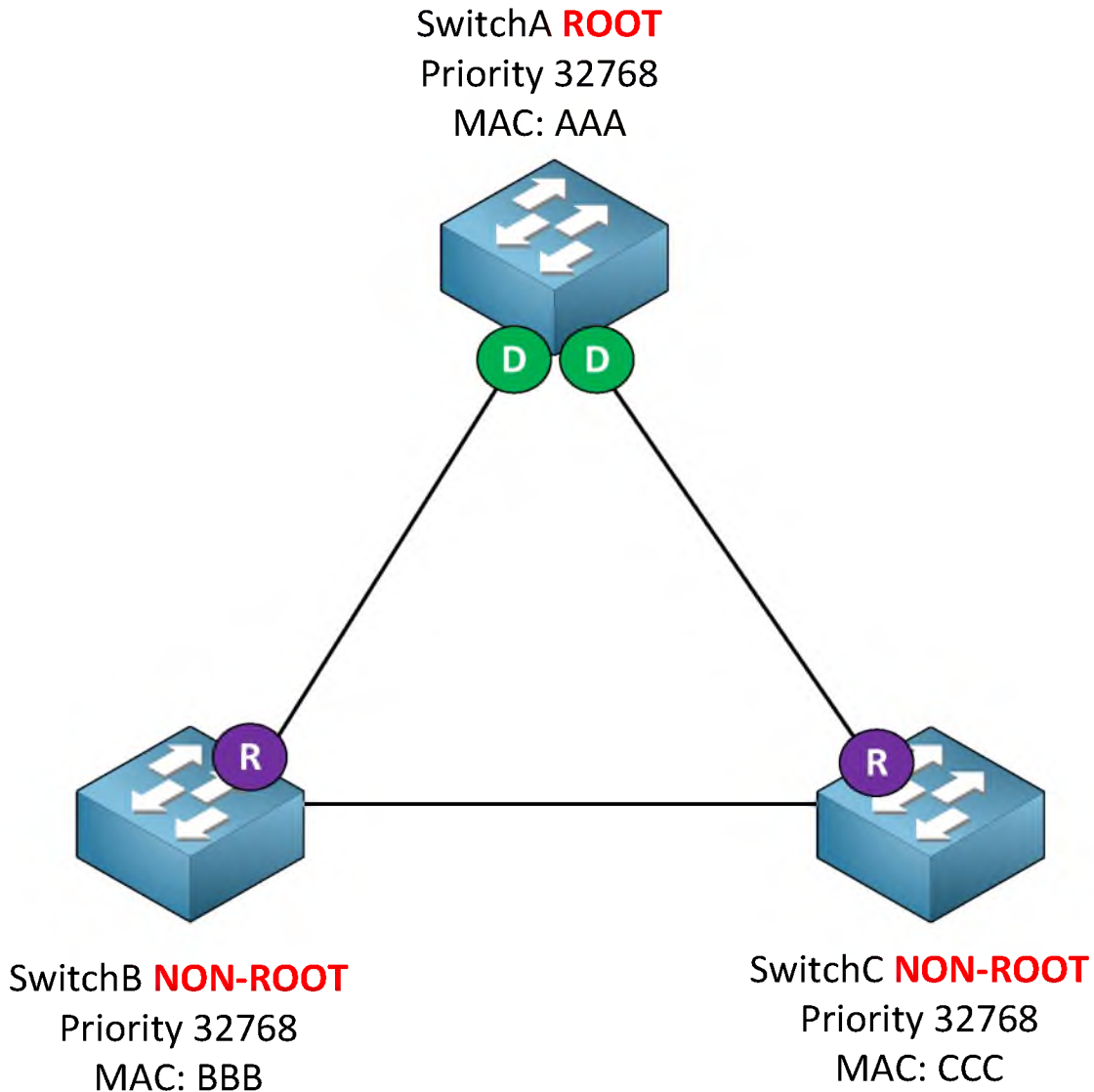
So who will become the root bridge? In our example switch A will become the root bridge! Priority and MAC address make up the bridge ID. Since the priority is the same on all switches it will be the MAC address that is the tiebreaker. Switch A has the lowest MAC address thus the best bridge ID and will become the root bridge.

The ports on our root bridge are always **designated** which means they are in a **forwarding** state. Don't worry about the different port states I'll get that to that later.



Take a look at my example, I've put the "D" of "designated" on the ports of our root-bridge.

Now we have agreed on the root bridge our next step for all our "non-root" bridges (so that's every switch that is not the root) will have to find the shortest path to our root bridge! The shortest path to the root bridge is called the "root port". Take a look at my example:



I've put an "R" for "root port" on switch B and switch C, that interface is the shortest path to get to the root bridge. In my example I've kept things simple but "shortest path" in spanning tree means it will actually look at the speed of the interface; obviously it's better to pass 3 Gigabit links than to pass a single 10mbit link...

If you want to determine the shortest path to the root we have to look at the cost of the interface.

| Cost             |     |
|------------------|-----|
| <b>10 Mbit</b>   | 100 |
| <b>100 Mbit</b>  | 19  |
| <b>1000 Mbit</b> | 4   |

As you can see the faster the interface the lower the cost. It's better to cross a path with 4 Gigabit links than crossing a single 100Mbit link since  $4 \times 4 = \text{COST } 16$  compared to COST 19 of a single 100Mbit link.

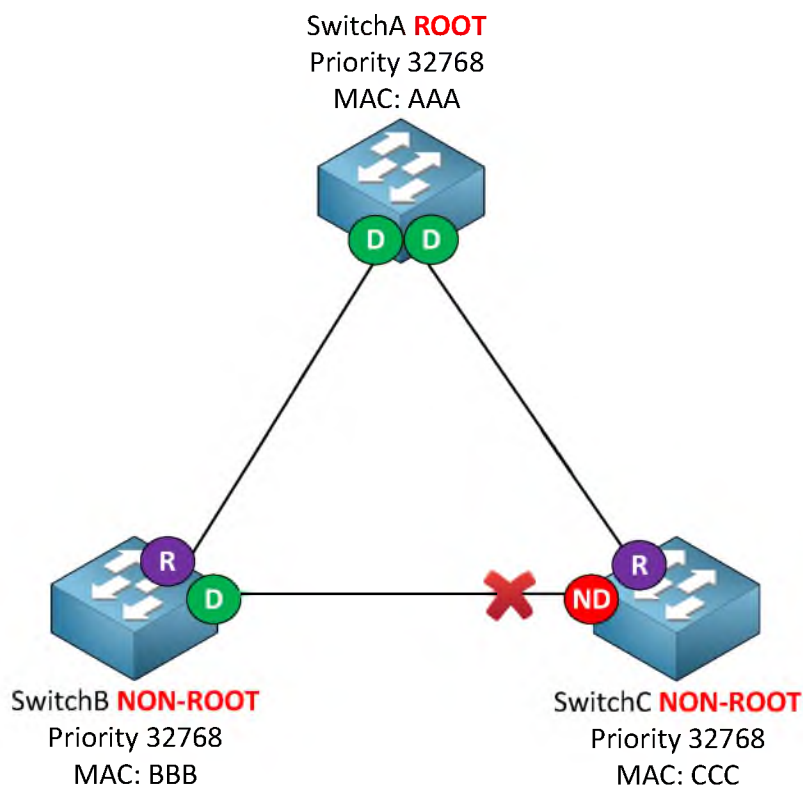
Excellent!...we have designated ports on our root bridge and root ports on our non-root bridges, we still have a loop however so we need to shut down a port between switch B and C to break that loop.

So which port are we going to shut down? The one on switch B or the one on switch C? We'll look again at the best bridge ID.

Bridge ID = MAC address + Priority.

Lower is better, both switches have the same priority but the MAC address of switch B is lower, this means that switch B will "win this battle".

Switch C is our loser here which means it will have to block its port, effectively breaking our loop! Take a look at my example:



If you look at the link between switch B and switch C you can see that the interface of switch C says "ND" which stands for non-designated. A non-designated port is blocked! By shutting down this interface we have solved our loop problem.

Because the priority is 32768 by default the MAC address is the tie-breaker for the root bridge election. Which switch do you think will become the root bridge?

Your brand-spanking-brand-new-just-out-of-the-box switch or that old dust collector that has been in the datacenter for 10 years?

The old switch probably has a lower MAC address and will become the root bridge...not a good idea right? I'll show you how to change the priority so the MAC address is no longer the tie-breaker!

Are you following me so far? Good! You just learned the basics of spanning-tree. Let's add some more detail to this story...

Victory at last...no more loops!

Are you getting this so far? Awesome since spanning tree can be a bit mind-boggling if you see this for the first time. You have now learned the basics of spanning tree!

Let's continue our spanning tree story and further enhance your knowledge. If you have played with some Cisco switches before you might have noticed that every time you plugged in a cable the led above the interface was orange and after a while became green. What is happening at this moment is that spanning tree is determining the state of the interface; this is what happens as soon as you plug in a cable:

- The port is in 'listening' mode for 15 seconds. In this phase it will receive and send BPDU's, still neither learning MAC addresses nor data transmission.
- The port is in 'learning' mode for 15 seconds. We are still sending and receiving BPDU's but now the switch will also learn MAC addresses, still no data transmission though.
- Now we go in forwarding mode and finally we can start transmitting data!

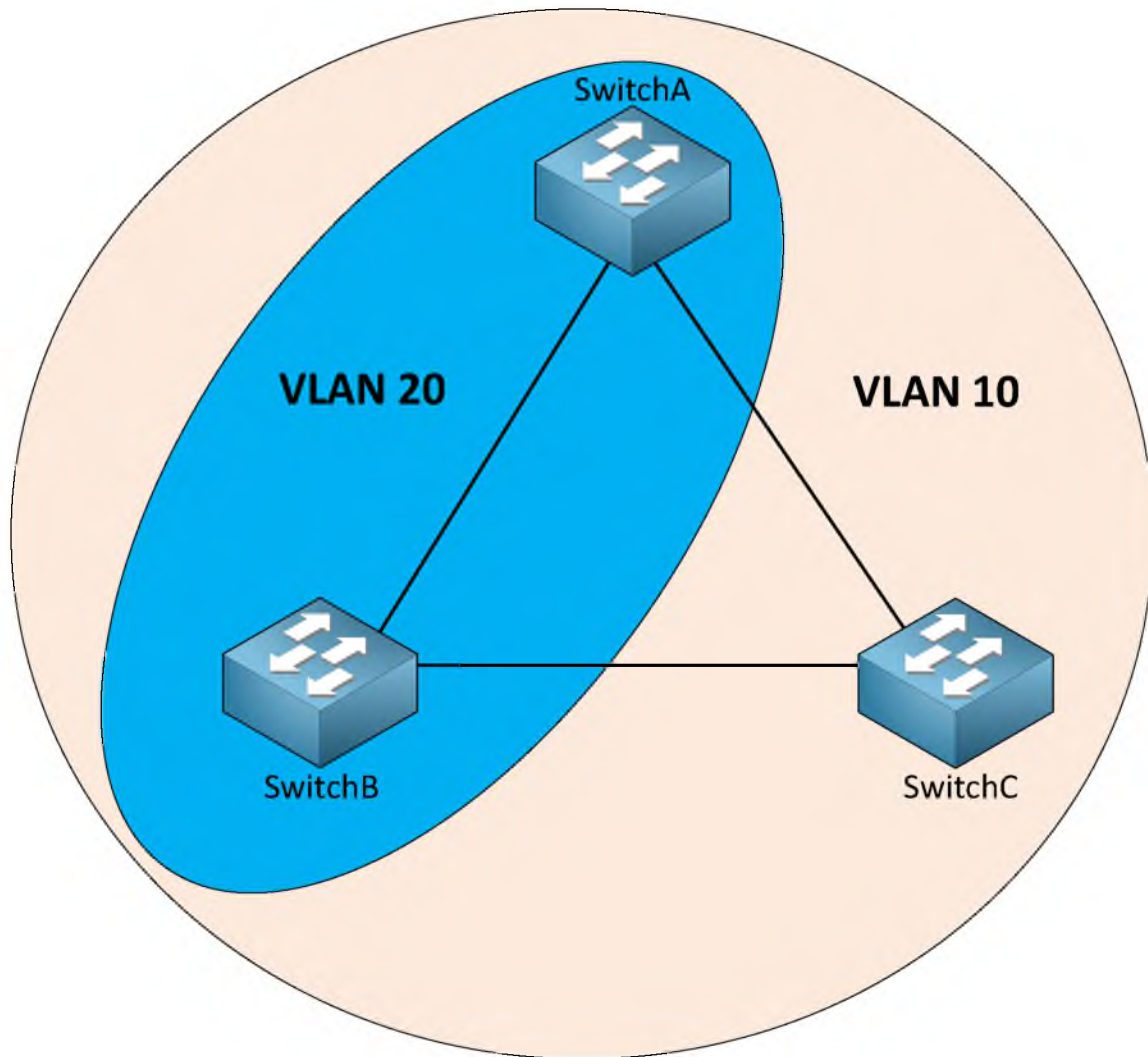
So all those steps took 30 seconds to go from listening to forwarding mode, that's VERY slow for networks nowadays.

You now know how spanning tree works and you also know how about the different port states. There is one last thing about spanning tree I'd like to teach you. There is no "single" version of spanning tree but we have different 'flavors', let me sum them up for you:

- **CST:** Common (classic) spanning tree
- **PVST:** Per VLAN spanning tree
- **RST:** Rapid spanning tree
- **PVRST:** Per VLAN rapid spanning tree
- **MST:** Multiple spanning tree

That's a lot of different flavors with not so easy to remember acronyms...lucky for you, you don't have to know all the details about the different versions for your CCNA but let's discuss what you DO need to know about them.

To start off let me show you a picture of our spanning tree topology but now we are working with VLANs. We only have 2 VLANs, VLAN 10 and 20. As you can see switch A, B and C have VLAN 10. VLAN 20 is only configured on switch A and B.



Now let me ask you 2 questions:

- Is there a loop in VLAN 10?
- Is there a loop in VLAN 20?

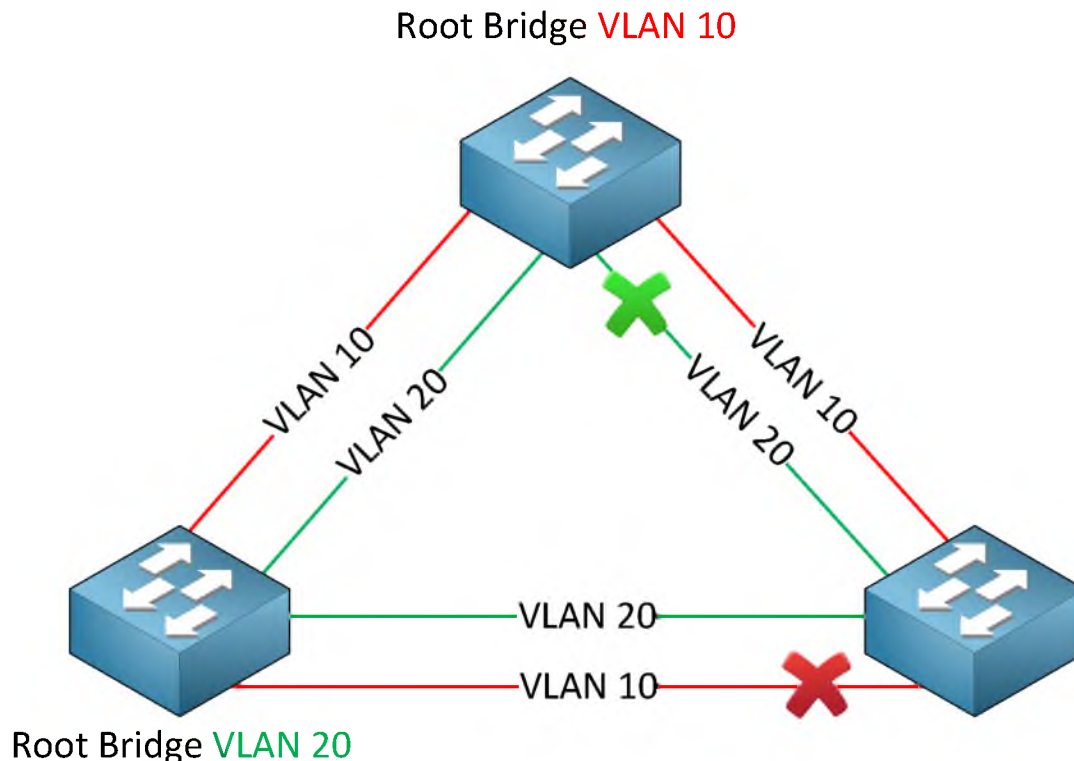
The answer is, in VLAN 20 we don't have a loop since there's only a single link between Switch A and switch B. In VLAN 10 we do have a loop since all 3 switches are in this VLAN. There is a big difference between our physical topology and our logical topology because of our VLANs. So how do we deal with this with spanning tree? Which ports are we going to block and why?

The answer is simple...we are going to run a separate spanning tree calculation for EACH VLAN! This is what we call **per VLAN spanning tree**. So now you know why we have



**common spanning tree** which calculates a single spanning tree instance for our switches and why we have **per VLAN spanning tree** which calculates a spanning tree instance for each VLAN.

For each VLAN you can have a different root bridge if you like, this might be useful as seen in the following picture:

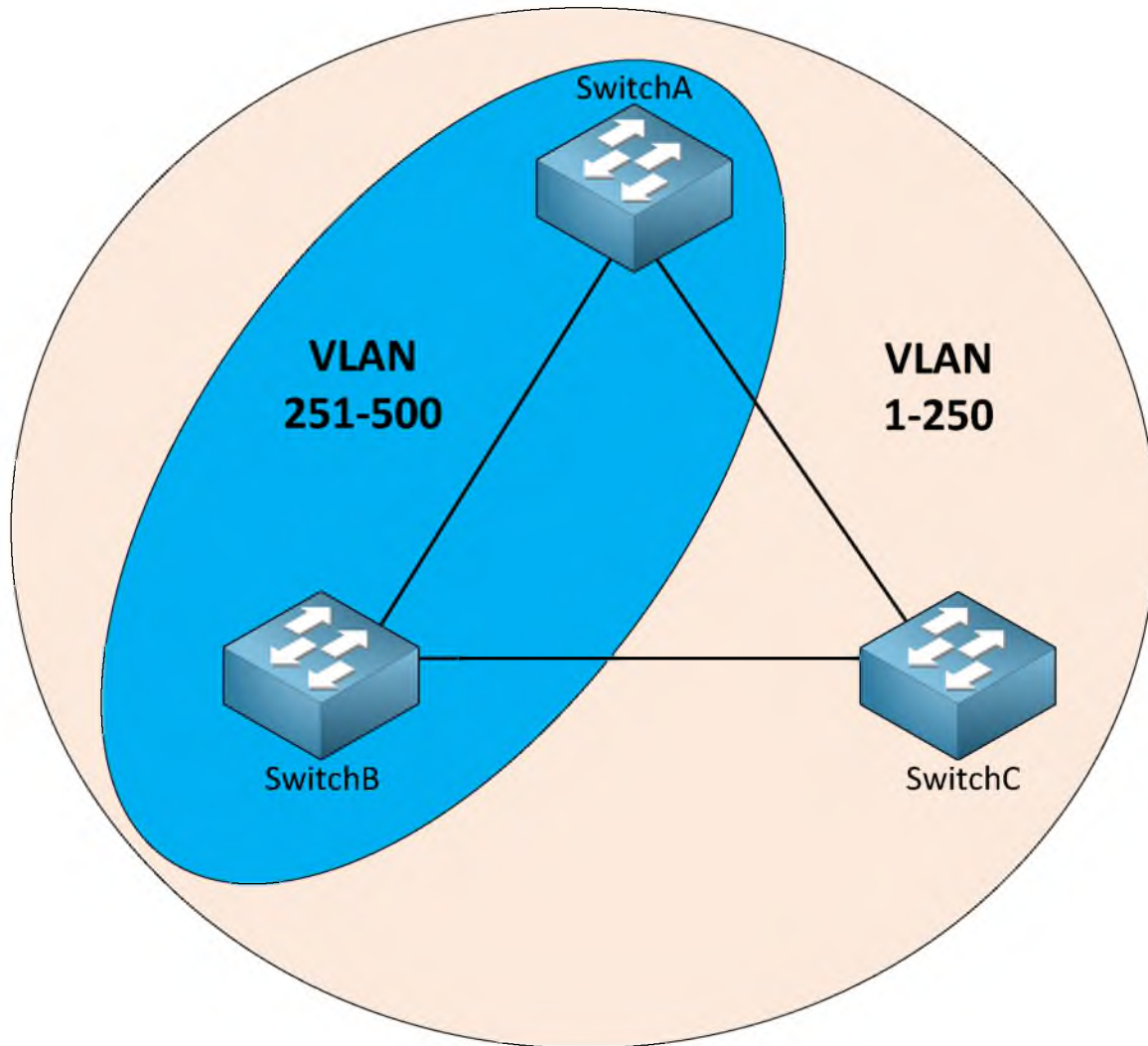


We have VLAN 10 and VLAN 20 and instead of making the one on top the root bridge for both VLANs we made the one on top the root bridge for VLAN 10 and the switch on the left is the root bridge for VLAN 20. As you can see different ports will be blocked per VLAN. If you have the same root bridge for both VLANs you will block a interface for both VLANs...which means no traffic will ever pass that link. This way you can do some load-balancing.

We also have **rapid spanning tree** and **per VLAN rapid spanning tree**. The difference with the other types of spanning tree is in the keyword 'rapid'. The old spanning tree standard is too slow for a network nowadays which is why there is a rapid version now. You don't have to know the details for CCNA but if you plan to study the CCNP SWITCH exam you'll see it in detail. Just remember that it's much faster for now.

The last version we have left is MST also known as **multiple spanning tree**. You know we calculate a spanning tree instance per VLAN with the per VLAN spanning tree versions but let's say you have 500 VLANs configured as in the following picture:



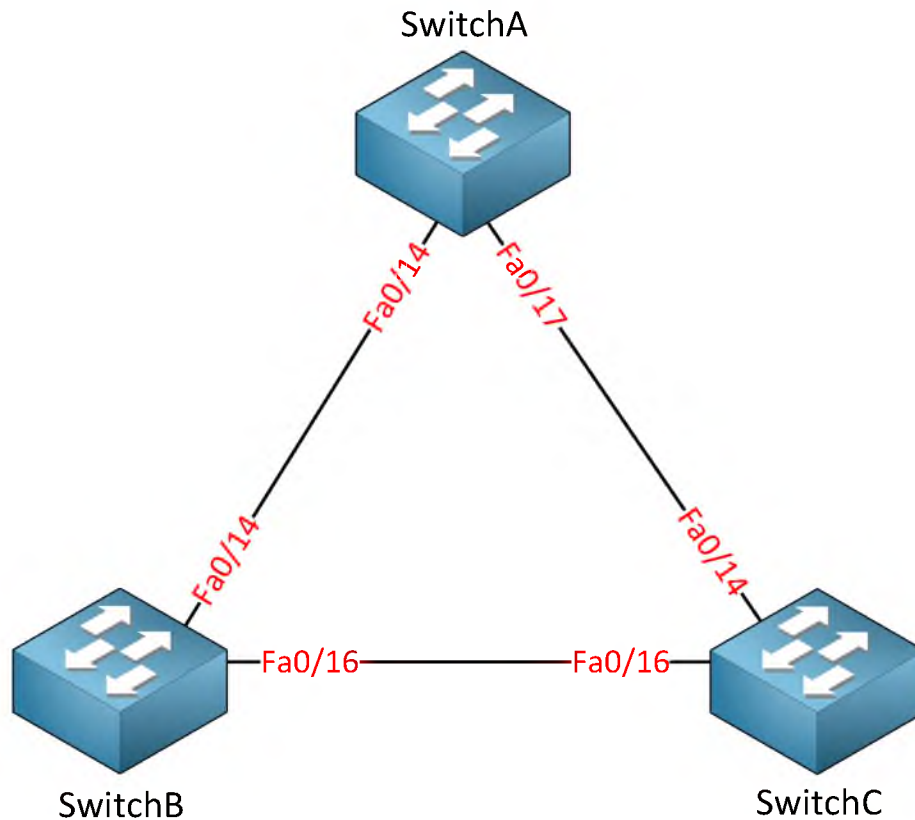


VLAN 1-250 has been configured on all switches. VLAN 251 - 500 are configured on switch A and B.

Is it really useful to do a spanning tree calculation for each VLAN? That's 500 spanning tree calculations! But as you can see there are only two different topologies...VLAN 1 - 250 and VLAN 251 - 500. With MST you can assign VLAN numbers to a spanning tree instance. In this case we would have one spanning tree calculation for VLAN 1- 250 and another one for 251 - 500. That's only two calculations instead of 500!

Forget about the details on MST for now, it's not on your CCNA exam but you will encounter if you continue for your CCNP SWITCH exam.

Now you have an idea what spanning-tree is about, let's take a look at it on our switches!



This is the topology we will use. Spanning-tree is enabled by default; let's start by checking some show commands.

```
SwitchA#show spanning-tree
```

```
VLAN0001
```

```
Spanning tree enabled protocol ieee
```

```

Root ID      Priority    32769
Address      000f.34ca.1000
Cost         19
Port         19 (FastEthernet0/17)
Hello Time    2 sec  Max Age 20 sec  Forward Delay 15 sec

```

```

Bridge ID    Priority    32769 (priority 32768 sys-id-ext 1)
Address      0011.bb0b.3600
Hello Time    2 sec  Max Age 20 sec  Forward Delay 15 sec
Aging Time    300

```

| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| ---       | ---  | --- | ---  | ---      | ---  |
| Fa0/14    | Desg | FWD | 19   | 128.16   | P2p  |
| Fa0/17    | Root | FWD | 19   | 128.19   | P2p  |

The **show spanning-tree** command is the most important show command to remember.

There's quite some stuff here so I'm going to break it down for you!

```
VLAN0001
Spanning tree enabled protocol ieee
```

We are looking at the spanning-tree information for VLAN 1. Spanning-tree has multiple versions and the default version on Cisco switches is PVST (Per VLAN spanning-tree). This is the spanning-tree for VLAN 1.

```
Root ID      Priority    32769
            Address    000f.34ca.1000
            Cost      19
            Port      19 (FastEthernet0/17)
```

Here you see the **information of the root bridge**. You can see that it has a priority of 32769 and its MAC address is 000f.34ca.1000. From the perspective of SwitchA it has a cost of 19 to reach the root bridge. The port that leads to the root bridge is called the root port and for SwitchA this is fa0/17.

```
Bridge ID  Priority    32769 (priority 32768 sys-id-ext 1)
            Address    0011.bb0b.3600
```

This part shows us the **information about the local switch**, SwitchA in our case. There's something funny about the priority here....you can see it show two things:

- Priority 32769
- Priority 32768 sys-id-ext 1

The **sys-id-ext** value that you see is the VLAN number. The priority is 32768 but spanning-tree will add the VLAN number so we end up with priority value 32769. Last but not least we can see the MAC address of SwitchA which is 0011.bb0b.3600.

```
Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec
```

Here's some information on the different times that spanning-tree uses:

- **Hello time:** every 2 seconds a BPDU is sent.
- **Max Age:** If we don't receive BPDUs for 20 seconds we know something has changed in the network and we need to re-check the topology.
- **Forward Delay:** It takes 15 seconds to move to the forwarding state.

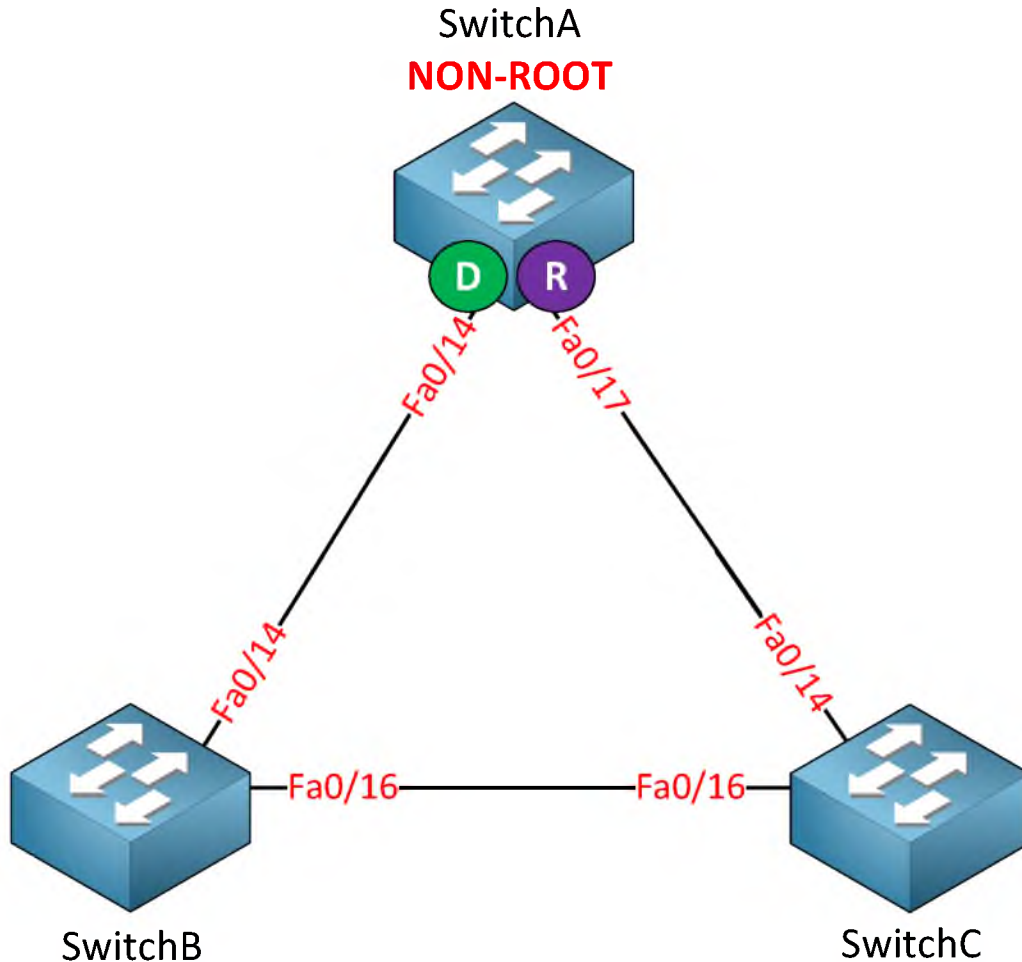
| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| ---       |      |     |      |          |      |
| Fa0/14    | Desg | FWD | 19   | 128.16   | P2p  |
| Fa0/17    | Root | FWD | 19   | 128.19   | P2p  |

The last part of the show spanning-tree commands shows us the interfaces and their status. SwitchA has two interfaces:

- Fa0/24 is a **designated** port and in **(FWD) forwarding mode**.
- Fa0/17 is a **root** port and in **(FWD) forwarding mode**.

The **prio.nbr** you see here is the **port priority** that I explained earlier. We'll play with this in a bit.

Because only non-root switches have a root-port I can conclude that SwitchA is a non-root switch. I know that fa0/17 on SwitchA leads to the root bridge.



For the sake of having a good overview I just added what we saw in the show spanning-tree command in the picture above. We know that SwitchA is a non-root, fa0/14 is a designated port and fa0/17 is a root port.

```
SwitchB#show spanning-tree
```

```
VLAN0001
```

```
Spanning tree enabled protocol ieee
```

```
Root ID      Priority      32769
Address      000f.34ca.1000
Cost         19
Port         18 (FastEthernet0/16)
Hello Time    2 sec    Max Age 20 sec    Forward Delay 15 sec
```

```
Bridge ID    Priority      32769 (priority 32768 sys-id-ext 1)
Address      0019.569d.5700
Hello Time    2 sec    Max Age 20 sec    Forward Delay 15 sec
Aging Time    300
```

| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| ---       | ---  | --- | ---  | ---      | ---  |
| Fa0/14    | Altn | BLK | 19   | 128.16   | P2p  |
| Fa0/16    | Root | FWD | 19   | 128.18   | P2p  |

Let's take a look at SwitchB...what do we have here?

|         |          |                       |
|---------|----------|-----------------------|
| Root ID | Priority | 32769                 |
| Address |          | 000f.34ca.1000        |
| Cost    |          | 19                    |
| Port    |          | 18 (FastEthernet0/16) |

Here we see information about the root bridge. This information is similar to what we saw on SwitchA. The root port for SwitchB seems to be fa0/16.

|           |          |                                     |
|-----------|----------|-------------------------------------|
| Bridge ID | Priority | 32769 (priority 32768 sys-id-ext 1) |
| Address   |          | 0019.569d.5700                      |

This is the information about SwitchB. The priority is the same as on SwitchA, only the MAC address (0019.569d.5700) is different.

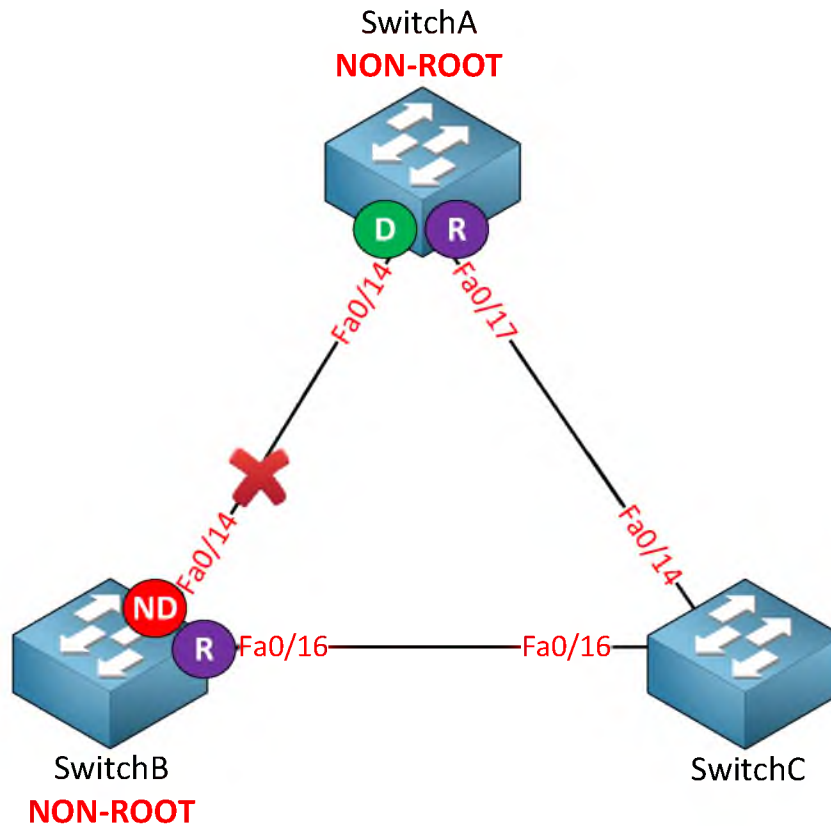
| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| ---       | ---  | --- | ---  | ---      | ---  |
| Fa0/14    | Altn | BLK | 19   | 128.16   | P2p  |
| Fa0/16    | Root | FWD | 19   | 128.18   | P2p  |

This part looks interesting; there are two things we see here:

- Interface fa0/14 is an **alternate** port and in **(BLK) blocking** mode.
- Interface fa0/16 is a **root** port and in **(FWD) forwarding** mode.



*In the beginning of this chapter I explained that a blocked port is the non-designated port. When we run PVST (Per VLAN Spanning-Tree) this port will show up as the "alternate" port.*



With the information we just found on SwitchB we can add more items to our topology picture. We are almost finished!

```
SwitchC#show spanning-tree
```

```
VLAN0001
```

```
Spanning tree enabled protocol ieee
```

```
Root ID      Priority    32769
```

```
Address      000f.34ca.1000
```

```
This bridge is the root
```

```
Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
```

```
Bridge ID    Priority    32769 (priority 32768 sys-id-ext 1)
```

```
Address      000f.34ca.1000
```

```
Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
```

```
Aging Time 300
```

| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| Fa0/14    | Desg | FWD | 19   | 128.14   | P2p  |
| Fa0/16    | Desg | FWD | 19   | 128.16   | P2p  |

Let's break down what we have here:

|                                |          |                |
|--------------------------------|----------|----------------|
| Root ID                        | Priority | 32769          |
|                                | Address  | 000f.34ca.1000 |
| <b>This bridge is the root</b> |          |                |

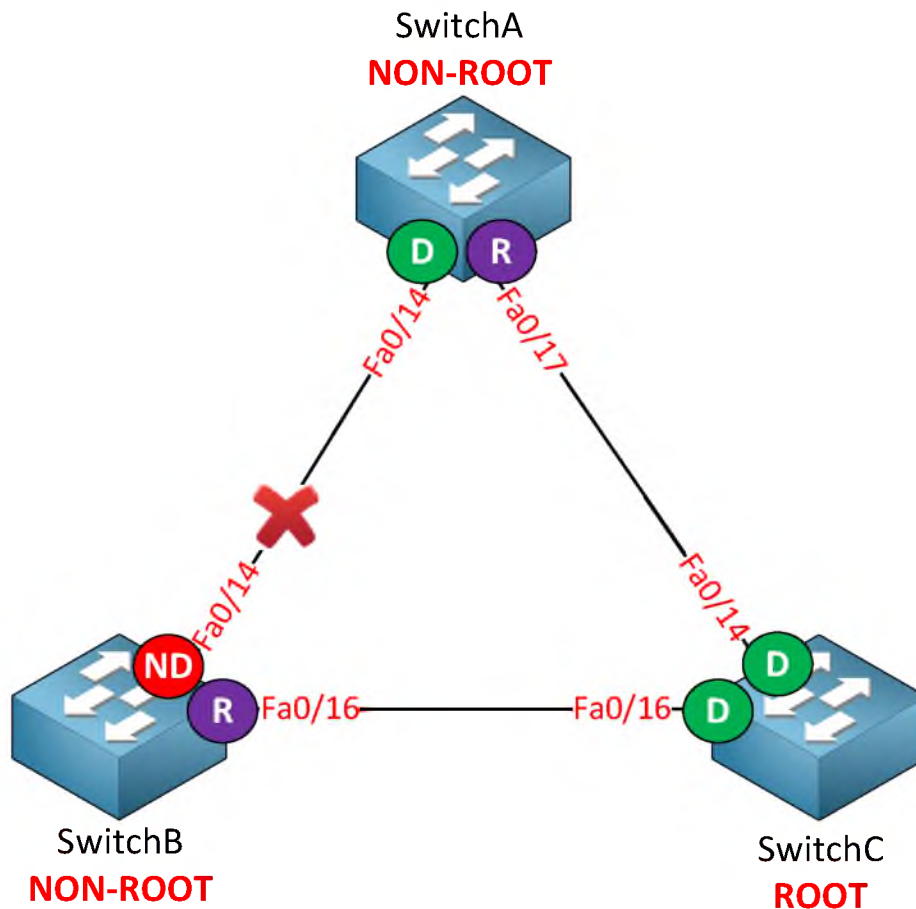
Bingo...SwitchC is the root bridge in this network. We already knew that because SwitchA and SwitchB are both non-root but this is how we verify it by looking at SwitchC.

|           |          |                                     |
|-----------|----------|-------------------------------------|
| Bridge ID | Priority | 32769 (priority 32768 sys-id-ext 1) |
|           | Address  | 000f.34ca.1000                      |

We can also see the MAC address of SwitchC.

| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| Fa0/14    | Desg | FWD | 19   | 128.14   | P2p  |
| Fa0/16    | Desg | FWD | 19   | 128.16   | P2p  |

Both interfaces on SwitchC are **designated ports** and in **(FWD) forwarding** mode.



Our picture is now complete. We successfully found out what the spanning-tree topology looks

like by using the show spanning-tree command! Why was SwitchC chosen as the root bridge? We have to look at the bridge identifier for the answer:

```
SwitchA#show spanning-tree | begin Bridge ID
Bridge ID  Priority    32769  (priority 32768 sys-id-ext 1)
Address    0011.bb0b.3600
```

```
SwitchB#show spanning-tree | begin Bridge ID
Bridge ID  Priority    32769  (priority 32768 sys-id-ext 1)
Address    0019.569d.5700
```

```
SwitchC#show spanning-tree | begin Bridge ID
Bridge ID  Priority    32769  (priority 32768 sys-id-ext 1)
Address    000f.34ca.1000
```

The priority is the same on all switches (32768) so we have to look at the MAC addresses:

- SwitchA: 0011.bb0b.3600
- SwitchB: 0019.569d.5700
- SwitchC: 000f.34ca.1000

SwitchC has the lowest MAC address so that's why it became root bridge. Why was the fa0/14 interface on SwitchB blocked and not the fa0/14 interface on SwitchA? Once again we have to look at the bridge identifier. The priority is 32768 on both switches so we have to compare the MAC address:

- SwitchA: 0011.bb0b.3600
- SwitchB: 0019.569d.5700

SwitchA has a lower MAC address and thus a better bridge identifier. That's why SwitchB lost this battle and has to shut down its fa0/14 interface.

What if I want another switch to become root bridge? For example SwitchA:

```
SwitchA(config)#spanning-tree vlan 1 root primary
```

There are two methods how I can change the root bridge. The **spanning-tree vlan root primary** command is the first one. This is a **macro** that looks at the current priority of the root bridge and changes your running-config to lower your own priority. Because we use PVST (Per VLAN Spanning-Tree) we can change this for each VLAN.

```
SwitchA#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID    Priority    24577
             Address    0011.bb0b.3600
             This bridge is the root
             Hello Time  2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    24577  (priority 24576 sys-id-ext 1)
```



You can see that SwitchA is now the root bridge because its priority has been changed to 24576.

```
SwitchA#show run | include priority
spanning-tree vlan 1 priority 24576
```

If you look at the running-config you can see that that the spanning-tree vlan root primary command changed the priority for us.

```
SwitchA(config)#spanning-tree vlan 1 priority ?
<0-61440> bridge priority in increments of 4096

SwitchA(config)#spanning-tree vlan 1 priority 4096
```

Changing the priority manually is the second method.

Just type in the **spanning-tree vlan priority** command and set it to whatever value you like.

```
SwitchA#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID    Priority    4097
             Address     0011.bb0b.3600
             This bridge is the root
             Hello Time  2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    4097  (priority 4096 sys-id-ext 1)
             Address     0011.bb0b.3600
```

We can verify this by checking the show spanning-tree command once again. Before we continue there's one more question...why was the interface fa0/14 on SwitchB

Because SwitchA is now the root bridge our spanning-tree topology looks different.

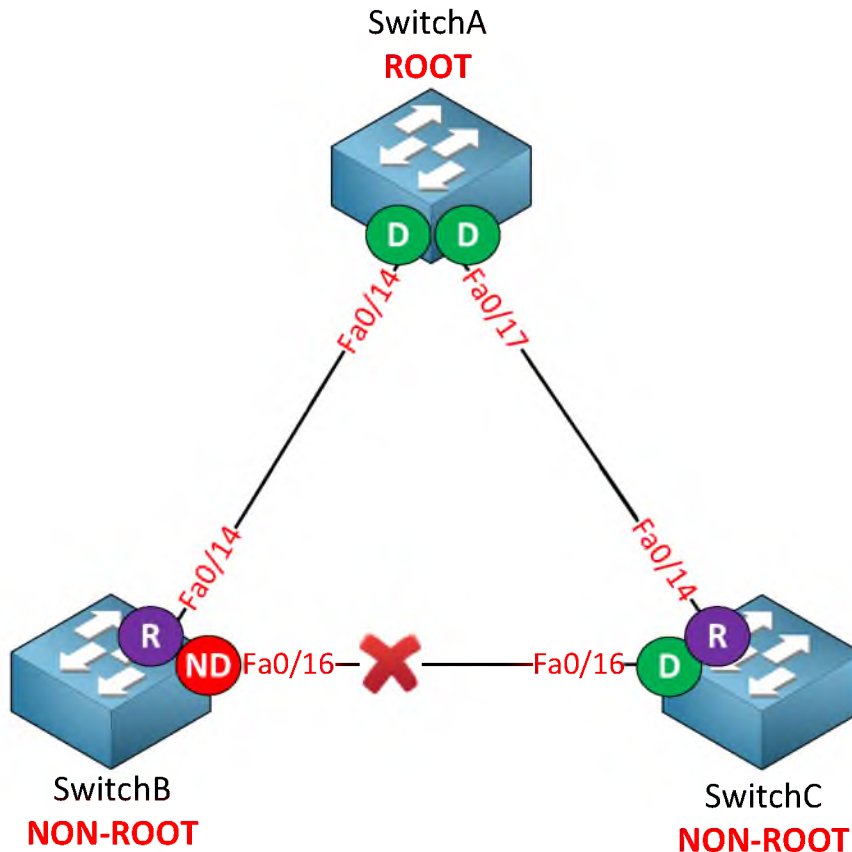
```
SwitchA#show spanning-tree | begin Interface
Interface          Role Sts Cost          Prio.Nbr Type
-----
Fa0/14             Desg FWD 19           128.16 P2p
Fa0/17             Desg FWD 19           128.19 P2p
```

```
SwitchB#show spanning-tree | begin Interface
Interface          Role Sts Cost          Prio.Nbr Type
-----
Fa0/14             Root FWD 19           128.16 P2p
Fa0/16             Altn BLK 19           128.18 P2p
```

```
SwitchC#show spanning-tree | begin Interface
```

| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| Fa0/14    | Root | FWD | 19   | 128.14   | P2p  |
| Fa0/16    | Desg | FWD | 19   | 128.16   | P2p  |

This is all the information we need. Let's update our topology picture...



Let's play some more with spanning-tree! What if I want to change the root port on SwitchB so it reaches the root bridge through SwitchC? From SwitchB's perspective it can reach the root bridge through fa0/14 (cost 19) or by going through fa0/16 (cost 19+19 = 38). Let's change the cost and see what happens.

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#spanning-tree cost 500
```

Let's change the cost of the fa0/14 interface by using the **spanning-tree cost** command.

```
SwitchB#show spanning-tree | begin Interface
```

| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| ---       |      |     |      |          |      |
| Fa0/14    | Altn | BLK | 500  | 128.16   | P2p  |
| Fa0/16    | Root | FWD | 19   | 128.18   | P2p  |

You can see that the fa0/14 now has a cost of 500 and it has been blocked. Fa0/16 is now the root port.

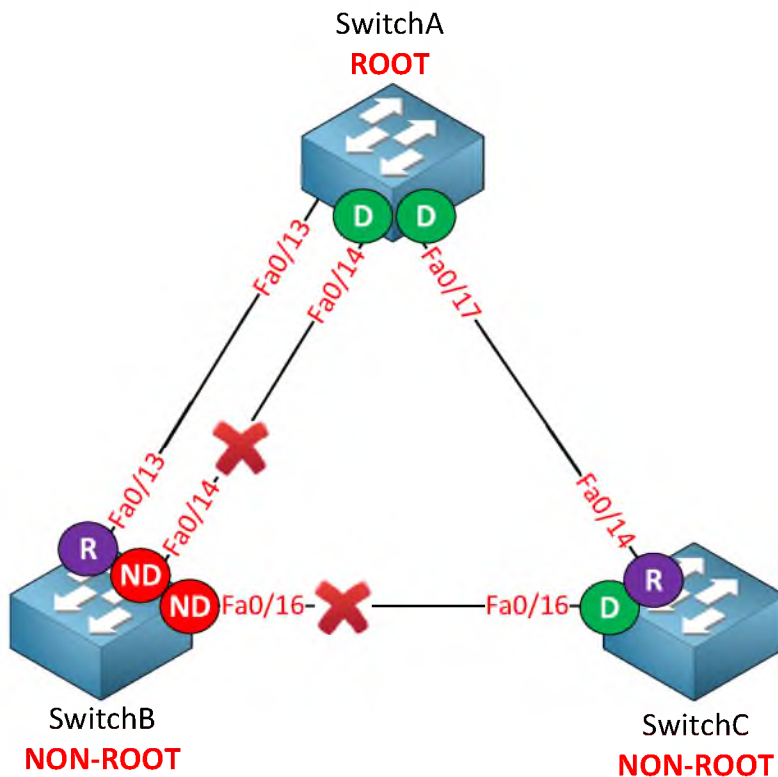
```
SwitchB#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID    Priority    4097
            Address      0011.bb0b.3600
            Cost        38
```

To reach the root the total cost is now 38.

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#no spanning-tree cost 500
```

Let's get rid of the higher cost before we continue.



I have added another cable between SwitchA and SwitchB. In the picture above you can see that fa0/13 is now the root port. Fa0/14 has been blocked.

```
SwitchB#show spanning-tree | begin Interface
```

| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| ---       |      |     |      |          |      |
| Fa0/13    | Root | FWD | 19   | 128.15   | P2p  |
| Fa0/14    | Altn | BLK | 19   | 128.16   | P2p  |
| Fa0/16    | Altn | BLK | 19   | 128.18   | P2p  |

Why did fa0/13 become the root port instead of fa0/14? The cost to reach the root bridge is the same on both interfaces. The answer lies in the port priority:

- Fa0/13: port priority 128.15
- Fa0/14: port priority 128.16

The “128” is a default value which we can change. 15 and 16 are the port numbers, each interface is assigned a port number. Fa0/13 has a lower port priority so that’s why it was chosen.

Let’s change the port priority and see what happens:

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#spanning-tree port-priority 16
```

Note that I’m changing the port priority on SwitchA, not on SwitchB. At the moment SwitchB is receiving a BPDU on its fa0/13 and fa0/14 interfaces. Both BPDUs are the same. By changing the port priority on SwitchA, SwitchB will receive a BPDU with a better port priority on its fa0/14 interface.

```
SwitchA#show spanning-tree | begin Interface
```

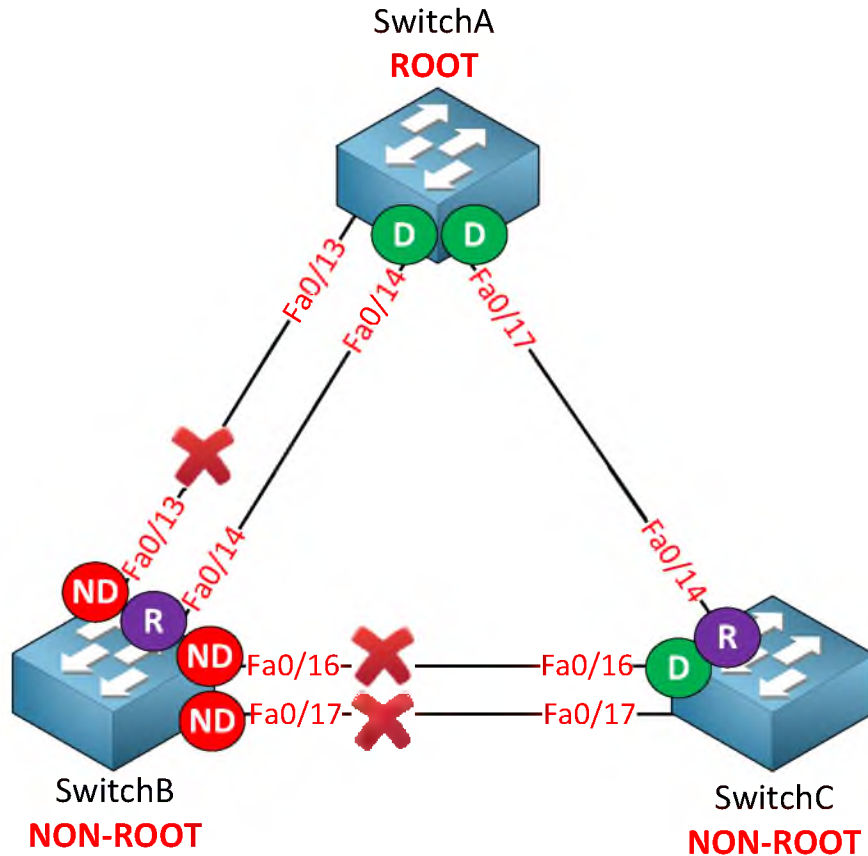
| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| ---       |      |     |      |          |      |
| Fa0/13    | Desg | FWD | 19   | 128.15   | P2p  |
| Fa0/14    | Desg | FWD | 19   | 16.16    | P2p  |

You can see the port priority has been changed on SwitchA.

```
SwitchB#show spanning-tree | begin Interface
```

| Interface | Role | Sts | Cost | Prio.Nbr | Type |
|-----------|------|-----|------|----------|------|
| ---       |      |     |      |          |      |
| Fa0/13    | Altn | BLK | 19   | 128.15   | P2p  |
| Fa0/14    | Root | FWD | 19   | 128.16   | P2p  |
| Fa0/16    | Altn | BLK | 19   | 128.18   | P2p  |

Interface fa0/14 on SwitchB is now the root port!



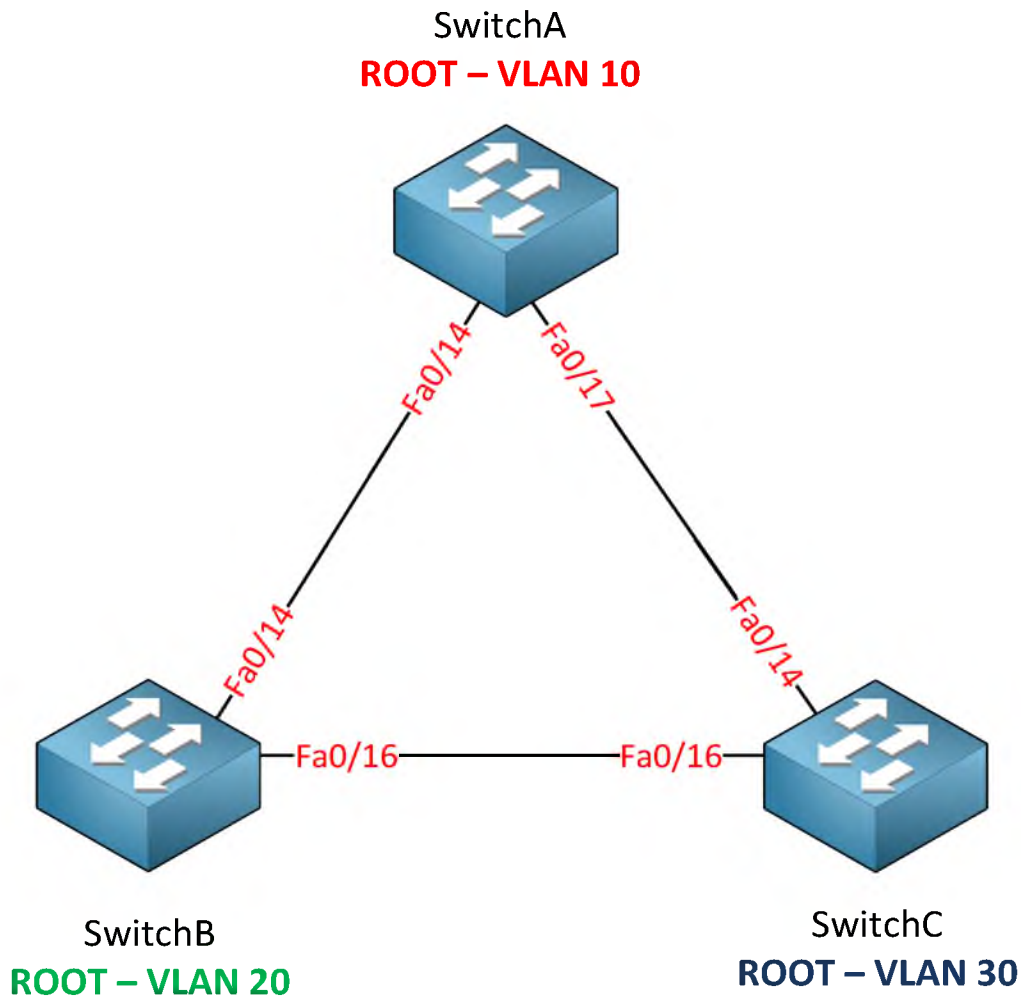
In the picture above I added another cable between SwitchB and SwitchC. This interface will also become an alternate port and it will be blocked.

```
SwitchB#show spanning-tree | begin Interface
Interface          Role Sts Cost      Prio.Nbr Type
-----
Fa0/13             Altn BLK 19        128.15  P2p
Fa0/14             Root FWD 19        128.16  P2p
Fa0/16             Altn BLK 19        128.18  P2p
Fa0/17           Altn BLK 19        128.19  P2p
```

We can verify our configuration here. Just another blocked port...

Are you following me so far? I hope so! If you are having trouble understanding the different spanning-tree commands I recommend you to build the same topology as the one I'm using above and to take a look at your own spanning-tree topology. Play with the priority, cost and port priority to see what the effect will be.

We'll continue by looking at how spanning-tree deals with multiple VLANs:



Let's get back to the basics. I have resetted all switches back to factory default settings because I want to show you how spanning-tree works with multiple VLANs. In the previous example we were only using VLAN 1. Now I'm going to add VLAN 10, 20 and 30 and each switch will become root bridge for a VLAN.

```
SwitchA(config)#vlan 10
SwitchA(config-vlan)#vlan 20
SwitchA(config-vlan)#vlan 30
```

```
SwitchB(config)#vlan 10
SwitchB(config-vlan)#vlan 20
SwitchB(config-vlan)#vlan 30
```

```
SwitchC(config)#vlan 10
SwitchC(config-vlan)#vlan 20
SwitchC(config-vlan)#vlan 30
```

First I'm going to create all VLANs. If you are running VTP server/client mode you only have to do this on one switch.

```
SwitchA(config)#interface fa0/14
SwitchA(config-if)#switchport trunk encapsulation dot1q
SwitchA(config-if)#switchport mode trunk
SwitchA(config)#interface fa0/17
SwitchA(config-if)#switchport trunk encapsulation dot1q
SwitchA(config-if)#switchport mode trunk
```

```
SwitchB(config)#interface fa0/14
SwitchB(config-if)#switchport trunk encapsulation dot1q
SwitchB(config-if)#switchport mode trunk
SwitchB(config)#interface fa0/16
SwitchB(config-if)#switchport trunk encapsulation dot1q
SwitchB(config-if)#switchport mode trunk
```

```
SwitchC(config)#interface fa0/14
SwitchC(config-if)#switchport trunk encapsulation dot1q
SwitchC(config-if)#switchport mode trunk
SwitchC(config)#interface fa0/16
SwitchC(config-if)#switchport trunk encapsulation dot1q
SwitchC(config-if)#switchport mode trunk
```

Make sure the interfaces between the switches are trunks. Mine were access interfaces so I changed them to trunk mode myself.

```
SwitchA#show spanning-tree summary | begin Name
```

| Name     | Blocking | Listening | Learning | Forwarding | STP Active |
|----------|----------|-----------|----------|------------|------------|
| VLAN0001 | 0        | 0         | 0        | 2          | 2          |
| VLAN0010 | 0        | 0         | 0        | 2          | 2          |
| VLAN0020 | 0        | 0         | 0        | 2          | 2          |
| VLAN0030 | 0        | 0         | 0        | 2          | 2          |
| 4 vlans  | 0        | 0         | 0        | 8          | 8          |

```
SwitchB#show spanning-tree summary | begin Name
```

| Name     | Blocking | Listening | Learning | Forwarding | STP Active |
|----------|----------|-----------|----------|------------|------------|
| VLAN0001 | 1        | 0         | 0        | 1          | 2          |
| VLAN0010 | 1        | 0         | 0        | 1          | 2          |
| VLAN0020 | 1        | 0         | 0        | 1          | 2          |
| VLAN0030 | 1        | 0         | 0        | 1          | 2          |
| 4 vlans  | 4        | 0         | 0        | 4          | 8          |

```
SwitchC#show spanning-tree summary | begin Name
```

| Name     | Blocking | Listening | Learning | Forwarding | STP Active |
|----------|----------|-----------|----------|------------|------------|
| VLAN0001 | 0        | 0         | 0        | 2          | 2          |
| VLAN0010 | 0        | 0         | 0        | 2          | 2          |
| VLAN0020 | 0        | 0         | 0        | 2          | 2          |
| VLAN0030 | 0        | 0         | 0        | 2          | 2          |
| 4 vlans  | 0        | 0         | 0        | 8          | 8          |



You can use the **show spanning-tree summary** command for a quick overview of the spanning-tree topologies. You can also just use the **show spanning-tree** command and you will get information on all the VLANs. As you can see my switches have created a spanning-tree topology for each VLAN.

```
SwitchC#show spanning-tree vlan 10

VLAN0010
  Spanning tree enabled protocol ieee
  Root ID    Priority      32778
            Address      000f.34ca.1000
              This bridge is the root
```

```
SwitchC#show spanning-tree vlan 20

VLAN0020
  Spanning tree enabled protocol ieee
  Root ID    Priority      32788
            Address      000f.34ca.1000
              This bridge is the root
```

```
SwitchC#show spanning-tree vlan 30

VLAN0030
  Spanning tree enabled protocol ieee
  Root ID    Priority      32798
            Address      000f.34ca.1000
              This bridge is the root
```

Some show commands reveal to us that SwitchC is the root bridge for VLAN 10, 20 and 30.

```
SwitchA(config)#spanning-tree vlan 10 priority 4096
```

Let's lower the priority on SwitchA for VLAN 10 to 4096 so it will become the root bridge.

```
SwitchA#show spanning-tree vlan 10 | include root
This bridge is the root
```

Here's a quick way to verify our configuration.

```
SwitchB(config)#spanning-tree vlan 20 priority 4096
```

```
SwitchB#show spanning-tree vlan 20 | include root
This bridge is the root
```

SwitchB is the root for VLAN 20.

```
SwitchC(config)#spanning-tree vlan 30 priority 4096
SwitchC#show spanning-tree vlan 30 | include root
This bridge is the root
```

And last but not least here is SwitchC as the root bridge for VLAN 30.

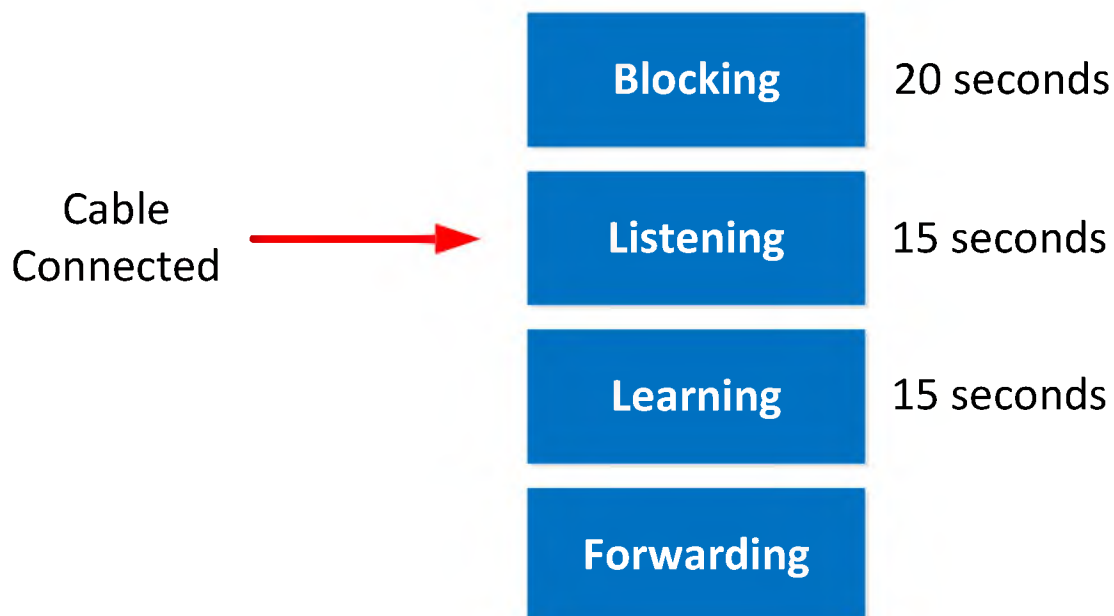


That's all there is to it! Of course different interfaces will be blocked because we have a different root bridge for each VLAN. I'm not going to try to create a picture that shows all the designated/alternate/root ports for all VLANs because we'll end up with a Picasso-style picture!



*The show spanning-tree command is excellent and gives you all the information you need to know. If you want to see in real-time what is going on you should try the debug spanning-tree command.*

A couple of pages ago I showed you the time it takes for an interface to move from the listening to the forwarding mode:



Every time you connect a cable to an interface it will move from the listening mode (15 seconds) to the learning mode (15 seconds) before it ends up in the forwarding mode. This takes 30 seconds in total.

Now let me ask you a question...how useful is it to listen for BPDU's or send them on an interface which only has a computer or server connected to it? The answer is...probably not! If there's an "end" device like a computer or server they probably aren't going to send any BPDU's to your switch and they are no part of our spanning tree puzzle.

There's a workaround so you can skip all those steps and go to forwarding mode immediately. Portfast is a Cisco proprietary solution to deal with topology changes. Interfaces with portfast enabled that come up will go to **forwarding mode immediately**. It will **skip the listening and learning state**.

It's a good idea to enable portfast on interfaces that are connected to hosts because these interfaces are likely to go up and down all the time. **Don't** enable portfast on an interface to another hub or switch.



This is how you enable it:

```
SwitchA(config)interface fa0/1
SwitchA(config-if)#spanning-tree portfast
%Warning: portfast should only be enabled on ports connected to a single
host. Connecting hubs, concentrators, switches, bridges, etc... to this
interface when portfast is enabled, can cause temporary bridging loops.
Use with CAUTION

%Portfast has been configured on FastEthernet0/1 but will only
have effect when the interface is in a non-trunking mode.
```

This is how you enable it on an interface level. Just type in **spanning-tree portfast** on the interface and you are ready to go. You will even get a warning message...(how nice!).

```
SwitchB(config)#spanning-tree portfast default
%Warning: this command enables portfast by default on all interfaces. You
should now disable portfast explicitly on switched ports leading to hubs,
switches and bridges as they may create temporary bridging loops.
```

You can also enable it globally with the **spanning-tree portfast default** command. It will enable portfast on all interfaces in **access mode**.



*Some people think that enabling portfast means that you disable spanning-tree on the interface. This is not true! Spanning-tree is still active on portfast-enabled interfaces, the only thing it does is go straight to the forwarding state. Whenever the portfast enabled interface receives a BPDU it will switch back to "normal" spanning-tree operation.*

The last thing I want to show you is how to enable rapid spanning-tree on your switches. You don't have to know the details of how rapid spanning-tree works but you need to know how to enable it:

```
SwitchA(config)#spanning-tree mode rapid-pvst
```

```
SwitchB(config)#spanning-tree mode rapid-pvst
```

```
SwitchC(config)#spanning-tree mode rapid-pvst
```

That's it...just one command will enable rapid spanning tree on our switches. The implementation of rapid spanning tree is **rapid-pvst**. We are calculating a rapid spanning tree for each VLAN.

You made it all the way to the end...if you understand everything so far...good job!

If you want to play with spanning-tree yourself you can try the following lab on your real switches:

<http://gns3vault.com/Switching/pvst-per-vlan-spanning-tree.html>

I also created a lab for spanning-tree you can use with GNS3, the commands are different than on a switch but if you don't have access to your switches but still want to play with spanning-tree you can try it:

<http://gns3vault.com/Switching/spanning-tree-for-ccna.html>

You have now seen all the protocols that run on our switches that you need to know for CCNA. It might be wise to re-read some of the chapters, do some more labs and let all the switching stuff "sink in" before you continue. From now on we will talk about routers and their protocols.

## 12. Binary, Subnetting and Summarization.

Before we continue looking at routing and routing protocols I want to go back to IP addresses and dive deeper, this stuff is important when you want to understand routing!

Before we start calculating subnets and talk about IP addressing, let's first check out some basics of binary calculations. We are all used to work with decimal numbers where we count from 1 till 10. This is easy because we have 10 fingers so we don't have to count off the top of our head.

In the binary system, we only work with 0 or 1.

0 = Off

1 = On

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

The bit on the far left side is called the most significant bit (MSB) because this bit has the highest value. The bit on the far right side is called the least significant bit (LSB) because this one has the lowest value.

So how do we convert decimal numbers into binary? Let me show you an example:

If we want the decimal number "0" in binary this means we leave all the bits "off".

| Bits     | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|----------|-----|----|----|----|---|---|---|---|
| <b>0</b> | 0   | 0  | 0  | 0  | 0 | 0 | 0 | 0 |

Let's take the number 178 and turn it into binary, just start at the left and see which bits "fit in" to make this number.  $128 + 32 + 16 + 2 = 178$ .

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>178</b> | 1   | 0  | 1  | 1  | 0 | 0 | 1 | 0 |

Just one more! Let's turn 255 into binary.  $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>255</b> | 1   | 1  | 1  | 1  | 1 | 1 | 1 | 1 |

As you can see 255 is the highest decimal number you can create when you have 8 bits to play with.



*As you can see, whenever you add a bit, the decimal value doubles. For example: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 and so on. This is called the "powers of 2".*

Before we start calculating subnets, the first thing we need to do is take a look at what subnets and IP addresses are.



*This is a good moment to create your own "cheat sheet". Take a piece of paper and write down the 8 bits for yourself.*

An IP address is a numeric value that you configure on every device in a network, think about computers, laptops, servers but also networking equipment like routers, firewalls and switches. The IP address identifies every device with a “unique” number. Devices within the same IP subnet are able to communicate without using a router.

Let’s take a look at some of the terminology you might encounter when we talk about IP addresses:

#### IP Terminology:

|                          |   |
|--------------------------|---|
| <b>Bit(s)</b>            | A bit has 2 possible values, 1 or 0. (on or off)  |
| <b>Byte</b>              | A byte is 8 bits.   |
| <b>Octet</b>             | An octet is just like a byte 8 bits, you often see byte or octet both being used.   |
| <b>Nibble</b>            | A nibble is 4 bits, we’ll talk about this more in the Hexadecimal chapter.  |
| <b>Network address</b>   | When we talk about routing, the network address is important. Routers use the network address to send IP packets to the right destination. 192.168.1.0 with subnet mask 255.255.255.0 is an example of a network address. |
| <b>Subnet</b>            | A subnet is a network that you split up in multiple smaller subnetworks.  |
| <b>Broadcast address</b> | The broadcast address is being used by applications and computers to send information to all devices within a subnet, 192.168.1.255 with subnet mask 255.255.255.0 is an example of a broadcast address.                  |

IP addresses are 32 bits, divided in 4 “blocks” also known as 4 bytes or octets. Every byte has 8 bits.  $4 \times 8 = 32$  bits.

There are many ways to write down an IP address:

|                     |                                   |
|---------------------|-----------------------------------|
| <b>Decimal:</b>     | 192.168.1.1                       |
| <b>Binary:</b>      | 11000000.10101000.0000001.0000001 |
| <b>Hexadecimal:</b> | C0.A8.01.01                       |

Decimal is what we are used to work with, as this is the way you normally configure an IP address in operating systems like Microsoft Windows, Linux or most networking equipment. Hexadecimal you won’t see often but for example you might encounter this in the windows registry.

IP addresses are hierarchical unlike non-hierarchical addresses like MAC-addresses. This has some advantages, you can use a lot of IP addresses (with 32 bits the biggest number you can create is 4.3 billion or to be precise 4,294,967,296). The advantage of having a hierarchical model is needed for routing, imagine that every router on the planet would need to know every IP address on the planet. Routing wouldn’t be very efficient that way...

A better solution is a hierarchical model where we use “network”, “subnet” and “hosts”.

Try to compare this to phone numbers:

|                |  |
|----------------|--|
| <b>0031</b>    | This is the country code for The Netherlands |
| <b>013</b>     | This is the city code for Tilburg            |
| <b>1234567</b> | This is a single number for a customer.      |

The complete phone number is 0031-013-1234567.

IP addresses use a similar hierarchical structure.

Network addresses:

**IMPORTANT:** *every subnet has 1 network address!*

The network address is a unique identification of the network. Every device within the same subnet shares this network address in its IP address, for example:

192.168.100.1  
192.168.100.2  
192.168.100.3

192.168.100. is the network address and .1, .2 and .3 are host addresses. The IP address will tell you in what subnet they are located. The network address has to be the same for all the hosts, the host part has to be unique. When the Internet was invented they created different "classes" of networks each with a different size. At this moment there are 3 classes that are important to us:

|                 | 8 bits    | 8 bits  | 8 bits  | 8 bits |
|-----------------|-----------|---------|---------|--------|
| <b>Class A:</b> | Network   | Host    | Host    | Host   |
| <b>Class B:</b> | Network   | Network | Host    | Host   |
| <b>Class C:</b> | Network   | Network | Network | Host   |
| <b>Class D:</b> | Multicast |         |         |        |
| <b>Class E:</b> | Research  |         |         |        |

Class A:

Back to our network addresses, let's take a look at Class A. The first bit always has to be a 0. This leaves us 7 bits to "play" with. The lowest value you can create by changing all bits to "0" is 0. By changing all 7 bits to "1" you get 127.

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>0</b>   | 0   | 0  | 0  | 0  | 0 | 0 | 0 | 0 |
| <b>127</b> | 0   | 1  | 1  | 1  | 1 | 1 | 1 | 1 |

$64 + 32 + 16 + 8 + 4 + 2 + 1 = 127$ .

As you can see the Class A range is between 0. and 127.

### Class B:

For a class B network the first bit has to be a 1. The second bit has to be a 0.

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>128</b> | 1   | 0  | 0  | 0  | 0 | 0 | 0 | 0 |
| <b>191</b> | 1   | 0  | 1  | 1  | 1 | 1 | 1 | 1 |

$$128 + 32 + 16 + 8 + 4 + 2 + 1 = 191$$

As you can see class B networks always start with 128. and the last network is 191.

### Class C:

For a class C network the first bit has to be a 1, the second bit a 1 and the third a 0.

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>192</b> | 1   | 1  | 0  | 0  | 0 | 0 | 0 | 0 |
| <b>223</b> | 1   | 1  | 0  | 1  | 1 | 1 | 1 | 1 |

$$128 + 64 = 192$$

$$128 + 64 + 16 + 8 + 4 + 2 + 1 = 223$$

As you can see Class B networks start at 192. and the last network is 223.

### Class D and E:

There is also a class D for multicast traffic which starts at 224. and ends at 239. Class E is for "research usage". We are not going to use these classes for our binary calculations.

### **Class A Addressing:**

A class A network has 1 byte reserved for the network address which means the other 3 bytes are left for hosts. This means we have more than a couple of networks and every network can have a lot of hosts (how to determine how many hosts each network has we will see later!).

| Byte<br>Network | Byte<br>Hosts | Byte<br>Hosts | Byte<br>Hosts |
|-----------------|---------------|---------------|---------------|
|-----------------|---------------|---------------|---------------|

If we look at the IP address 53.21.43.63 then "53" is the network address and "21.43.63" is the host address, all machines on this subnet will have the "53" as network address.

| Byte<br>Network | Byte<br>Hosts | Byte<br>Hosts | Byte<br>Hosts |
|-----------------|---------------|---------------|---------------|
| <b>53.</b>      | <b>21.</b>    | <b>43.</b>    | <b>63</b>     |



**Class B Addressing:**

A class B network has 2 bytes reserved for the network address which means the other 2 bytes are left for hosts. This means we have even more networks but fewer hosts per network compared to class A.

| Byte    | Byte    | Byte  | Byte  |
|---------|---------|-------|-------|
| Network | Network | Hosts | Hosts |

For example, 172.16.100.68, the network address is 172.16. and the host address is 100.68.

| Byte    | Byte    | Byte  | Byte  |
|---------|---------|-------|-------|
| Network | Network | Hosts | Hosts |
| 172.    | 16.     | 100.  | 68    |

**Class C Addressing:**

A class C network has 3 bytes reserved for the network address which means the other byte is left for hosts. Now we have a lot of networks but only a few hosts per network.

| Byte    | Byte    | Byte    | Byte  |
|---------|---------|---------|-------|
| Network | Network | Network | Hosts |

Another example, 192.168.200.53, the network address is 192.168.200. and the host address is .53.

| Byte    | Byte    | Byte    | Byte  |
|---------|---------|---------|-------|
| Network | Network | Network | Hosts |
| 192.    | 168.    | 200.    | 53    |

Let's take a Class C network and take a good look at it, so we can play around with binary numbers.

For example: 192.168.1.0

In binary it looks like this:

|            | 192      | 168      | 1        | 0        |
|------------|----------|----------|----------|----------|
| IP address | 11000000 | 10101000 | 00000001 | 00000000 |

In the previous chapter I explained that a class C network consists of 3 bytes for the network part, and one byte for hosts:

| Byte    | Byte    | Byte    | Byte  |
|---------|---------|---------|-------|
| Network | Network | Network | Hosts |
| 192.    | 168.    | 1.      | 0     |

Now the question is...how does a network device know which part is the network-part, and which side is the host-part? Is it because it's a Class C network? Is it some secret rule that everyone just knows about?

The answer is no, we use something called a subnet mask! For this network, it would be the following subnet mask:

255.255.255.0

Now what does this subnet mask exactly do? The word "mask" might tell you that it must mean that it's hiding something...but that is not the case, and to show you the answer we have to look at some binary numbers:

| IP address (decimal)  | 192        | 168        | 1          | 0        |
|-----------------------|------------|------------|------------|----------|
| IP address (binary)   | 11000000   | 10101000   | 00000001   | 00000000 |
| Subnet mask (decimal) | <b>255</b> | <b>255</b> | <b>255</b> | <b>0</b> |
| Subnet mask (binary)  | 11111111   | 11111111   | 11111111   | 00000000 |

The subnet mask will specify which part of the IP address is the network-part and which part is the host-part. The 1 means it's the network-part, the 0 means the host-part.

To clarify this let me just take the binary numbers. The subnet mask tells you the first 24 bits are the network-address and the 8 bits that are left we can use for hosts.

|             |          |          |          |          |
|-------------|----------|----------|----------|----------|
| IP address  | 11000000 | 10101000 | 00000001 | 00000000 |
| Subnet mask | 11111111 | 11111111 | 11111111 | 00000000 |

For our 192.168.1.0 example this means 24 bits are reserved for network and 8 bits are reserved for hosts.

Let's write down those 8 host-bits:

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

What's the highest value you can create with 8 bits? Let's have a look:

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

|            | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>255</b> | 1   | 1  | 1  | 1  | 1 | 1 | 1 | 1 |

Cool! So now we know that with 8 bits the highest value we can create is 255, does this mean we can have 255 hosts in this network? The answer is no because for every network there are 2 addresses we can't use:

- 1) Network address: this is the address where all the host bits are set to 0.

| IP address | 192      | 168      | 1        | 0        |
|------------|----------|----------|----------|----------|
|            | 11000000 | 10101000 | 00000001 | 00000000 |

- 2) Broadcast address: this is the address where all the host bits are set to 1.

| IP address | 192      | 168      | 1        | 255      |
|------------|----------|----------|----------|----------|
|            | 11000000 | 10101000 | 00000001 | 11111111 |

**IMPORTANT:** *The network address has all hosts bits set to **0**!  
The broadcast address has all host bits set to **1**!*

Alright so let's take  $255 - 2 = 253$ . Does this mean we can have a maximum of 253 hosts on our network?

The answer is still no! I messed with your head because the highest value you can create with 8 bits is not 255 but 256. Why? Because you can also use a value of "0".

Does this make your head spin? Let's take a look at our 192.168.1.0 network in binary:

| IP address  | 192      | 168      | 1        | 0        |
|-------------|----------|----------|----------|----------|
|             | 11000000 | 10101000 | 00000001 | 00000000 |
| Subnet mask | 255      | 255      | 255      | 0        |
|             | 11111111 | 11111111 | 11111111 | 00000000 |

| Network | 192      | 168      | 1        | 0        |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 00000000 |

| Broadcast | 192      | 168      | 1        | 255      |
|-----------|----------|----------|----------|----------|
|           | 11000000 | 10101000 | 00000001 | 11111111 |

The network address has all host bits set 0, so in decimal this is 0.

The broadcast address has all host bits set to 1, so in decimal this is 255.

This means everything in between we can use for hosts, 1 – 254 so that's 254 valid IP addresses we can use to configure hosts!

**IMPORTANT:** *don't start counting at "1", but start counting at "0". The "0" is a valid number.*

Great! So now you have seen what a network looks like in binary, what the subnet mask does, what the network and broadcast addresses are and that we can fit in 254 hosts in this Class C network.

Now let's say I don't want to have a single network where I can fit in 254 hosts, but I want to have 2 networks? Is this possible? It sure is! Basically what we are doing is taking a Class C network and chop it in 2 pieces, and this is what we call subnetting. Let's take a look at it in binary:

| IP address  | 192      | 168      | 1        | 0        |
|-------------|----------|----------|----------|----------|
|             | 11000000 | 10101000 | 00000001 | 00000000 |
| Subnet mask | 255      | 255      | 255      | 0        |
|             | 11111111 | 11111111 | 11111111 | 00000000 |

If we want to create more subnets we need to borrow bits from the host-part. For every bit you borrow you can double the number of subnets. By borrowing 1 bit we create 2 subnets out of this single network. There are 8 host-bits so if we steal one to create more subnets this means we have only 7 bits left for hosts.

What will the new subnet mask be? Let's take a look at it in binary:

| Subnet mask | 255      | 255      | 255      | 128      |
|-------------|----------|----------|----------|----------|
|             | 11111111 | 11111111 | 11111111 | 10000000 |

The first 24 bits are the same so we only have to look at the 4<sup>th</sup> octet, let's write down those bits:

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|-----|----|----|----|---|---|---|---|
| 1   | 0  | 0  | 0  | 0 | 0 | 0 | 0 |

Calculate it back to decimal and you'll have 128. The subnet mask will be 255.255.255.128. The second question is, how "big" are these 2 subnets and how many hosts can we fit in?

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|-----|----|----|----|---|---|---|---|
| N/A | 1  | 1  | 1  | 1 | 1 | 1 | 1 |

We have 7 bits left so let's do the binary to decimal calculation:

$$64 + 32 + 16 + 8 + 4 + 2 + 1 = 127.$$

Don't forget about the 0! Because we can use the 0 the highest value we can create with 7 bits is 128.

Our original class C network has now been divided in 2 subnets with a size of 128 each. So what will the network addresses of the 2 new subnets be? Let's work this example out in binary:

Subnet #1:

By applying the new subnet mask we only have **7 host bits** to play with.

192.168.1.0  
255.255.255.128

| IP address  | 192      | 168      | 1        | 0        |
|-------------|----------|----------|----------|----------|
|             | 11000000 | 10101000 | 00000001 | 00000000 |
| Subnet mask | 255      | 255      | 255      | 128      |
|             | 11111111 | 11111111 | 11111111 | 10000000 |

Network address:

The network address has all host bits set to 0, so the network address will be: 192.168.1.0

| Network | 192      | 168      | 1        | 0        |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 00000000 |

First usable host IP address:

The first usable host IP address is the one that comes after the network address, so this will be: 192.168.1.1

| Network | 192      | 168      | 1        | 1        |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 00000001 |

Last usable host IP address:

The last IP address we can use for a host is the one before the broadcast address, so this will be: 192.168.1.126

| Network | 192      | 168      | 1        | 126      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 01111110 |

Broadcast address:

The broadcast address has all host bits set to 1 so the broadcast address we get is: 192.168.1.127

| Broadcast | 192      | 168      | 1        | 127      |
|-----------|----------|----------|----------|----------|
|           | 11000000 | 10101000 | 00000001 | 01111111 |

Subnet #2:

The first subnet ended at 192.168.1.127 so we just continue with the next subnet at 192.168.1.128:

192.168.1.128  
255.255.255.128

| IP address  | 192      | 168      | 1        | 128      |
|-------------|----------|----------|----------|----------|
|             | 11000000 | 10101000 | 00000001 | 10000000 |
| Subnet mask | 255      | 255      | 255      | 128      |
|             | 11111111 | 11111111 | 11111111 | 10000000 |

Network address:

The network address has all host bits set to 0, so the network address will be: 192.168.1.128

| Network | 192      | 168      | 1        | 128      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 10000000 |

First usable host IP address:

The first usable host IP address is the one that comes after the network address, so this will be: 192.168.1.129

| Network | 192      | 168      | 1        | 129      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 10000001 |

Last usable host IP address:

The last IP address we can use for a host is the one before the broadcast address, so this will be: 192.168.1.254

| Network | 192      | 168      | 1        | 254      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 11111110 |

Broadcast address:

The broadcast address has all host bits set to 1 so the broadcast address we get is: 192.168.1.255

| Broadcast | 192      | 168      | 1        | 255      |
|-----------|----------|----------|----------|----------|
|           | 11000000 | 10101000 | 00000001 | 11111111 |

That's it! That's the first network we just subnetted in 2 subnets and we found out what the network and broadcast addresses are, and what IP addresses we can use for hosts.

Let me show you another one. We take the same Class C 192.168.1.0 network but now we want to have 4 subnets. For every host-bit we borrow we can double the number of subnets we can create, so by borrowing 2 host bits we can create 4 subnets.



*Every "host-bit" you "borrow" doubles the number of subnets you can create.*

What will the new subnet mask be? Let's take a look at it in binary:

| Subnet mask | 255      | 255      | 255      | 192      |
|-------------|----------|----------|----------|----------|
|             | 11111111 | 11111111 | 11111111 | 11000000 |

Calculate it from binary to decimal:  $128+64 = 192$ .

The new subnet mask will be 255.255.255.192

Subnet #1:

By applying the new subnet mask we only have **6 host bits** to play with.

192.168.1.0  
255.255.255.192

| IP address  | 192      | 168      | 1        | 0        |
|-------------|----------|----------|----------|----------|
|             | 11000000 | 10101000 | 00000001 | 00000000 |
| Subnet mask | 255      | 255      | 255      | 192      |
|             | 11111111 | 11111111 | 11111111 | 11000000 |



Network address:

The network address has all host bits set to 0, so the network address will be: 192.168.1.0

| Network | 192      | 168      | 1        | 0        |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 00000000 |

First usable host IP address:

The first usable host IP address is the one that comes after the network address, so this will be: 192.168.1.1

| Network | 192      | 168      | 1        | 1        |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 00000001 |

Last usable host IP address:

The last IP address we can use for a host is the one before the broadcast address, so this will be: 192.168.1.62

| Network | 192      | 168      | 1        | 62       |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 00111110 |

Broadcast address:

The broadcast address has all host bits set to 1 so the broadcast address we get is: 192.168.1.63

| Broadcast | 192      | 168      | 1        | 63       |
|-----------|----------|----------|----------|----------|
|           | 11000000 | 10101000 | 00000001 | 00111111 |

Subnet #2:

The first subnet ended at 192.168.1.63 so we just continue with the next subnet at 192.168.1.64:

192.168.1.64  
255.255.255.192

| IP address  | 192      | 168      | 1        | 64       |
|-------------|----------|----------|----------|----------|
|             | 11000000 | 10101000 | 00000001 | 01000000 |
| Subnet mask | 255      | 255      | 255      | 192      |
|             | 11111111 | 11111111 | 11111111 | 11000000 |

Network address:

The network address has all host bits set to 0, so the network address will be: 192.168.1.64

| Network | 192      | 168      | 1        | 64       |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 01000000 |

First usable host IP address:

The first usable host IP address is the one that comes after the network address, so this will be: 192.168.1.65

| Network | 192      | 168      | 1        | 65       |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 01000001 |

Last usable host IP address:

The last IP address we can use for a host is the one before the broadcast address, so this will be: 192.168.1.126

| Network | 192      | 168      | 1        | 126      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 01111110 |

Broadcast address:

The broadcast address has all host bits set to 1 so the broadcast address we get is: 192.168.1.127

| Broadcast | 192      | 168      | 1        | 127      |
|-----------|----------|----------|----------|----------|
|           | 11000000 | 10101000 | 00000001 | 01111111 |

Subnet #3:

The second subnet ended at 192.168.1.127 so we just continue with the next subnet at 192.168.1.128:

192.168.1.128  
255.255.255.192

| IP address  | 192      | 168      | 1        | 128      |
|-------------|----------|----------|----------|----------|
|             | 11000000 | 10101000 | 00000001 | 10000000 |
| Subnet mask | 255      | 255      | 255      | 192      |
|             | 11111111 | 11111111 | 11111111 | 10000000 |

Network address:

The network address has all host bits set to 0, so the network address will be: 192.168.1.128

| Network | 192      | 168      | 1        | 128      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 10000000 |

First usable host IP address:

The first usable host IP address is the one that comes after the network address, so this will be: 192.168.1.129



| Network | 192      | 168      | 1        | 129      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 10000001 |

Last usable host IP address:

The last IP address we can use for a host is the one before the broadcast address, so this will be: 192.168.1.190

| Network | 192      | 168      | 1        | 190      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 10111110 |

Broadcast address:

The broadcast address has all host bits set to 1 so the broadcast address we get is: 192.168.1.191

| Broadcast | 192      | 168      | 1        | 191      |
|-----------|----------|----------|----------|----------|
|           | 11000000 | 10101000 | 00000001 | 10111111 |

Subnet #4:

The second subnet ended at 192.168.1.191 so we just continue with the next subnet at 192.168.1.192:

192.168.1.192  
255.255.255.192

| IP address  | 192      | 168      | 1        | 192      |
|-------------|----------|----------|----------|----------|
|             | 11000000 | 10101000 | 00000001 | 11000000 |
| Subnet mask | 255      | 255      | 255      | 192      |
|             | 11111111 | 11111111 | 11111111 | 11000000 |

Network address:

The network address has all host bits set to 0, so the network address will be: 192.168.1.192

| Network | 192      | 168      | 1        | 192      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 11000000 |

First usable host IP address:

The first usable host IP address is the one that comes after the network address, so this will be: 192.168.1.193

| Network | 192      | 168      | 1        | 193      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 11000001 |

Last usable host IP address:

The last IP address we can use for a host is the one before the broadcast address, so this will be: 192.168.1.254

| Network | 192      | 168      | 1        | 254      |
|---------|----------|----------|----------|----------|
|         | 11000000 | 10101000 | 00000001 | 11111110 |

Broadcast address:

The broadcast address has all host bits set to 1 so the broadcast address we get is: 192.168.1.255

| Broadcast | 192      | 168      | 1        | 255      |
|-----------|----------|----------|----------|----------|
|           | 11000000 | 10101000 | 00000001 | 11111111 |

There we go! We just chopped down our 192.168.1.0 class C network into 4 subnets! If you understand everything up to this point...great job!

You have probably seen enough binary numbers now, so let's work some more with decimal numbers. We can do subnetting just by working with decimal numbers.

As you have seen in the binary examples, the rule of "powers of 2" is very useful. By taking an extra bit the decimal value doubles every time:

- For every host bit you borrow the number of subnets you can create doubles.
- Every host bit left doubles the size of the subnet.

Instead of thinking/working in binary, we'll start thinking in "**blocks**".

Take this 192.168.1.0 network with subnet mask 255.255.255.0 as an example:

We know because the subnet mask is 255.255.255.0 we have 8 bits left, and with 8 bits the highest "number" we can create is 256.

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255.$$

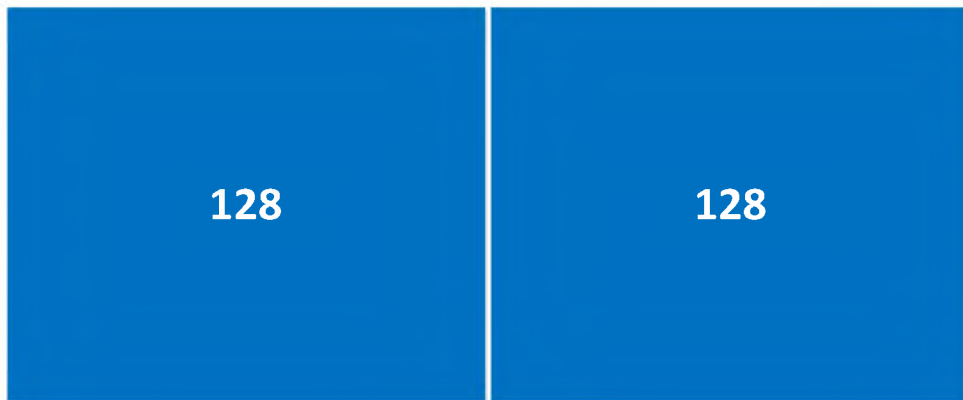
Don't forget about the 0! The 0 is being used so the highest value you can create is 256.

Visualize this as a block:



We want to subnet our 192.168.1.0 network, so we'll chop our "block" in 2 pieces.

When we chop this block in 2, this is what we get:



So now we created 2 subnets out of our Class C network.

The next questions are:

- What are the network addresses?
- What are the broadcast addresses?
- What is the subnet mask?
- What are the usable host IP addresses?

The network addresses we can write down, they are both blocks of "128", we'll start at 192.168.1.0 and the 2<sup>nd</sup> subnet will be 192.168.1.128. From .0 - .127 = "128".

Subnet #1: Network: 192.168.1.0

Subnet #2: Network: 192.168.1.128

The second question is, what are the broadcast addresses?

Well we know that the broadcast address is the last address within a subnet, so we can just write those down now we know the network addresses:

Subnet #1:    Network:     192.168.1.0  
                 Broadcast:   192.168.1.127

Subnet #2:    Network:     192.168.1.128  
                 Broadcast:   192.168.1.255

The third question, what is the subnet mask? To solve this question I'll teach you a new trick.

Take "256" minus "block size" will give you the subnet mask:

$$256 - 128 = 128.$$

The subnet mask will be 255.255.255.128



*This is a trick to remember, I would write it down on your cheat sheet.*

One question left; what are the usable host IP addresses?

- The first usable host IP address comes after the network address.
- The last usable host IP address comes before the broadcast address.
- Everything in between is a usable host IP address.

Subnet #1:    Network:     192.168.1.0  
                 First Host:    192.168.1.1  
                 Last Host:    192.168.1.126  
                 Broadcast:   192.168.1.127

Subnet #2:    Network:     192.168.1.128  
                 First Host:    192.168.1.129  
                 Last Host:    192.168.1.254  
                 Broadcast:   192.168.1.255

That was a lot faster right? We just subnetted this Class C network, calculated the network address, broadcast address and the usable host IP addresses.

Once upon a time when the IP addressing scheme was invented, the people who developed this thought it would be enough to have 3 different classes as we have seen so far, class A,B and C networks. There were only 3 subnet masks:

|         |               |            |
|---------|---------------|------------|
| Class A | 255.0.0.0     | 16,777,216 |
| Class B | 255.255.0.0   | 65,536     |
| Class C | 255.255.255.0 | 256        |

These networks are also known as "classful networks".

When the internet started growing rapidly in the beginning of the 90's this caused some problems. There were only 16,000 class B networks available which are used by a lot of medium sized companies, imagine they only needed 1000 IP addresses this would mean about 15,000 IP addresses would be wasted. A class C would be too small because you only have 256 IP addresses, and taking a couple of class C networks is no scalable option.

The solution to this problem is Classless Inter-Domain Routing, in other words we stop working with the classful networks and start working with classless networks.

Classless networks means we don't use the Class A,B or C networks anymore but are free to use any subnet mask we like, and most often you will see this in a bit-notation.

For example:

192.168.1.0  
255.255.255.0

Is the same as 192.168.1.0 /24

Instead of writing down the full subnet mask we only specify how many bits the subnet mask is.

172.16.1.0  
255.255.0.0

Is the same as: 172.16.1.0 /16

10.0.0.0  
255.0.0.0

Is the same as: 10.0.0.0 /8

One more:

172.16.1.64  
255.255.255.192

Is the same as: 172.16.64 /26

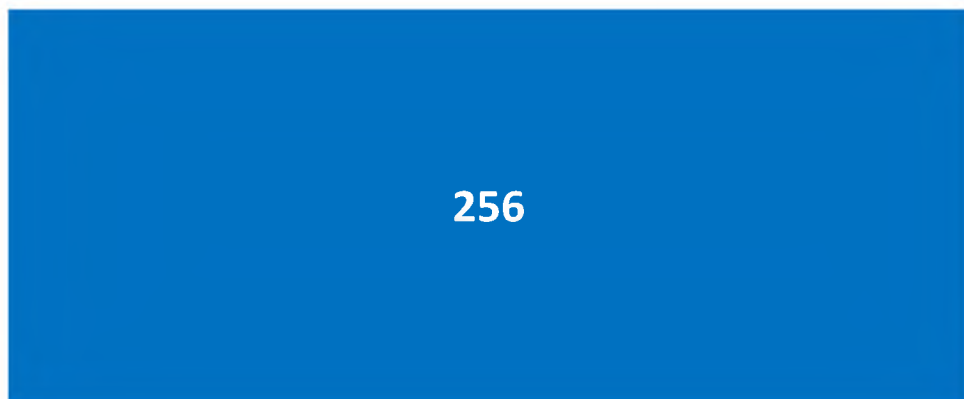
Here's a little overview with the CIDR notation and the full subnet mask:

|                 |     |
|-----------------|-----|
| 255.0.0.0       | /8  |
| 255.128.0.0     | /9  |
| 255.192.0.0     | /10 |
| 255.224.0.0     | /11 |
| 255.240.0.0     | /12 |
| 255.248.0.0     | /13 |
| 255.252.0.0     | /14 |
| 255.254.0.0     | /15 |
| 255.255.0.0     | /16 |
| 255.255.128.0   | /17 |
| 255.255.192.0   | /18 |
| 255.255.224.0   | /19 |
| 255.255.240.0   | /20 |
| 255.255.248.0   | /21 |
| 255.255.252.0   | /22 |
| 255.255.254.0   | /23 |
| 255.255.255.0   | /24 |
| 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 |

Do you see the pattern here? The numbers are the same; it's just a different octet you are playing with. For example take a look at /11 or /19. 255.224.0.0 or 255.255.224.0. Or for example /18 or /26. 255.255.192.0 and 255.255.255.192.

I believe the CIDR notation is easier then writing down the complete subnet mask, saves some time. Unfortunately on most operating systems and network equipment you still have to configure the full subnet mask instead of the CIDR notation.

Until this chapter we have been subnetting using a 'fixed size' for our blocks, so for example we took a 192.168.1.0 Class C network and divided it in 4 blocks:



Create 4 blocks out of it:



Is this a really efficient way of creating subnets? Let's say I have the following requirements:

- One subnet for 12 hosts.
- One subnet for 44 hosts.
- One subnet for 2 hosts (point-to-point links are a good example where you only need 2 IP host addresses).
- One subnet for 24 hosts.

I have 4 subnets so it's no problem, but I'm still wasting a lot of IP addresses. If we use a block of 64 for our subnet where I only need 2 IP addresses I'm throwing 62 IP addresses away.

Now you might think why we could care about this because we are using a private network address (192.168.1.0) and we have plenty of space. This is true, try to think about this on a global scale with the Internet. We don't want to throw away valuable public IP addresses.

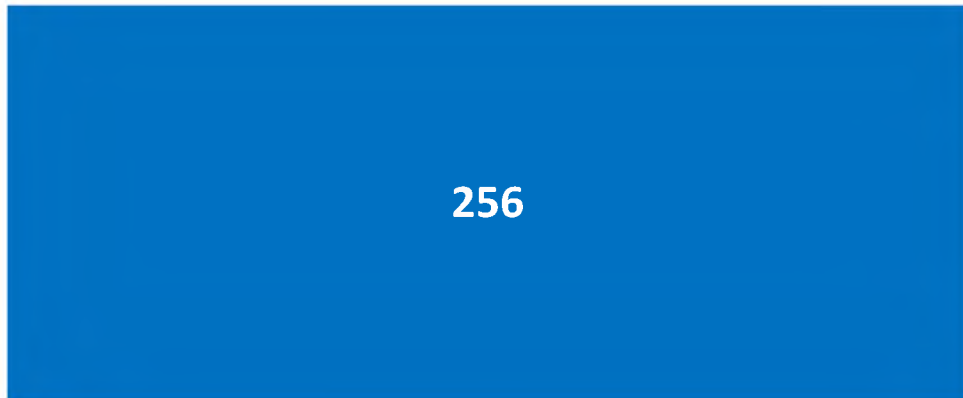
Let's say I want to subnet my 192.168.1.0 network in the most efficient way, let's take our requirements I just specified:

- One subnet for 12 hosts.
- One subnet for 44 hosts.
- One subnet for 2 hosts.
- One subnet for 24 hosts.

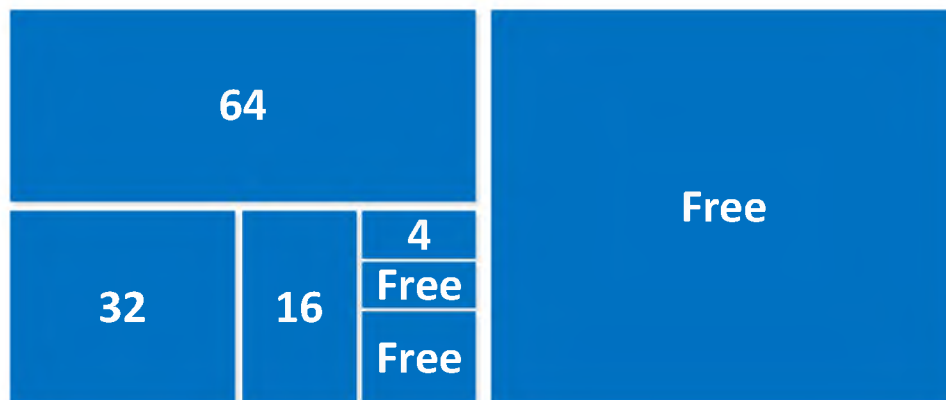
What kind of subnets would we need to fit in these hosts? Let's see:

- 12 hosts, the smallest subnet would be a block of 16.
- 44 hosts, the smallest subnet would be a block of 64.
- 2 hosts, the smallest subnet would be a block of 4.
- 24 hosts, the smallest subnet would be a block of 32.

We take our block of "256":



And divide it with the blocks we just specified:



We just saved ourselves some valuable IP addresses, now the next thing to do is answer the following questions:

- What are the network addresses?
- What are the broadcast addresses?
- What is the subnet mask?
- What are the usable host IP addresses?

Chopping a network into multiple subnets with different subnet masks is called **variable length subnet mask (VLSM)**.



*When using VLSM, start with the biggest subnet first! Otherwise you have overlapping address space. It's a good idea to remember the idea of "blocks" because you will visualize the size of the subnets this way.*



The network addresses:

Subnet #1: Network: 192.168.1.0

Subnet #2: Network: 192.168.1.64

Subnet #3: Network: 192.168.1.96

Subnet #4: Network: 192.168.1.112

Subnet #5: Network: 192.168.1.116 (this is where the Free space starts)

Let's fill in the broadcast addresses:

Subnet #1: Network: 192.168.1.0  
Broadcast: 192.168.1.63

Subnet #2: Network: 192.168.1.64  
Broadcast: 192.168.1.95

Subnet #3: Network: 192.168.1.96  
Broadcast: 192.168.1.111

Subnet #4: Network: 192.168.1.112  
Broadcast: 192.168.1.115

Because we have different subnet sizes, we need to calculate the subnet mask for each subnet.

Subnet #1: Block size: 64

$256 - 64 = 192$  so the subnet mask is 255.255.255.192

Subnet #2: Block size: 32

$256 - 32 = 224$  so the subnet mask is 255.255.255.224

Subnet #3: Block size: 16

$256 - 16 = 240$  so the subnet mask is 255.255.255.240

Subnet #4: Block size: 4

$256 - 4 = 252$  so the subnet mask is 255.255.255.252

The only thing left to do is fill in the usable host IP addresses:

Subnet #1: Network: 192.168.1.0  
First host: 192.168.1.1  
Last host: 192.168.1.62  
Broadcast: 192.168.1.63

Subnet #2: Network: 192.168.1.64  
First Host: 192.168.1.65  
Last Host: 192.168.1.94  
Broadcast: 192.168.1.95

Subnet #3: Network: 192.168.1.96  
First host: 192.168.1.97  
Last host: 192.168.1.110  
Broadcast: 192.168.1.111

Subnet #4: Network: 192.168.1.112  
First host: 192.168.1.113  
Last host: 192.168.1.114  
Broadcast: 192.168.1.115

Here we go, we just subnetted our 192.168.1.0 by using VLSM.

Now you know how to create subnets out of networks but we can also do it the *other way around*. Multiple subnets can be combined into a single network. This is what we call **summarization** or sometimes people call it **supernetting**.

Let me give you an example, here are 4 subnets:

Subnet #1: 192.168.0.0 / 24  
Subnet #2: 192.168.1.0 / 24  
Subnet #3: 192.168.2.0 / 24  
Subnet #4: 192.168.3.0 / 24

I can summarize these 4 subnets to a single entry: 192.168.0.0 / 22

How did I come up with this address? Let me show it to you in binary:

| Subnet #1 | 192      | 168      | 0        | 0        |
|-----------|----------|----------|----------|----------|
|           | 11000000 | 10101000 | 00000000 | 00000000 |
| Subnet #2 | 192      | 168      | 1        | 0        |
|           | 11000000 | 10101000 | 00000001 | 00000000 |
| Subnet #3 | 192      | 168      | 2        | 0        |
|           | 11000000 | 10101000 | 00000010 | 00000000 |
| Subnet #4 | 192      | 168      | 3        | 0        |
|           | 11000000 | 10101000 | 00000011 | 00000000 |
| Subnet #5 | 192      | 168      | 4        | 0        |
|           | 11000000 | 10101000 | 00000100 | 00000000 |

Above you see our 4 subnets in binary. The first 22 bits are exactly the same, only the last 2 bits of the third octet are different. So if I use the summary 192.168.0.0 /22 it will cover our 4 subnets. I added subnet #5 so you can see it is not covered by our summary since the 5<sup>th</sup> bit of the 3<sup>rd</sup> octet is different than the first 4 subnets.

When you convert /22 to a subnet mask you'll end up with 255.255.252.0.

When we talk about routing and routing protocols I will show you *why* we use summarization.

You don't have to create summaries by looking at binary numbers, there's an easier method. Here are the same subnets again:

Subnet #1: 192.168.0.0 / 24  
Subnet #2: 192.168.1.0 / 24  
Subnet #3: 192.168.2.0 / 24  
Subnet #4: 192.168.3.0 / 24

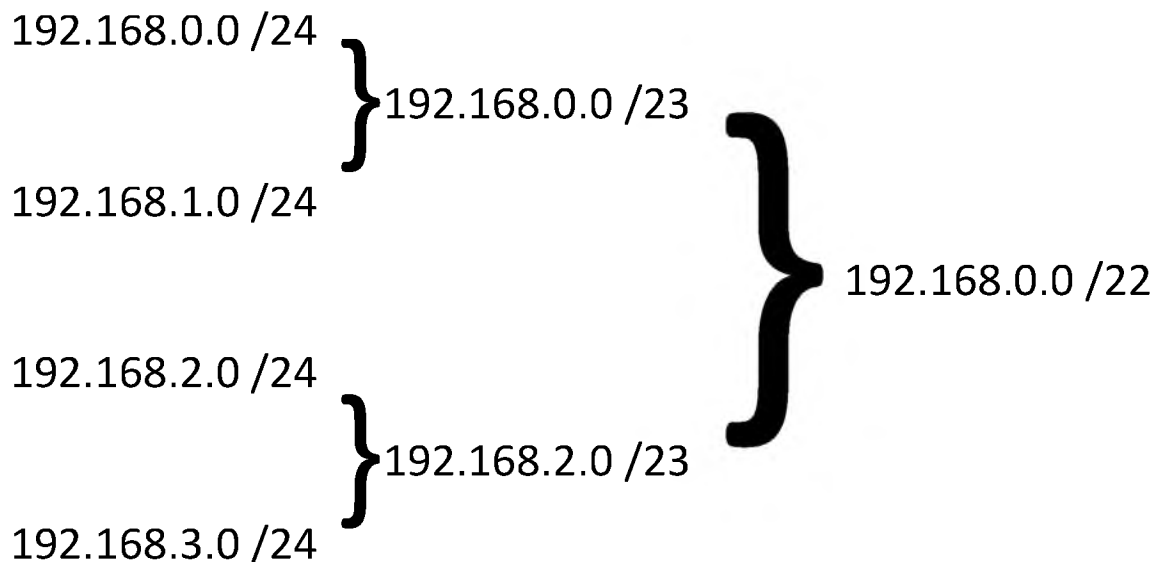
As you can see we have 4 subnets, or when we speak in 'blocks' it's a block of 4.

Take  $256 - 4 = 252$ .

The subnet mask will be 255.255.252.0

Another way to look at it is by using the CIDR notation. You know a /24 is a '256' block. Using a /23 means you have 2x 256, and a /22 means you have 4x 256.

You can also visualize it like this:



Let's do another example, let's say you want to summarize the following subnets:

Subnet #1: 10.16.0.0 /16  
 Subnet #2: 10.17.0.0 /16  
 Subnet #3: 10.18.0.0 /16  
 Subnet #4: 10.19.0.0 /16  
 Subnet #5: 10.20.0.0 /16  
 Subnet #6: 10.21.0.0 /16  
 Subnet #7: 10.22.0.0 /16  
 Subnet #8: 10.23.0.0 /16

Let's write these subnets down in binary first:

| Subnet #1 | 10       | 16       | 0        | 0        |
|-----------|----------|----------|----------|----------|
|           | 00001010 | 00010000 | 00000000 | 00000000 |
| Subnet #2 | 10       | 17       | 0        | 0        |
|           | 00001010 | 00010001 | 00000000 | 00000000 |
| Subnet #3 | 10       | 18       | 0        | 0        |
|           | 00001010 | 00010010 | 00000000 | 00000000 |
| Subnet #4 | 10       | 19       | 0        | 0        |
|           | 00001010 | 00010011 | 00000000 | 00000000 |
| Subnet #5 | 10       | 20       | 0        | 0        |
|           | 00001010 | 00010100 | 00000000 | 00000000 |
| Subnet #6 | 10       | 21       | 0        | 0        |
|           | 00001010 | 00010101 | 00000000 | 00000000 |
| Subnet #7 | 10       | 22       | 0        | 0        |
|           | 00001010 | 00010110 | 00000000 | 00000000 |
| Subnet #8 | 10       | 23       | 0        | 0        |
|           | 00001010 | 00010111 | 00000000 | 00000000 |
| Subnet #9 | 10       | 24       | 0        | 0        |
|           | 00001010 | 00011000 | 00000000 | 00000000 |

Above you can see that the first 13 bits are equal for subnet #1 up to #7.

10.16.0.0 /13 is the summary that covers all these 8 subnets. Subnet #9 isn't covered by our summary because the first 13 bits are different.

Of course you can do it much faster without looking at the binary numbers:

That's 8 subnets we want to summarize into 1 single entry, what will the CIDR notation / subnet mask be?

8 subnets that's a 'block' of 8.

$$256 - 8 = 248$$

The subnet mask will be 255.248.0.0

Or if you feel like just using the CIDR notation:

10.16.0.0 / 16 is one subnet.

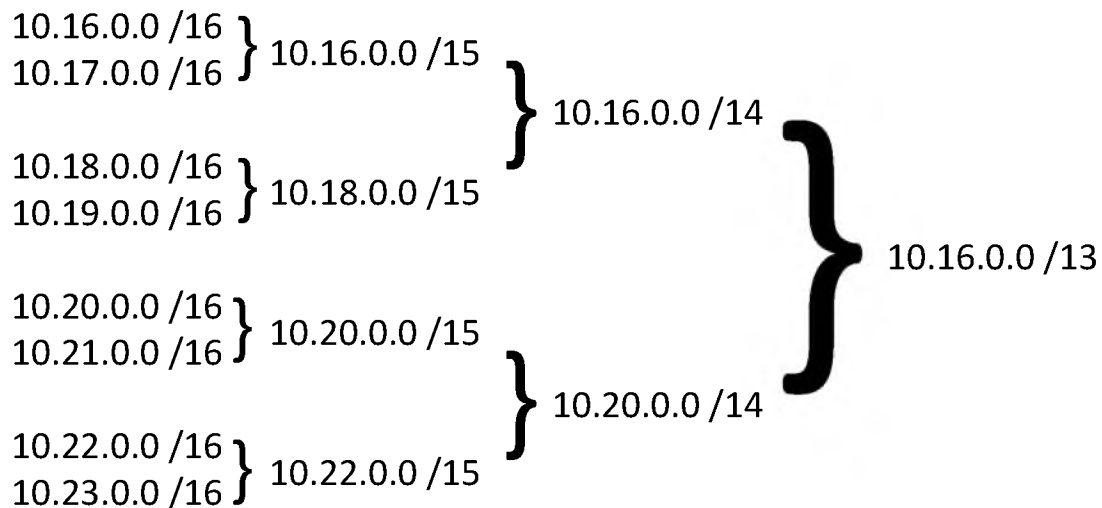
10.16.0.0 / 15 are 2 subnets.

10.16.0.0 / 14 are 4 subnets.

10.16.0.0 / 13 are 8 subnets.

So 10.60.0.0 /13 will be the answer to this question.

You can also visualize it like this:



In the upcoming chapters we will talk about routing and I will show you the advantage of using these summaries.

This is the end of the chapter. The key to binary and subnetting calculations is practice, practice and practice! You need to be lightning fast at this when you do the CCNA exam. Just write down some random IP addresses with a random CIDR notation and see if you can answer the following questions:

- What are the network addresses?
- What are the broadcast addresses?
- What is the subnet mask?
- What are the usable host IP addresses?

## 13. IP Routing

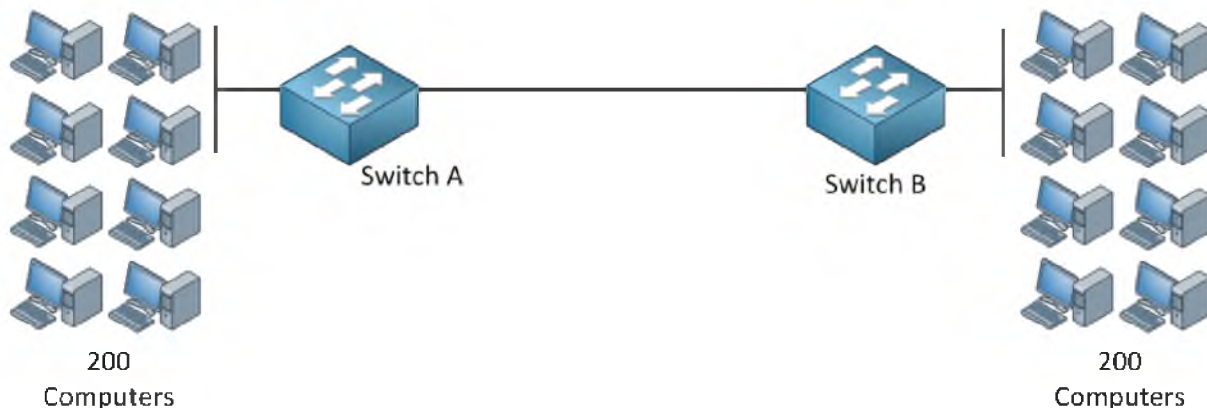
Have you seen enough switching and binary / subnetting stuff? Hopefully since we are going to continue and take a look at routers. We'll talk about the differences between routers and switches, the different routing protocols and more.

First of all...what is a router or what is routing exactly? A switch "switches" and a router "routes" but what does this exactly mean?

We have seen switches and you have learned that they "switch" based on MAC address information. The only concern for our switch is to know when an Ethernet frame enters one of its interfaces where it should send this Ethernet frame by looking at the destination MAC address. Switches make decisions based on Data Link layer information (layer 2).

Routers have a similar task but this time we are going to look at IP packets and as you might recall IP is on the Network layer (layer 3). **Routers look at the destination IP address in an IP packet and send it out the correct interface.**

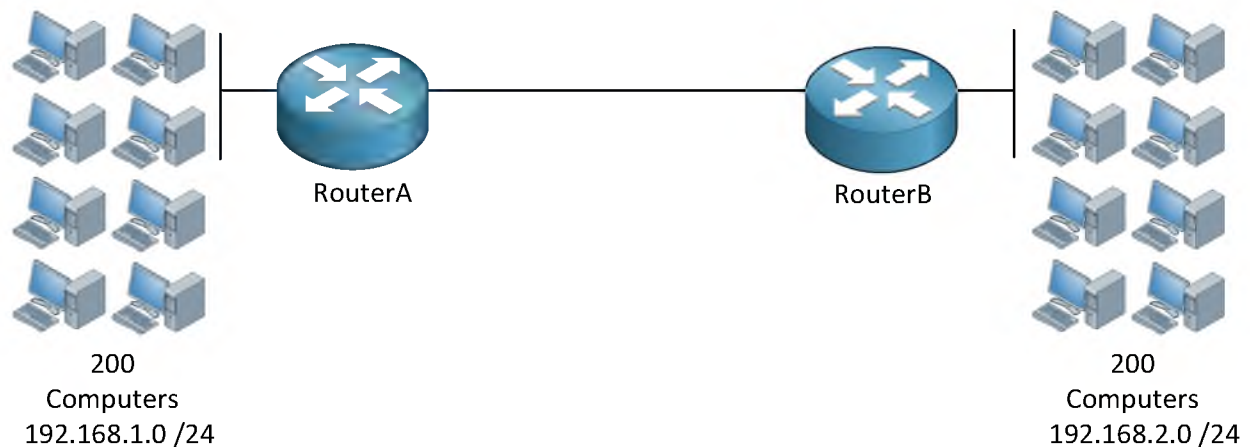
Maybe you are thinking...what is the big difference here? Why don't we use MAC addresses everywhere and switch? Why do we need to look at IP addresses and route? Both MAC addresses and IP addresses are unique per network device. Good question and I'm going to show you a picture to answer this:



We have two switches and to each switch are 200 computers connected. Now if all 400 computers want to communicate with each switch has to learn 400 MAC addresses. The need to know the MAC addresses of the computers on the left and right side.

Now think about a really large network...for example the Internet. There are millions of devices! Would it be possible to have millions of entries in your MAC-address table? For each device on the Internet? No way!

The problem with switching is that it's not scalable; we don't have any hierarchy just flat 48-bit MAC addresses. Let's look at the same example but now we are using routers.



What we have here is our 200 computers on the left are connected to router A and in the 192.168.1.0 /24 network. Router B has 200 computers behind it and the network we use over there is 192.168.2.0 /24.

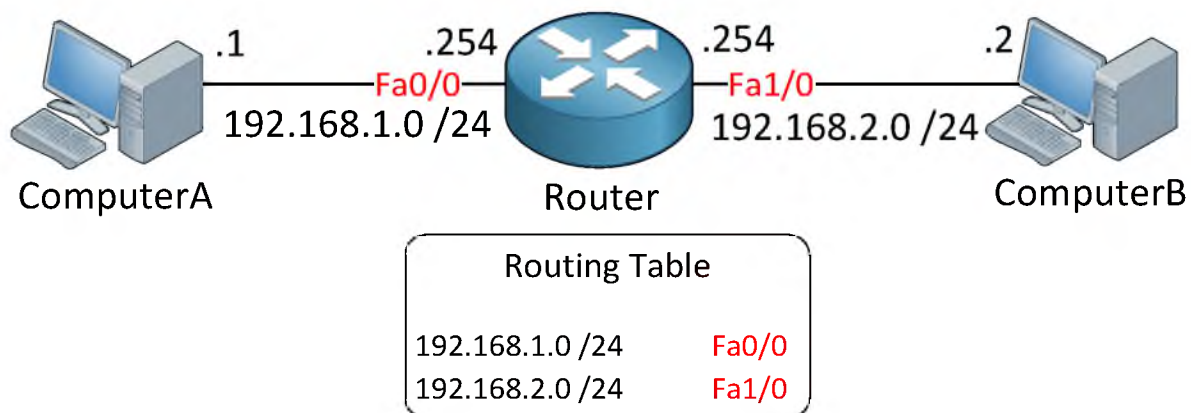
Routers "route" based on IP information, in our example Router A only has to know that network 192.168.2.0 /24 is behind Router B. Router B only needs to know that the 192.168.1.0 /24 network is behind Router A.

Are you following me here? Instead of having a MAC-address-table with 400 MAC addresses we now only need a single entry on each router for each other's networks.

Switches use mac address tables to forward Ethernet frames and routers **use a routing table to learn where to forward IP packets to.**

As soon as you take a brand new router out of the box It will build a routing table but the only information you'll find are the **directly connected interfaces.**

Let's start with a simple example:



Above we have one router and two computers:

- ComputerA has IP address 192.168.1.1 and has configured IP address 192.168.1.254 as its default gateway.
- ComputerB has IP address 192.168.2.2 and has configured IP address 192.168.2.254 as its default gateway.
- On our router we have configured IP address 192.168.1.254 on interface FastEthernet 0/0 and IP address 192.168.2.254 on interface FastEthernet 1/0.
- Since we also configured a subnet mask with the IP addresses our router knows the network addresses and will store these in its routing table.

Whenever ComputerA wants to send something to ComputerB this will happen:

1. ComputerA sends an IP packet with destination IP address 192.168.2.2.
2. ComputerA checks its own IP address and subnet mask and concludes that 192.168.2.2 is in another subnet. As a result it will forward the IP packet to its default gateway.
3. The router receives the IP packet, checks the destination IP address and scans the routing table. IP address 192.168.2.2 matches the 192.168.2.0 /24 entry and the router will forward the IP packet out if its FastEthernet 1/0 interface.
4. ComputerB receives the IP packet and life is good!

Are you following me so far? Let's configure this scenario on a real router to see what it looks like.



First I'll show you the configuration of the computers:

```
C:\Documents and Settings\ComputerA>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.1.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.254
```

```
C:\Documents and Settings\ComputerB>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.2.2
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.2.254
```

Above you see the IP addresses and the default gateways. Let's configure our router:

```
Router(config)#interface fastEthernet 0/0
Router(config-if)#no shutdown
Router(config-if)#ip address 192.168.1.254 255.255.255.0
Router(config-if)#exit
Router(config)#interface FastEthernet 1/0
Router(config-if)#no shutdown
Router(config-if)#ip address 192.168.2.254 255.255.255.0
```

I will configure the IP addresses on the interfaces, that's it. Now we can check the routing table:

```
R1#show ip route
Codes: 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
       ia - IS-IS inter area, * - candidate default, U - per-user static
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet0/0
C    192.168.2.0/24 is directly connected, FastEthernet1/0
```

As you can see the router knows about both directly connected networks.

Let's see if we can ping from ComputerA to ComputerB:

```
C:\Users\ComputerA>ping 192.168.2.2

Pinging 192.168.2.2 with 32 bytes of data:
Reply from 192.168.2.2: bytes=32 time<1ms TTL=128
Reply from 192.168.2.2: bytes=32 time<1ms TTL=128
Reply from 192.168.2.2: bytes=32 time<1ms TTL=128
Reply from 192.168.2.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.2.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Excellent the ping is working! We just successfully routed our first IP packet ☺



*Be aware that when you try to ping from one Windows computer to another that your firewall might be blocking ICMP traffic...*

If you are reaching some server on the Internet you are going through a lot of routers to reach your destination. If you want you can see through which routers your IP packets are traveling in order to reach the destination. You can do this with **tracert**. This is what it looks like if I want to reach [www.cisco.com](http://www.cisco.com) from my computer:

```
C:\Users\Computer>tracert www.cisco.com

Tracing route to e144.dscb.akamaiedge.net [95.100.128.170]
over a maximum of 30 hops:

  0  <1 ms    <1 ms    <1 ms    192.168.154.2
  1  <1 ms    <1 ms    <1 ms    192.168.81.254
  2   9 ms     7 ms     9 ms     10.224.124.1
  3   8 ms     7 ms    10 ms    tb-rc0001-cr101-irb-201.core.as9143.net [213.51.150.129]
  4  31 ms    10 ms    13 ms    asd-lc0006-cr101-ae5-0.core.as9143.net [213.51.158.18]
  5  11 ms    12 ms    11 ms    ae1.ams10.ip4.tinet.net [77.67.64.61]
  6  11 ms    14 ms    14 ms    r22.amstnl02.nl.bb.gin.ntt.net [195.69.144.36]
  7  14 ms    15 ms    11 ms    ae-2.r03.amstnl02.nl.bb.gin.ntt.net [129.250.2.211]
  8  14 ms    11 ms    11 ms    81.20.67.150
  9  12 ms    11 ms    11 ms    95.100.128.170

Trace complete.
```

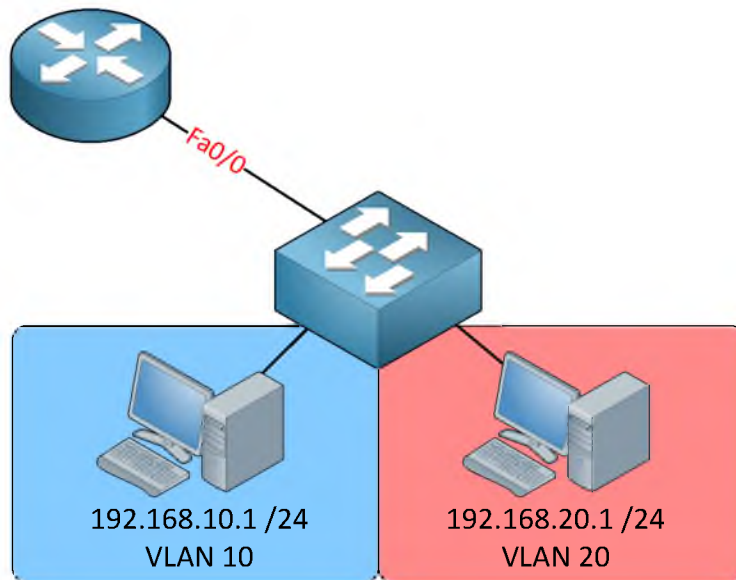
Above you can see that I travel through 10 routers in order to reach [www.cisco.com](http://www.cisco.com). You'll see the IP addresses of the routers and my computer also did a hostname lookup so you'll see the router names. Traceroute uses the ICMP protocol.



*Traceroute can also be used on Cisco routers. Just type the **traceroute** command and the IP address you want to reach.*

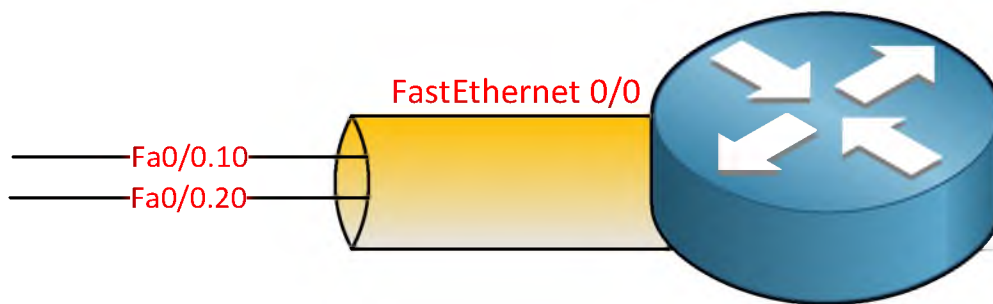
Many pages ago when we talked about VLANs I explained to you how computers in different VLANs are able to communicate with each other. We can do this with a router on a stick but back then I didn't show you the configuration because we didn't talk about routers yet.

Now you know how a router works, let's take a look at the configuration. Here's the picture:



On the switch we have VLAN 10 and VLAN 20 and there's only a single cable between the router and switch. The router needs access to both VLANs so the link between the router and switch will be a trunk!

It's possible to create **sub-interfaces** on a router. These are virtual interfaces and on each sub-interface we can configure an IP address, basically it looks like this:



You can pick any number that you like but I decided to use the VLAN numbers, one sub-interface for VLAN 10 and another for VLAN 20.

Here's what the configuration looks like on the router:

```
R1(config)#interface fastEthernet 0/0
R1(config-if)#no shutdown
R1(config-if)#exit

R1(config)#interface fastEthernet 0/0.10
R1(config-subif)#encapsulation dot1Q 10
R1(config-subif)#ip address 192.168.10.254 255.255.255.0
R1(config-subif)#exit

R1(config)#interface fastEthernet 0/0.20
R1(config-subif)#encapsulation dot1Q 20
R1(config-subif)#ip address 192.168.20.254 255.255.255.0
```

Above you can see my two sub-interfaces and the IP addresses that I assigned to them. IP address 192.168.10.254 will be the default gateway for computers in VLAN 10 and 192.168.20.254 for computers in VLAN 20.

One important command is the **encapsulation dot1Q**. There is no way for our router to know which VLAN belongs to which sub-interface so we have to use this command. Fa0/0.10 will belong to VLAN 10 and Fa0/0.20 to VLAN 20. Let's check the routing table:

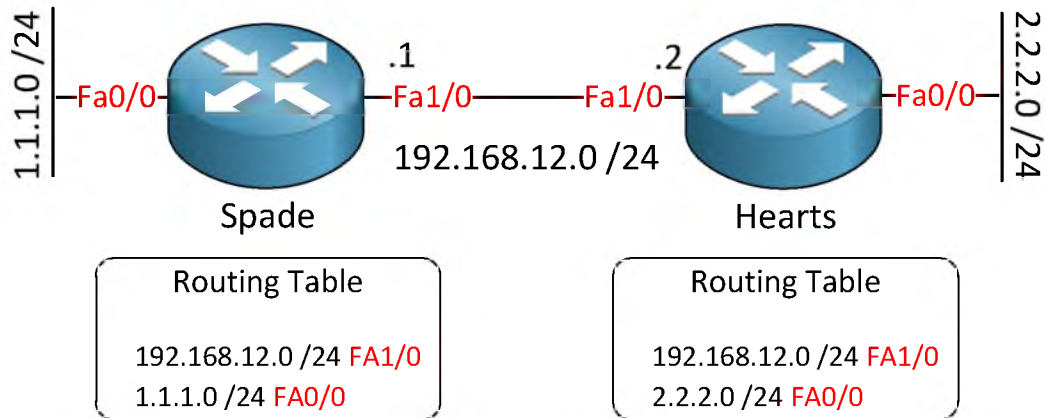
```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level
       ia - IS-IS inter area, * - candidate default, U - per-user static
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C      192.168.10.0/24 is directly connected, FastEthernet0/0.10
C      192.168.20.0/24 is directly connected, FastEthernet0/0.20
```

You can see both sub-interfaces in the routing table. This allows the router to route between the two VLANs.

Now let's move on to another routing scenario. This time we have two routers:



We have two routers called router Spade and router Hearts. If we look at their routing table this is what you will find:

Router Spade has two interfaces, FastEthernet 0/0 and FastEthernet 1/0 . Network 1.1.1.0 /24 has been configured on FastEthernet 0/0 and 192.168.12.0 /24 is configured on FastEthernet 1/0.

Router Hearts also has two interfaces, FastEthernet 0/0 and FastEthernet 1/0. Network 2.2.2.0 /24 has been configured on FastEthernet 0/0 and 192.168.12.0 /24 has been configured on FastEthernet 1/0.

One of the differences between routers and switches is that we configure an unique IP address on each interface that the router has.

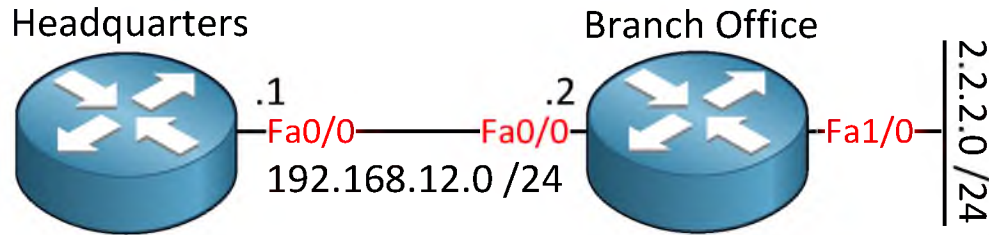
Both routers have this routing table with only their **directly connected interfaces**. You can see that they stored this information in their routing table.

Now the trick is that router Spade wants to know about the 2.2.2.0 /24 network behind router Hearts. Router Hearts wants to know about the 1.1.1.0 /24 network behind router Spade. This information has to make it into the routing table somehow.

Now the big question is...how do router Spade and router Hearts know that there is a network behind each other? There are 2 ways how they can learn this information:

- **Static Routing**
- **Dynamic Routing**

If you use static routing you will have to do **everything by yourself**. YOU tell the router where to send IP packets for a certain network, this might be entertaining but it's a lot of work. Dynamic routing means we use a routing protocol that will **exchange network information between routers**. Let's start with a configuration example of static routes on some real routers:



Look at the network in the picture above. We have a network with two sites, headquarters and a branch office.

The headquarters is connected to the Branch office. Behind the branch office is a network with the 2.2.2.0 /24 network. We want to make sure that the headquarters can reach the 2.2.2.0 /24 network.

Let me show you how we configure this network using a static route:

```
Headquarters>enable
Headquarters#configure terminal
```

First I'll go to enable mode and enter configuration mode.

```
Headquarters(config)#interface FastEthernet 0/0
Headquarters(config-if)#no shutdown
Headquarters(config-if)#ip address 192.168.12.1 255.255.255.0
```

```
Branch>enable
Branch#configure terminal
Branch(config)#interface fastEthernet0/0
Branch(config-if)#no shutdown
Branch(config-if)#ip address 192.168.12.2 255.255.255.0
Branch(config-if)#exit
Branch(config)#interface fastEthernet 1/0
Branch(config-if)#no shutdown
Branch(config-if)#ip address 2.2.2.2 255.255.255.0
```

Then I'll configure the IP addresses on the interfaces; don't forget to do a "no shutdown" on the interfaces.

Let's take a look at the routing tables of both routers:

```
Headquarters#show ip route
```

```
Codes: 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,
       ia - IS-IS inter area, * - candidate default,
       o - ODR, P - periodic downloaded static route
```

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

```
C    192.168.12.0/24 is directly connected, FastEthernet0/0
```

```
Branch#show ip route
```

```
Codes: 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,
       o - ODR, P - periodic downloaded static route
```

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

```
C    192.168.12.0/24 is directly connected, FastEthernet0/0
```

```
    2.0.0.0/24 is subnetted, 1 subnets
```

```
C    2.2.2.0 is directly connected, FastEthernet1/0
```

Use the **show ip route** command to view the routing table. This is what a router uses to make decisions where to forward IP packets to. By default a router only knows its **directly connected networks**. We configured an IP address with a subnet mask on the interface so the router also knows the network address.

- Router Headquarters knows about network 192.168.12.0/24.
- Router Branch knows about network 192.168.12.0/24 and 2.2.2.0/24.

At this moment our Headquarters router has no idea how to reach network 2.2.2.0/24 because there is no entry in the routing table. What will happen when we try to reach it?

Let's check:

```
Headquarters#ping 2.2.2.2
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
```

```
.....
```

```
Success rate is 0 percent (0/5)
```

The ping will fail. This router checks its routing table, discovers that it doesn't know how to reach network 2.2.2.0 /24 and will **drop the traffic**.

Let's use a static route to tell router Headquarters how to reach this network!

```
Headquarters(config)#ip route 2.2.2.0 255.255.255.0 192.168.12.2
```

We use the **ip route** command to create a static route.  
Let me break it down for you:

- 2.2.2.0 is the network we want to reach.
- 255.255.255.0 is the subnet mask of the network.
- 192.168.12.2 is called the **next hop IP address**. It's the IP address where we want to send traffic to. In this example that's the branch router.

I'm telling router Headquarters that it can reach network 2.2.2.0 /24 by sending traffic to IP address 192.168.12.2 (the Branch router).

Let's take another look at the routing table to see if anything has changed:

```
Headquarters#show ip route
Codes: 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 inter
area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.12.0/24 is directly connected, FastEthernet1/0
     1.0.0.0/24 is subnetted, 1 subnets
C      1.2.3.0 is directly connected, FastEthernet0/0
     2.0.0.0/24 is subnetted, 1 subnets
S      2.2.2.0 [1/0] via 192.168.12.2
```

We can now see an entry for network 2.2.2.0/24 in our routing table. Whenever router Headquarters has traffic for network 2.2.2.0 /24 it will send it to IP address 192.168.12.2 (router Branch). Let's see if our ping is now working:

```
Headquarters#ping 2.2.2.2

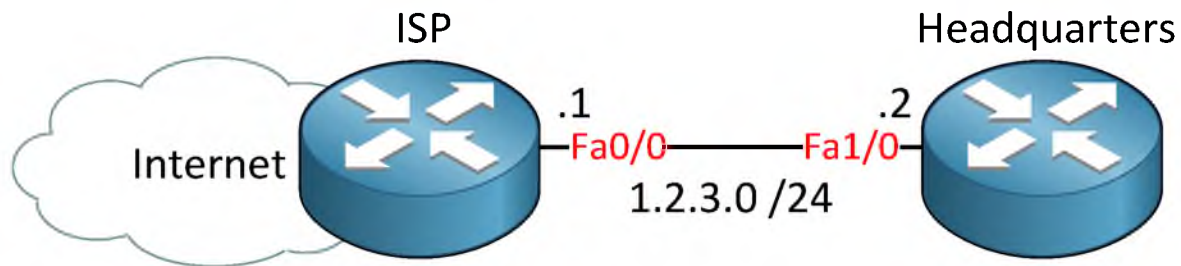
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/12 ms
```

Bingo now it's working. Router Headquarters knows how to reach network 2.2.2.0 /24 because of our static route.

Are you following me so far? Whenever an IP packet arrives at a router it will check its routing table to see if it knows about the destination network. If it does it will forward the IP packet and if it has no idea where to send traffic it will drop the IP packet.



There is another situation where a static route might be useful, let me demonstrate another network:



In the picture above our Headquarters router is connected to an ISP (Internet Service Provider). There are many networks on the Internet so do we require all of those networks on the Internet in our routing table? The answer is no because we can use a **default route**, let me show you what it is:

```
Headquarters(config)#interface fastEthernet 1/0
Headquarters(config-if)#ip address 1.2.3.2 255.255.255.0
Headquarters(config-if)#no shutdown
Headquarters(config-if)#exit
```

First we'll configure an IP address on the Fastethernet 1/0 of the headquarters router.

```
Headquarters#ping 1.2.3.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1.2.3.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/12 ms
```

It's always a good idea to check connectivity. A quick ping to the ISP router proves that we can reach the ISP.

```
Headquarters#show ip route
Codes: 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 inter
       area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C      1.2.3.0 is directly connected, FastEthernet1/0
```

Right now the Headquarters router only knows how to reach network 1.2.3.0/24 because it's directly connected.

Let's configure a default route so that we can reach the Internet:

```
Headquarters(config)#ip route 0.0.0.0 0.0.0.0 1.2.3.1
```

Let me explain this one:

- The first 0.0.0.0 is the network address; in this case it means **all networks**.
- The second 0.0.0.0 is the subnet mask; all 0s means all **subnet masks**.
- 1.2.3.1 is the next hop IP address. In this case the IP address of the ISP router.

In other words, this static route will **match all networks** and that's why we call it a default route. When our router doesn't know where to deliver IP packets to, we'll throw it over the fence towards the ISP and it will be their job to deliver it...sounds good right?

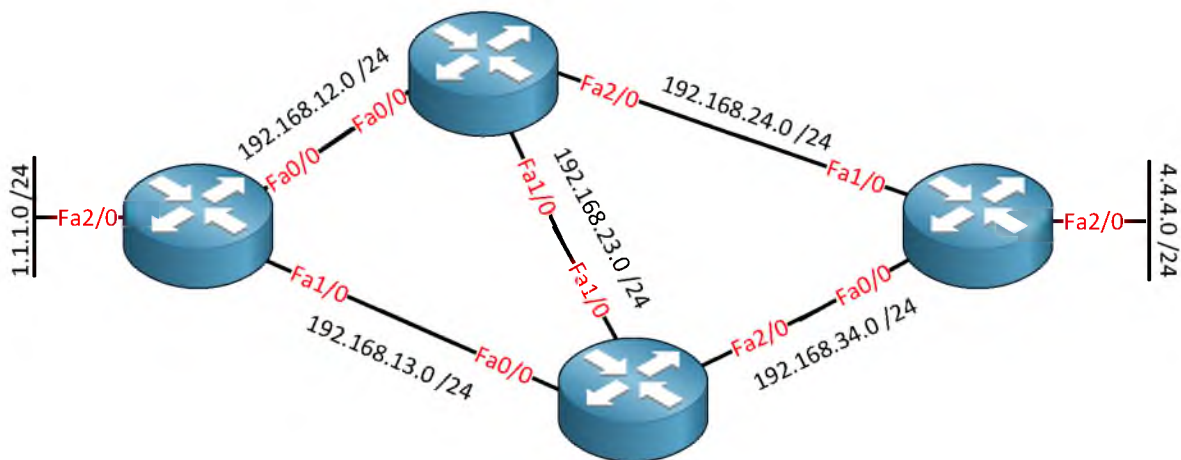
Something important to know about routers is that they always will use the **most specific match** in their routing table. Let me give you an example:

```
Router#show ip route static
      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
S       192.168.1.0/24 [1/0] via 10.2.2.2
S       192.168.1.128/25 [1/0] via 10.3.3.2
S       192.168.0.0/16 [1/0] via 10.1.1.2
```

Imagine the router above receives an IP packet with destination IP address 192.168.1.140. Will it send the IP packet towards 10.2.2.2, 10.3.3.2 or 10.1.1.2?

All 3 entries in the routing table match this destination IP address but in this case 192.168.1.128 /25 is the **most specific** entry. The IP packets will be forwarded to 10.3.3.2.

Now you know how a router uses its routing table and how to configure a static route. Are there any disadvantages to static routes? Let me show you an example:



In the picture above I have many routers and a lot of networks. If I want to configure full reachability between the routers then I have to configure a LOT of static routes to make this work and you **don't have any backups**. If a link fails you'll need to edit your static route

and send traffic another direction. The picture above would be more suitable for **dynamic routing**.

Dynamic routing is where we use a routing protocol; routing protocols are cool because they take care of our work. Routing protocols will send network information to each other and they will keep it up-to-date. If there are any changes in the network our routers will update each other to reflect this new information.

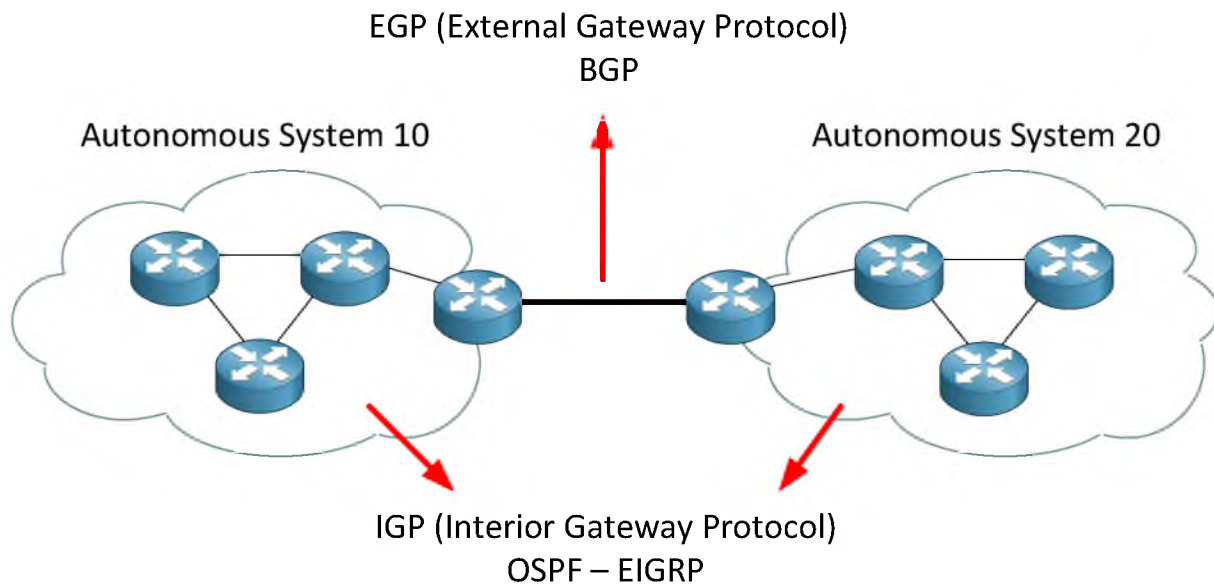
If we talk about dynamic routing there are 2 different terms that sound similar but are completely different:

- Routing protocols
- Routed protocols

A **routing protocol** is used between routers to exchange routing information and build the routing table.

**Routed protocols** are the protocols that we are routing...for example IPv4 or IPv6. In the past we used to have some other funky stuff like IPX or AppleTalk.

We have different types of routing protocols; let me show you a picture:



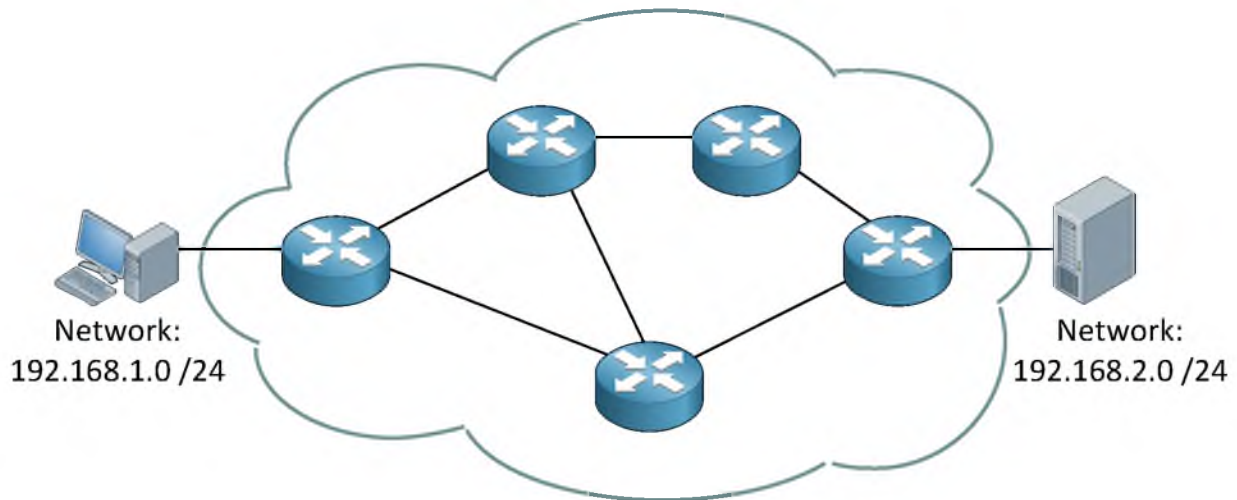
Let's break this one down: An **autonomous system** is a collection of routers/networks that belongs to a single administrative domain. Now that sounds a bit weird but in plain English what I'm trying to say is that this is the network you are responsible for. Your company network could be an autonomous system. Your internet service provider has its own network which is another autonomous system.

Within an autonomous system we run a routing protocol and we call these **interior gateway protocols** or **IGP**. OSPF and EIGRP are IGP's and we will discuss those 2 routing protocols in detail in the upcoming chapters.

Between autonomous systems we also run a routing protocol but we call these **external gateway protocols** or **EGP**. You need to think on an internet scale here, the whole internet is a bunch of autonomous systems connected to each other and in between we run an **EGP**. There is only one routing protocol we use on the internet which is called **BGP (Border Gateway Protocol)**.

You can forget about BGP for now, CCNA is only about interior gateway protocols and we are going to take a close look at OSPF and EIGRP.

Let's focus on interior gateway protocols and another picture I have for you:



You are looking at an autonomous system with a bunch of routers. On the left side there's a computer in the 192.168.1.0 /24 network and on the right side a server in the 192.168.2.0 /24 network. Our computer wants to reach the server on the right side. It's up to our routers to "route" our IP packets towards the server...the question is; which path are we going to use to get there?

This depends on the routing protocol you are using; we have 2 different routing protocols namely OSPF and EIGRP. Each of them has a different view of the world and what they think is the **shortest path** to get from source to destination. How do they determine the best path? They do so by using something called **metrics**.

OSPF and EIGRP each use **different metrics** and we'll discuss them in detail, I'm about to give you a big chunk of information. Get ready!

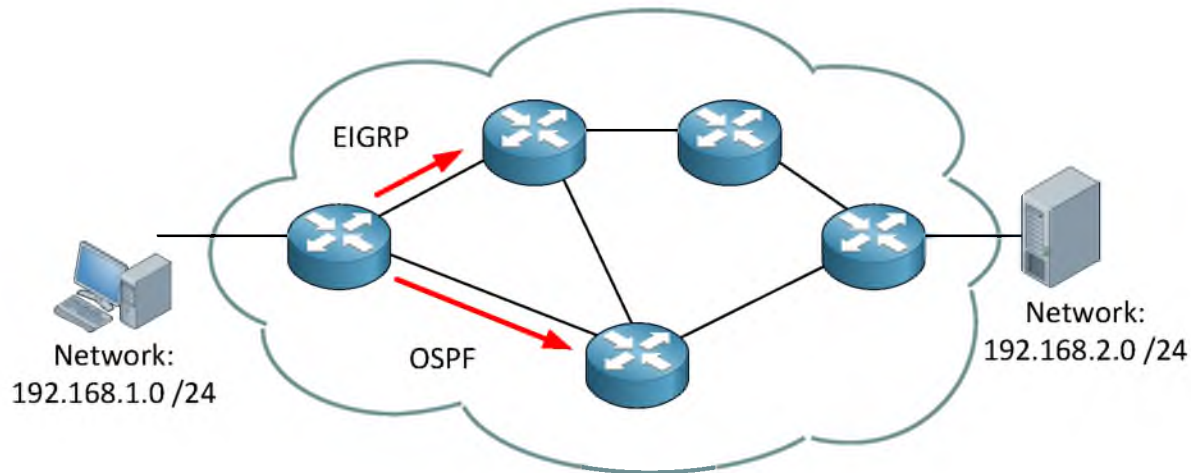
OSPF uses something called **cost** as the metric and it will actually look at the bandwidth of an interface. OSPF will prefer a 100Mbit link over a 10Mbit link.

EIGRP is Cisco's flagship routing protocol and it can use multiple metrics. It will look at the **bandwidth** and **delay** of an interface and if you want it to you can make it look at the **load** and **reliability** of an interface as well.

All three routing protocols use different metrics, for now just keep in mind that metrics are what routing protocols use to determine the shortest path to the destination. I have an entire chapter about OSPF and EIGRP for you where we will look in detail at the metrics.

Can we use multiple routing protocols at the same time on a router? Yes we can...perhaps you have to because your company bought another company and you have to merge two networks.

We do however get some funky situations like in the following example:



This time we are running two routing protocols on our network, OSPF and EIGRP. Both routing protocols are giving us routing information:

- EIGRP tells us the router should send IP packets using the path on the top.
- OSPF tells us the router should send IP packets using the path on the bottom.

What routing information are we going to use? Both? Use OSPF or EIGRP?

The answer is that when two routing protocols are giving us **information about the same destination network** we have to make a choice...you can't go left and right at the same time. We need to look at the **administrative distance** or **AD**.

Let me show you the administrative distance list:

| Administrative Distance   |     |
|---------------------------|-----|
| <b>Directly connected</b> | 0   |
| <b>Static route</b>       | 1   |
| <b>EIGRP</b>              | 90  |
| <b>OSPF</b>               | 110 |

The lower the administrative distance the better. As you can see a directly connected route has an AD of 0. This makes sense since there's nothing better than having it directly connected to your router. A static route has a very low administrative distance of 1 which also makes sense since this is something you configure manually. Sometimes you use a static route to "override" a routing protocol's decisions.

EIGRP has an administrative distance of 90 which makes sense since well, it's a Cisco routing protocol (you have to promote your own stuff right?) and OSPF has 110. In our example above we will use the information EIGRP tells us in the routing table since its AD of 90 is better (lower) than OSPF which has 110.

Are you following me? I hope so...you are half-way there to becoming a routing guru! Make sure you understand the concept of metrics and administrative distance before you continue; these are two important concepts to understand about routing! We'll talk more about metrics and the administrative distance in the OSPF and EIGRP chapters.

Interior routing protocols are divided in three different classes:

- **Distance Vector (RIP)**
- **Hybrid or Advanced Distance Vector (EIGRP)**
- **Link-State (OSPF)**

We will look at the differences of each routing protocol in their own chapter so no worries; we are going to cover everything. Just remember for now to which class they belong or write it down if you are making notes along the way.

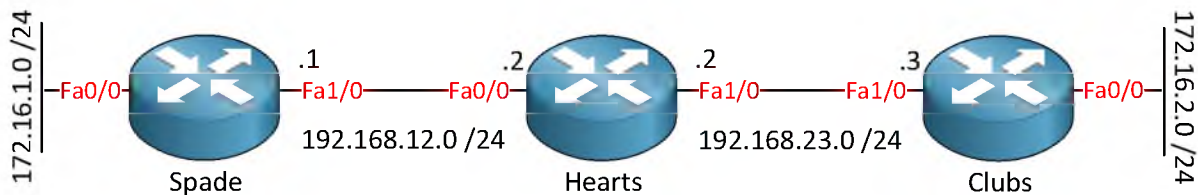
EIGRP by nature is a distance vector class but has so many extra features which make it far more superior to a normal distance vector routing protocol like RIP. This is why Cisco decided to call it a "hybrid" or "advanced distance vector".

One last thing to tell you about routing protocols:

Routing protocols can be **classful** or **classless**:

- Classful routing protocols DO NOT send the subnet mask along with their updates.
- Classless routing protocols DO send the subnet mask along with their updates.

Let's take a look at an example to emphasize this:



We have three routers and a bunch of networks. Take a close look at the networks that we have here:

172.16.1.0 /24  
172.16.2.0 /24  
192.168.12.0 /24  
192.168.23.0 /24

Remember our class A,B and C IP addresses in the beginning of this book?

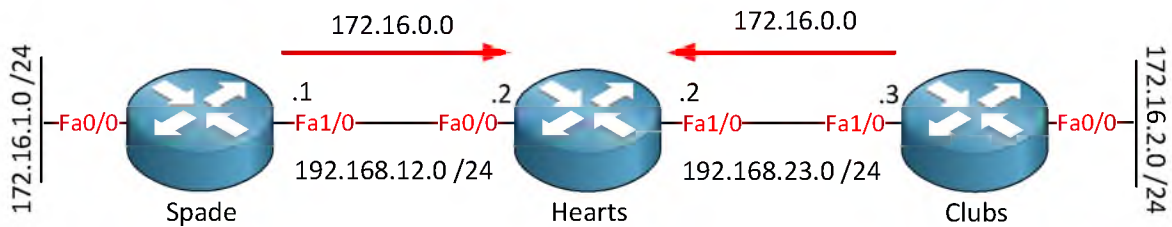
- 172.16.1.0 and 172.16.2.0 fall within the class B range.
- 192.168.12.0 and 192.168.23.0 fall within the class C range.

What subnet mask do class B and class C networks have by default?

- Class B: 255.255.0.0
- Class C: 255.255.255.0

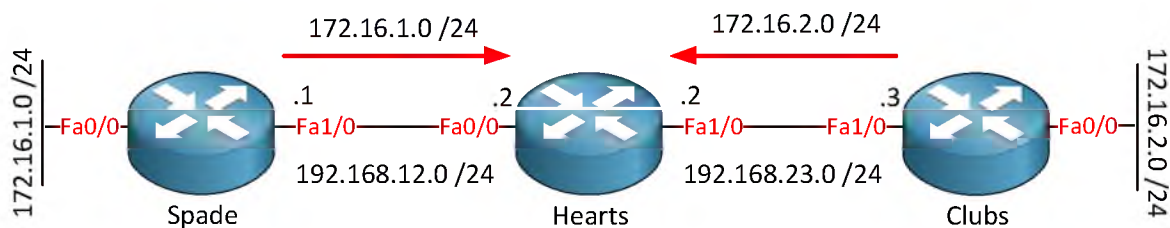


A **classful routing protocol** will not send the subnet mask along with the routing update so this is what will happen:



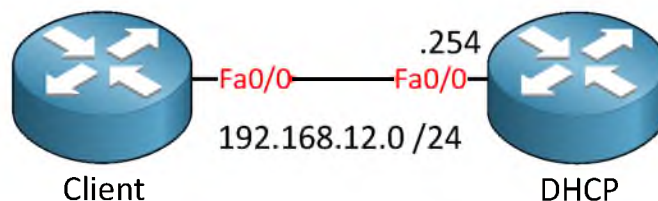
Router Spade and router Clubs don't send the subnet mask along with the routing update so it will advertise the classful network which is 172.16.0.0 in this case. So what happens with router Hearts? It thinks it can reach the 172.16.0.0 network by sending packets either left or right and if the metric is equal it will try to load-balance. Obviously this is going to cause problems.

**Classless routing protocols** advertise the subnet mask along with their updates:



As you can see router Spade is now advertising its 172.16.1.0 subnet with a subnet mask. Router Clubs is advertising its 172.16.2.0 subnet with a subnet mask as well.

Before we end this chapter about routing, there's one more thing I'd like to show you. In the beginning of this book we talked about DHCP and DHCP servers but I haven't had the chance yet to show you how to configure this. So before we move on, I quickly want to show you how to configure your Cisco router as a DHCP server. This is the topology that we'll use:



On the left side we have a router that I called "Client". I will configure its FastEthernet 0/0 interface as DHCP client so it will request an IP address.

On the right side is a router called "DHCP" that will be our DHCP server. Here's the configuration:

```
DHCP(config)#interface fastEthernet 0/0
DHCP(config-if)#no shutdown
DHCP(config-if)#ip address 192.168.12.254 255.255.255.0
DHCP(config-if)#exit
```

```
DHCP(config)#ip dhcp pool MYPOOL
DHCP(dhcp-config)#network 192.168.12.0
DHCP(dhcp-config)#default-router 192.168.12.254
DHCP(dhcp-config)#dns-server 8.8.8.8
DHCP(dhcp-config)#exit
```

With the **ip dhcp pool** command we can create a DHCP pool and I called mine "MYPOOL". The network command tells the router for which network we need to hand out IP addresses, optionally you can include default gateway for the clients and a DNS-server. In my example I will include 192.168.12.254 as the default-gateway and the DNS server is 8.8.8.8 (that's Google DNS).

Optionally you can exclude certain IP addresses from being handed out to DHCP clients with the following command:

```
DHCP(config)#ip dhcp excluded-address 192.168.12.10 192.168.12.20
```

Use ip dhcp excluded-address to configure a range of IP addresses that you don't want to assign to clients, in my case I won't hand out 192.168.12.10 – 20.



*Be careful! The ip dhcp excluded-address command is a **global** command, it would have made more sense if you could configure this under the DHCP pool...*

Let's see if our DHCP client receives an IP address from the DHCP server:

```
Client(config)#interface fastEthernet 0/0
Client(config-if)#ip address dhcp
Client(config-if)#no shutdown
```

We tell the router to configure its IP address through DHCP and this is what you will see on its console:

```
Client#
%DHCP-6-ADDRESS_ASSIGN: Interface FastEthernet0/0 assigned DHCP address
192.168.12.1, mask 255.255.255.0, hostname Client
```

There we go, we received an IP address from the DHCP server.

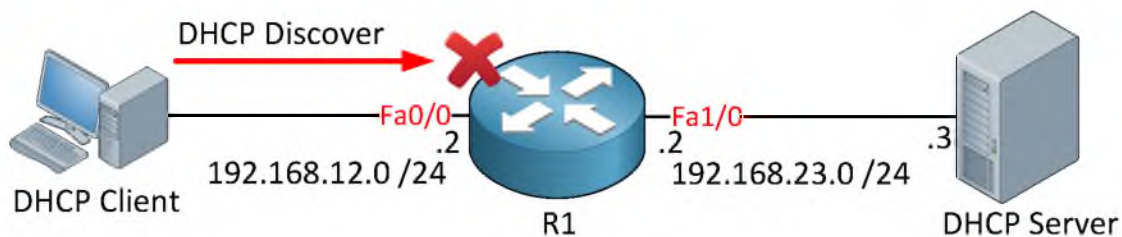


We can also verify this from the server:

```
DHCP#show ip dhcp binding
Bindings from all pools not associated with VRF:
IP address      Client-ID/      Lease expiration      Type
Hardware address/
User name
192.168.12.1     0063.6973.636f.2d63.  Mar 02 2002 12:06 AM  Automatic
                 3230.372e.3065.3734.
                 2e30.3030.302d.4661.
                 302f.30
```

Above we can see that the DHCP server has given IP address 192.168.12.1 to our client.

DHCP isn't difficult to configure, but there's one issue you might face, let me show you something:



Above we have a DHCP client on the left side, connected to R1. The DHCP server is on the right side of R1 and in another subnet. This is a situation that you will often encounter because most networks have their servers centralized.

When the DHCP client tries to get an IP address and sends a DHCP discover, what will R1 do?

Routers **don't forward broadcasts** so this DHCP discover will never make it to the DHCP server! There is a solution however, called **ip helper**.

```
R1(config)#interface fa0/0
R1(config-if)#ip helper-address 192.168.23.3
```

This little command will forward any DHCP messages and sends it with **unicast** to the DHCP server. The DHCP server will send the DHCP offer message back to the router so it can be forwarded to the client. So keep in mind, when you don't have a DHCP server on the local subnet...use ip helper to make the router forward the DHCP packets to the DHCP server!

This is the end of the chapter. You have now learned how routers use their routing table to forward IP packets and how you can use static routes to tell your routers about other networks. In the upcoming chapters we'll take a detailed look at OSPF and EIGRP.

If you want to practice a bit with static routing you can check out this GNS3 lab I created:

<http://gns3vault.com/Static-Routing/static-routing-beginners-lab.html>

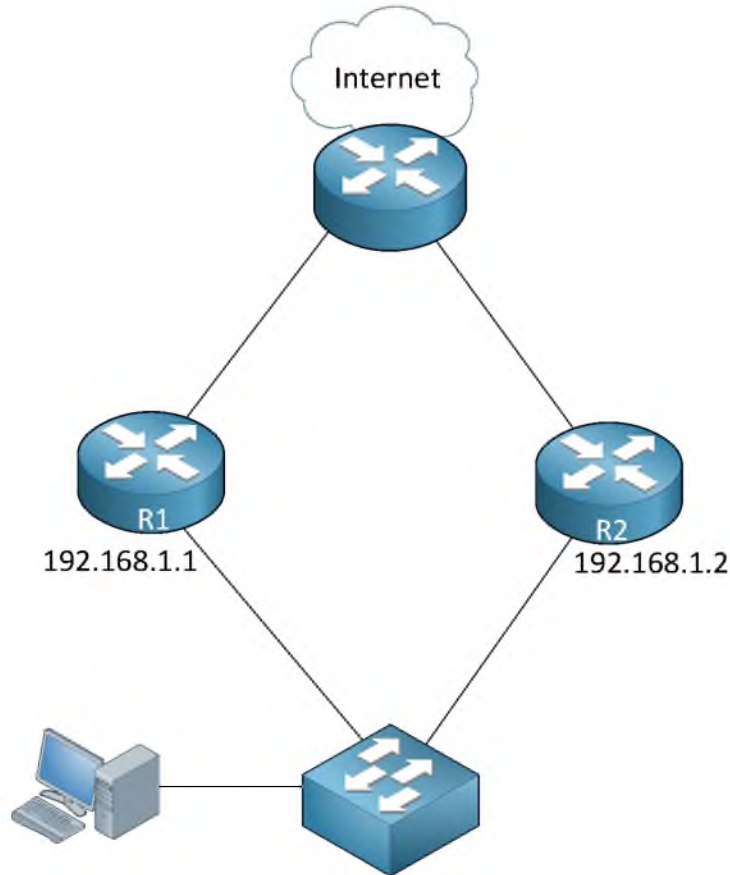
You can also take a look at the router on a stick lab here:

<http://gns3vault.com/Network-Services/router-on-a-stick.html>

More labs for routing protocols later!

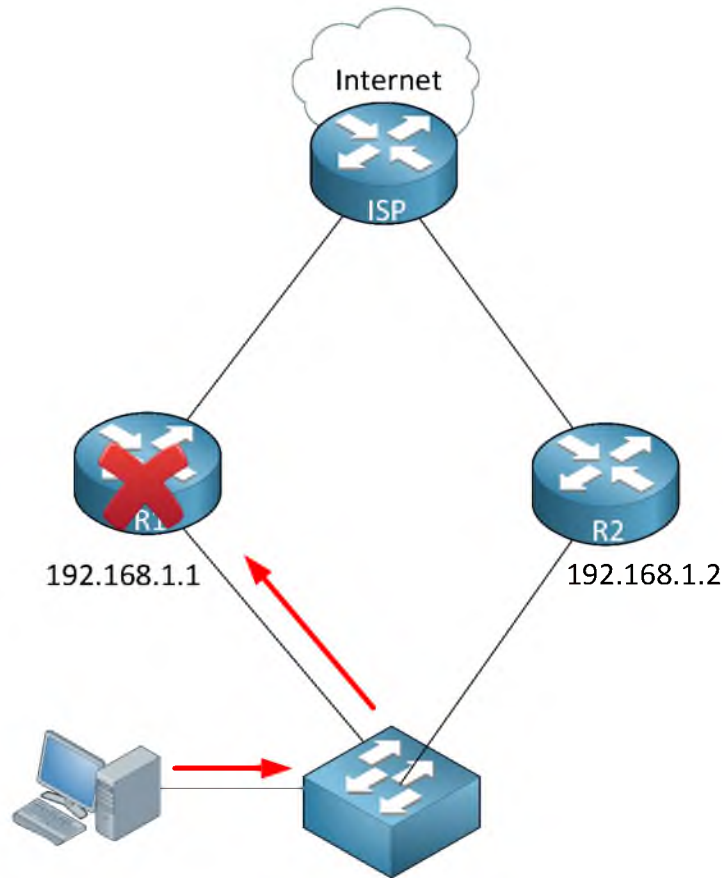
## 14. FHRP (First Hop Redundancy Protocols)

In this chapter we'll take a look at different protocols for **gateway redundancy**. So what is gateway redundancy and why do we need it? Let's start with an example!



The network in the picture above is fairly simple. I have one computer connected to a switch. In the middle you'll find two routers called R1 and R2 that both have an IP address that could be used as the default gateway for the computer. Behind R1 and R2 there's a router that is connected to the Internet.

Which gateway should we configure on the computer? R1 and R2? You can only configure a one default gateway after all...



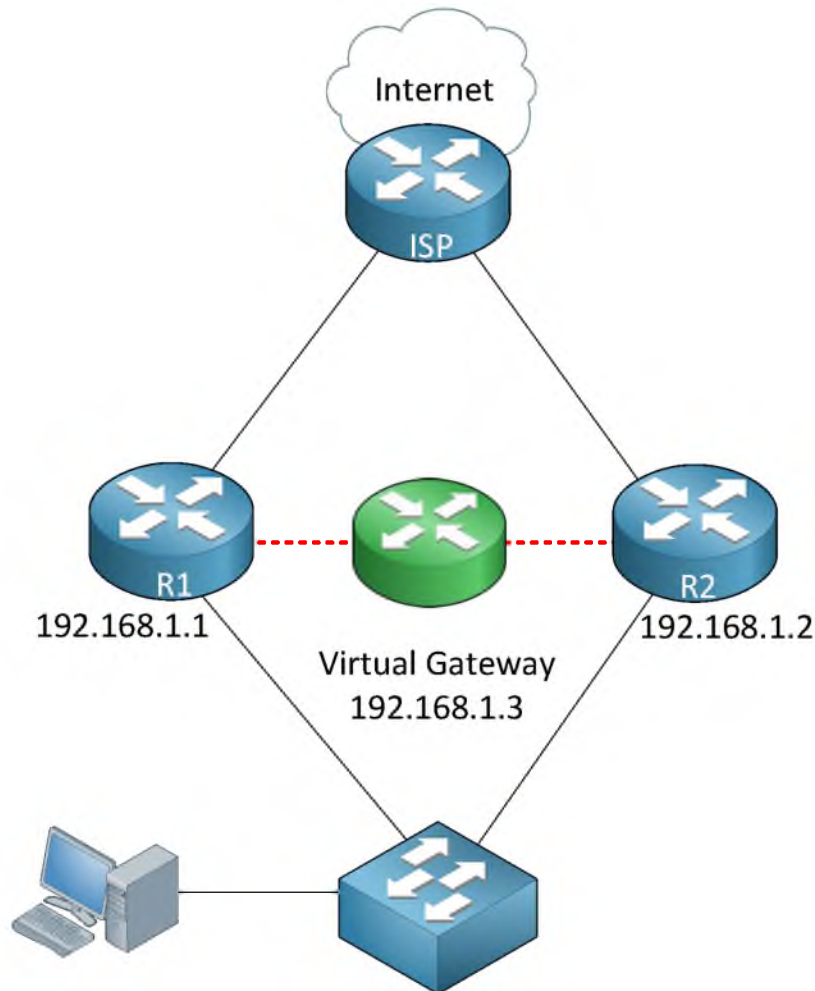
If we pick R1 and it crashes, the computer won't be able to get out of its own subnet because it only knows about **one** default gateway.



For all you Microsoft, Linux, MacOS, FreeBSD, [insert random operating system here] people. There are methods so you can configure multiple gateways and gateway failover on your operating system but since we are wearing our network engineer flip flops today we'll use a network solution to deal with gateway redundancy.

P.S. – Yes those are my Cisco flip flops from a Cisco Summer Barbecue.

P.P.S. – No I don't dare to wear them in public, they stay in the closet where they belong ;)



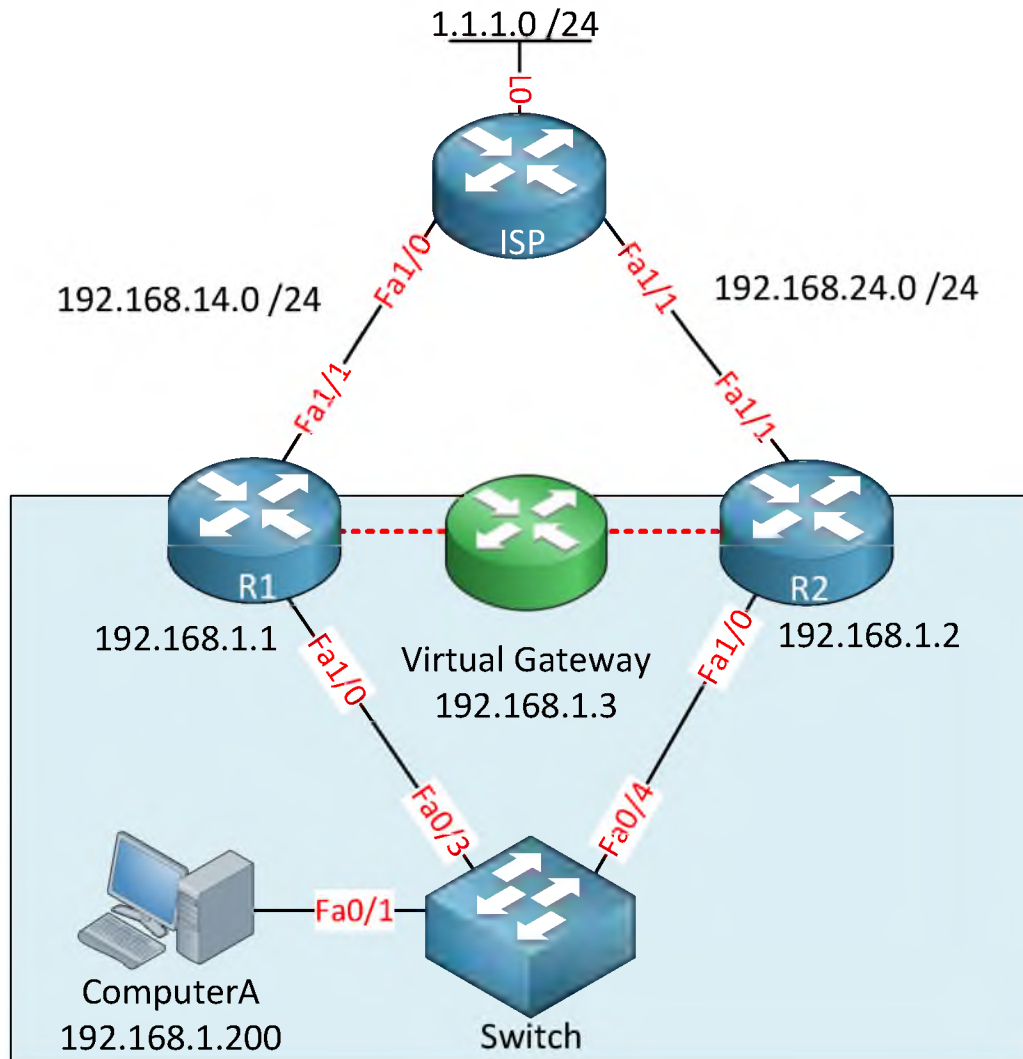
We are going to create a **virtual gateway** to solve the gateway problem. Between R1 and R2 we'll create a virtual gateway with its own IP address, in my example this is 192.168.1.3.

The computer will use 192.168.1.3 as its default gateway. One of the switches will be the active gateway and in case it fails the other one will take over.

There are three different protocols than can create a virtual gateway:

- **HSRP (Hot Standby Routing Protocol)**
- **VRRP (Virtual Router Redundancy Protocol)**
- **GLBP (Gateway Load Balancing Protocol)**

These protocols all work similar but there are a number of differences. We'll start with the configuration of HSRP which is a **Cisco proprietary** protocol. Oh by the way...HSRP, VRRP and GLBP are protocols that work perfectly in GNS3 so if you want to follow me along...boot up some emulated routers!



Here's the same topology but I have added a couple of IP addresses and interface numbers, this is what we have:

- R1, R2, the switch and ComputerA are in VLAN1 and we'll use the 192.168.1.0 /24 subnet.
- The link between R1 and the ISP router uses 192.168.14.0 /24.
- The link between R2 and the ISP router uses 192.168.24.0 /24.
- 192.168.1.3 will be the default gateway for the computer.

Let me show you the configuration!

```
R1(config)#interface fa1/0
R1(config-if)#ip address 192.168.1.1 255.255.255.0
```

```
R2(config)#interface fa1/0
R2(config-if)#ip address 192.168.1.2 255.255.255.0
```

These are the interfaces that are connected to the switch.

```
R1(config)#interface fa1/1
R1(config-if)#ip address 192.168.14.1 255.255.255.0
```

```
R2(config)#interface fa1/1
R2(config-if)#ip address 192.168.24.2 255.255.255.0
```

We will make the interfaces on R1 and R2 towards ISP layer 3 as well. Don't forget to configure an IP address.

```
ISP(config)#interface fa1/0
ISP(config-if)#ip address 192.168.14.4 255.255.255.0
```

```
ISP(config)#interface fa1/1
ISP(config-if)#ip address 192.168.24.4 255.255.255.0
```

```
ISP(config)#interface loopback 0
ISP(config-if)#ip address 1.1.1.1 255.255.255.0
```

I created a loopback interface and configured an IP address on it. When you ping the IP address on a loopback interface it will always reply.

```
R1(config)#ip route 1.1.1.0 255.255.255.0 192.168.14.4
```

```
R2(config)#ip route 1.1.1.0 255.255.255.0 192.168.24.4
```

R1 and R2 have no idea how to reach the loopback0 interface on the ISP router so I'm going to help them with a static route.

```
R1(config)#interface fa1/0
R1(config-if)#standby 1 ip 192.168.1.3
```

```
R2(config)#interface fa1/0
R2(config-if)#standby 1 ip 192.168.1.3
```

Use the **standby** command to configure HSRP. 192.168.1.3 will be the virtual gateway IP address. The "1" is the group number for HSRP.

It doesn't matter what you pick just make sure it's the same on both devices.

```
R1#
%HSRP-5-STATECHANGE: FastEthernet1/0 Grp 1 state Speak -> Standby
%HSRP-5-STATECHANGE: FastEthernet1/0 Grp 1 state Standby -> Active
```

```
R2#
%HSRP-5-STATECHANGE: FastEthernet1/0 Grp 1 state Speak -> Standby
```

Depending on which switch you configured first you'll see these messages. One of the switches will be the active gateway.

```
C:\Documents and Settings\ComputerA>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    IP Address. . . . . : 192.168.1.100
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.3
```

We can test HSRP by changing the default gateway on ComputerA. I'll set it to the virtual IP address 192.168.1.3.

```
C:\Documents and Settings\ComputerA>ping 192.168.1.3

Pinging 192.168.1.3 with 32 bytes of data:

Reply from 192.168.1.3: bytes=32 time<1ms TTL=128
Reply from 192.168.1.3: bytes=32 time<1ms TTL=128
Reply from 192.168.1.3: bytes=32 time<1ms TTL=128
Reply from 192.168.1.3: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

As you can see we can successfully reach the virtual gateway IP address.

That wasn't too bad right? Only one command and HSRP works! There are a couple of other things we can do however.

```
R1#show ip arp
```

| Protocol        | Address            | Age (min) | Hardware Addr         | Type        | Interface              |
|-----------------|--------------------|-----------|-----------------------|-------------|------------------------|
| Internet        | 192.168.14.4       | 49        | 0009.7c36.2880        | ARPA        | FastEthernet1/1        |
| Internet        | 192.168.14.1       | -         | 0011.bb0b.3642        | ARPA        | FastEthernet1/1        |
| Internet        | 192.168.1.1        | -         | 0011.bb0b.3641        | ARPA        | FastEthernet1/0        |
| <b>Internet</b> | <b>192.168.1.3</b> | <b>-</b>  | <b>0000.0c07.ac01</b> | <b>ARPA</b> | <b>FastEthernet1/0</b> |
| Internet        | 192.168.1.2        | 33        | 0019.569d.5741        | ARPA        | FastEthernet1/0        |



192.168.1.3 is our virtual IP address. What about the MAC address? It's also virtual as you can see:

**0000.0c07.ac01** is the MAC address that we have. HSRP uses the **0000.0c07.acXX** MAC address where XX is the **HSRP group number**. In my example I configured HSRP group number 1.

```
R1#show standby
FastEthernet1/0 - Group 1
  State is Active
    2 state changes, last state change 01:02:09
  Virtual IP address is 192.168.1.3
  Active virtual MAC address is 0000.0c07.ac01
    Local virtual MAC address is 0000.0c07.ac01 (v1 default)
  Hello time 3 sec, hold time 10 sec
    Next hello sent in 0.769 secs
  Preemption disabled
  Active router is local
  Standby router is 192.168.1.2, priority 100 (expires in 9.111 sec)
  Priority 100 (default 100)
  IP redundancy name is "hsrp-Fa1/0-1" (default)
```

```
R2#show standby
FastEthernet1/0 - Group 1
  State is Standby
    1 state change, last state change 01:01:51
  Virtual IP address is 192.168.1.3
  Active virtual MAC address is 0000.0c07.ac01
    Local virtual MAC address is 0000.0c07.ac01 (v1 default)
  Hello time 3 sec, hold time 10 sec
    Next hello sent in 2.959 secs
  Preemption disabled
  Active router is 192.168.1.1, priority 100 (expires in 8.608 sec)
  Standby router is local
  Priority 100 (default 100)
  IP redundancy name is "hsrp-Fa1/0-1" (default)
```

Use the **show standby** command to verify your configuration. There's a couple of interesting things here:

- We can see the virtual IP address here (192.168.1.3).
- It also shows the virtual MAC address (0000.0c07.ac01).
- You can see which router is active or in standby mode.
- The hello time is 3 seconds and the hold time is 10 seconds.
- Preemption is disabled.

The active router will **respond to ARP requests** from computers and it will be actively forwarding packets from them. It will send hello messages to the routers that are in standby mode. Routers in standby mode will listen to the hello messages, if they don't receive anything from the active router they will wait for the **hold time to expire** before taking over. The hold time is 10 seconds by default which is pretty slow; we'll see how to speed this up in a bit.

Each HSRP router will go through a number of states before it ends up as an active or standby router, this is what will happen:

| State   | Explanation  |
|---------|--|
| Initial | This is the first state when HSRP starts. You'll see this just after you configured HSRP or when the interface just got enabled. |
| Listen  | The router knows the virtual IP address and will listen for hello messages from other HSRP routers.                              |
| Speak   | The router will send hello messages and will join the election to see which router will become active or standby.                |
| Standby | The router didn't become the active router but will keep sending hello messages. If the active router fails it will take over.   |
| Active  | The router will actively forward packets from clients and sends hello messages.  |

We can see all these steps with a debug command.

```
R1(config)#interface fa1/0
R1(config-if)#shutdown
```

```
R2(config)#interface fa1/0
R2(config-if)#shutdown
```

I'm going to shut the interfaces so we can see all the states for HSRP from the beginning.

```
R1#debug standby events
HSRP Events debugging is on
```

Use the **debug standby events** command so we can take a look.

```
R1#
HSRP: Fa1/0 Grp 1 Init: a/HSRP enabled
HSRP: Fa1/0 Grp 1 Init -> Listen
HSRP: Fa1/0 Grp 1 Redundancy "hsrp-Fa1/0-1" state Init -> Backup
HSRP: Fa1/0 Grp 1 Listen: c/Active timer expired (unknown)
HSRP: Fa1/0 Grp 1 Listen -> Speak
HSRP: Fa1/0 Grp 1 Standby router is local
HSRP: Fa1/0 Grp 1 Speak -> Standby
HSRP: Fa1/0 Grp 1 Standby: c/Active timer expired (unknown)
HSRP: Fa1/0 Grp 1 Active router is local
HSRP: Fa1/0 Grp 1 Standby router is unknown, was local
HSRP: Fa1/0 Grp 1 Standby -> Active
%HSRP-5-STATECHANGE: FastEthernet1/0 Grp 1 state Standby -> Active
HSRP: Fa1/0 Grp 1 Redundancy "hsrp-Fa1/0-1" state Standby -> Active
```

I filtered out some debug information but you can clearly see the different states we go through before we end up in the active state.

```
R2#debug standby events
HSRP Events debugging is on
```

Let's enable the debug on R2 as well.

```
R2(config-if)#interface fa1/0
R2(config-if)#no shutdown
```

Get the interface up and running.

```
R2#
HSRP: Fa1/0 Grp 1 Init: a/HSRP enabled
HSRP: Fa1/0 Grp 1 Init -> Listen
HSRP: Fa1/0 Grp 1 Listen -> Speak
HSRP: Fa1/0 Grp 1 Speak -> Standby
HSRP: Fa1/0 Grp 1 Active router is local
HSRP: Fa1/0 Grp 1 Standby router is unknown, was local
HSRP: Fa1/0 Grp 1 Standby -> Active
%HSRP-5-STATECHANGE: FastEthernet1/0 Grp 1 state Standby -> Active
HSRP: Fa1/0 Grp 1 Redundancy "hsrp-Fa1/0-1" state Standby -> Active
HSRP: Fa1/0 Grp 1 Hello in 192.168.1.1 Active pri 100 vIP 192.168.1.3
HSRP: Fa1/0 Grp 1 Active: h/Hello rcvd from lower pri Active router
(100/192.168.1.1)
HSRP: Fa1/0 Grp 1 Redundancy group hsrp-Fa1/0-1 state Active -> Active
HSRP: Fa1/0 Grp 1 Redundancy group hsrp-Fa1/0-1 state Active -> Active
HSRP: Fa1/0 Grp 1 Standby router is 192.168.1.1
```

R2 will go through the same states. In the debug you can see it receives a hello message from R1 and decides that R1 has a lower priority and R2 takes over the role as the active router.

```
R1#
%HSRP-5-STATECHANGE: FastEthernet1/0 Grp 1 state Active -> Speak
%HSRP-5-STATECHANGE: FastEthernet1/0 Grp 1 state Speak -> Standby
```

We can confirm this by looking at R1. You can see that it is now in the standby state because R2 is active.

By default the switch with the **highest priority** will become the active HSRP device. If the priority is the same then the **highest IP address** will be the tie-breaker.

```
R1#show standby | include Priority
Priority 100 (default 100)
```

```
R2#show standby | include Priority
Priority 100 (default 100)
```

Both switches have a **priority of 100 by default** so the IP address is the tie-breaker. R2 has a higher IP address so it became active.

```
R1(config)#interface fa1/0
R1(config-if)#standby 1 priority 150
```

Let's say I want to make sure R1 becomes the active router. We can do this by using the **standby priority** command.

```

R1#show standby
FastEthernet1/0 - Group 1
  State is Standby
    16 state changes, last state change 00:27:52
  Virtual IP address is 192.168.1.3
  Active virtual MAC address is 0000.0c07.ac01
    Local virtual MAC address is 0000.0c07.ac01 (v1 default)
  Hello time 3 sec, hold time 10 sec
    Next hello sent in 0.039 secs
  Preemption disabled
  Active router is 192.168.1.2, priority 100 (expires in 8.960 sec)
  Standby router is local
  Priority 150 (configured 150)
  IP redundancy name is "hsrp-Fal/0-1" (default)

```

We can confirm R1 has a higher priority but R2 is still active. Once HSRP has decided which device should be active it will **stay active until it goes down**. We can overrule this if we want though...

```

R1#show standby brief
                P indicates configured to preempt.
                |
Interface    Grp Prio P State    Active        Standby        Virtual IP
Fal/0        1   150  Active local      192.168.1.2    192.168.1.3

```

```

R2#show standby brief
                P indicates configured to preempt.
                |
Interface    Grp Prio P State    Active        Standby        Virtual IP
Fal/0        1   100  Standby 192.168.1.1    local          192.168.1.3

```

The **show standby brief** command is another nice method to check your HSRP configuration.

```

R1(config)#interface fa1/0
R1(config-if)#standby 1 preempt

```

We can use **preempt** to ensure the device with the highest priority becomes active immediately.

```

R1#
%HSRP-5-STATECHANGE: FastEthernet1/0 Grp 1 state Standby -> Active

```

```

R2#
HSRP: Fal/0 Grp 1 Redundancy "hsrp-Fal/0-1" state Active -> Speak
HSRP: Fal/0 Grp 1 Speak -> Standby
HSRP: Fal/0 Grp 1 Redundancy "hsrp-Fal/0-1" state Speak -> Standby

```

There goes...R1 is now active and R2 goes to standby! We can also enable authentication if we want. You can choose between **plaintext** and **MD5**. Here's an example of plaintext:

```

R1(config)#interface fa1/0
R1(config-if)#standby 1 authentication text secret

```

```
R2(config)#interface fa1/0
R2(config-if)#standby 1 authentication text secret
```

```
R1(config)#interface fa1/0
R1(config-if)#standby 1 authentication md5 key-string md5pass
```

```
R2(config)#interface fa1/0
R2(config-if)#standby 1 authentication md5 key-string md5pass
```

Here's an example for MD5 authentication.

```
R1#show standby | include time
Hello time 3 sec, hold time 10 sec
```

By default HSRP is pretty slow. R2 is my standby router and it will wait for 10 seconds (hold time) before it will become active once R1 fails. That means we'll have 10 seconds of downtime...

```
R1(config)#interface fa1/0
R1(config-if)#standby 1 timers msec 100 msec 300
```

```
R2(config)#interface fa1/0
R2(config-if)#standby 1 timers msec 100 msec 300
```

We can speed things up by changing the timers with the **standby timers** command. We can even use millisecond values. I've set the hello time to 100 milliseconds and the hold timer to 300 milliseconds. Make sure your hold time is at **least three times** the hello timer.

```
R1(config)#interface fa1/0
R1(config-if)#standby 1 preempt
R1(config-if)#standby 1 preempt delay minimum 60
```

```
R2(config)#interface fa1/0
R2(config-if)#standby 1 preempt
R2(config-if)#standby 1 preempt delay minimum 60
```

Remember preemption that I just showed you? By default preemption will take effect immediately but it might be a good idea to use a delay. If a router or reboots it might need some time to "converge". Maybe spanning-tree isn't ready yet unblocking ports. I've changed the delay for preemption to 60 seconds, this way we make sure this device won't become active until its uplinks are up and running.

```
R1(config)#interface fa1/0
R1(config-if)#standby 1 version 2
```

```
R2(config)#interface fa1/0
R2(config-if)#standby 1 version 2
```

Depending on the router or switch model you might have the option to use HSRP version 2. You can change the version by using the **standby version** command.

|                     | HSRPv1                             | HSRPv2                              |
|---------------------|------------------------------------|-------------------------------------|
| Group numbers       | 0 – 255                            | 0 – 4095                            |
| Virtual MAC Address | 0000.0c07.acXX (XX = group number) | 0000.0c9f.fxxx (XXX = group number) |
| Multicast address   | 224.0.0.2                          | 224.0.0.102                         |

HSRP version 1 and version 2 are **not compatible** so make sure you use the same version on both devices.

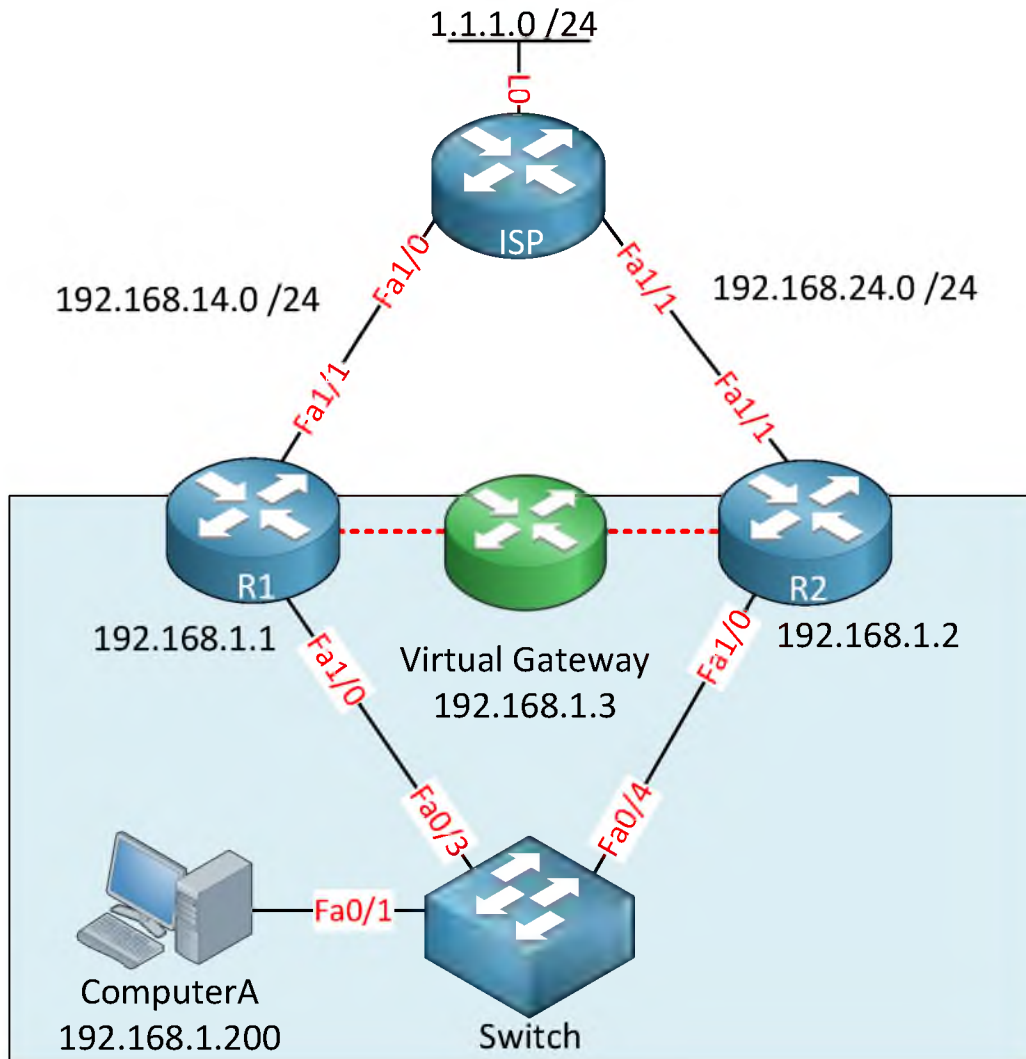
This is everything I want to show you about HSRP. We'll continue by taking a look at VRRP!

VRRP is very similar to HSRP; if you understood HSRP you'll have no trouble with VRRP which is a **standard protocol** defined by the IETF in RFC 3768. Configuration-wise it's pretty much the same but there are a couple of differences.

Let's start with an overview:

|                    | HSRP   | VRRP  |
|--------------------|--|---|
| Protocol           | Cisco proprietary                              | IETF – RFC 3768   |
| Number of groups   | 16 groups maximum                              | 255 groups maximum                                      |
| Active/Standby     | 1 active, 1 standby and multiple candidates.   | 1 active and several backups.                           |
| Virtual IP Address | Different from real IP addresses on interfaces | Can be the same as the real IP address on an interface. |
| Multicast address  | 224.0.0.2                                      | 224.0.0.18  |
| Tracking           | Interfaces or Objects                          | Objects   |
| Timers             | Hello timer 3 seconds, hold time 10 seconds.   | Hello timer 1 second, hold time 3 seconds.              |
| Authentication     | Supported                                      | Not supported in RFC 3768                               |

As you can see there are a number of differences between HSRP and VRRP. Nothing too fancy however. HSRP is a **Cisco proprietary protocol** so you can only use it between Cisco devices.



Let's use the same topology we used for HSRP but now we'll configure it for VRRP.

```
R1(config)#interface fa1/0
R1(config-if)#no standby 1 ip 192.168.1.3
```

```
R2(config)#interface fa1/0
R2(config-if)#no standby 1 ip 192.168.1.3
```

We'll start by getting rid of HSRP. This is the quick and dirty way of disabling it. It's better to remove all the "standby" commands on the interface so your config stays clean.

```
R1(config)#interface fa1/0
R1(config-if)#vrrp 1 ip 192.168.1.3
R1(config-if)#vrrp 1 priority 150
R1(config-if)#vrrp 1 authentication md5 key-string mykey
```

```
R2(config-if)#interface fa1/0
R2(config-if)#vrrp 1 ip 192.168.1.3
R2(config-if)#vrrp 1 authentication md5 key-string mykey
```

Here's an example how to configure VRRP. You can see the commands are pretty much the same but I didn't type "standby" but **vrrp**. I have changed the priority on R1 to 150 and I've enabled MD5 authentication on both switches.

```
R1#
%VRRP-6-STATECHANGE: Fa1/0 Grp 1 state Init -> Backup
%VRRP-6-STATECHANGE: Fa1/0 Grp 1 state Backup -> Master
```

```
R2#
%VRRP-6-STATECHANGE: Fa1/0 Grp 1 state Init -> Backup
%VRRP-6-STATECHANGE: Fa1/0 Grp 1 state Backup -> Master
%VRRP-6-STATECHANGE: Fa1/0 Grp 1 state Master -> Backup
```

You will see these messages pop-up in your console. VRRP uses different terminology than HSRP. R1 has the best priority and will become the **master router**. R2 will become a **standby router**.

```
R1#show vrrp
FastEthernet1/0 - Group 1
  State is Master
  Virtual IP address is 192.168.1.3
    Secondary Virtual IP address is 192.168.1.4
  Virtual MAC address is 0000.5e00.0101
  Advertisement interval is 1.000 sec
  Preemption enabled
  Priority is 150
  Authentication MD5, key-string "mykey"
  Master Router is 192.168.1.1 (local), priority is 150
  Master Advertisement interval is 1.000 sec
  Master Down interval is 3.414 sec
```

```
R2#show vrrp
FastEthernet1/0 - Group 1
  State is Backup
  Virtual IP address is 192.168.1.3
  Virtual MAC address is 0000.5e00.0101
  Advertisement interval is 1.000 sec
  Preemption enabled
  Priority is 100
  Authentication MD5, key-string "mykey"
  Master Router is 192.168.1.1, priority is 150
  Master Advertisement interval is 1.000 sec
  Master Down interval is 3.609 sec (expires in 3.065 sec)
```

Use **show vrrp** to verify your configuration.

The output looks similar to HSRP; one of the differences is that VRRP uses **another virtual MAC address**:



0000.5e00.01XX (where X = group number)

```
R1(config)#interface fa1/0
R1(config-if)#shutdown
```

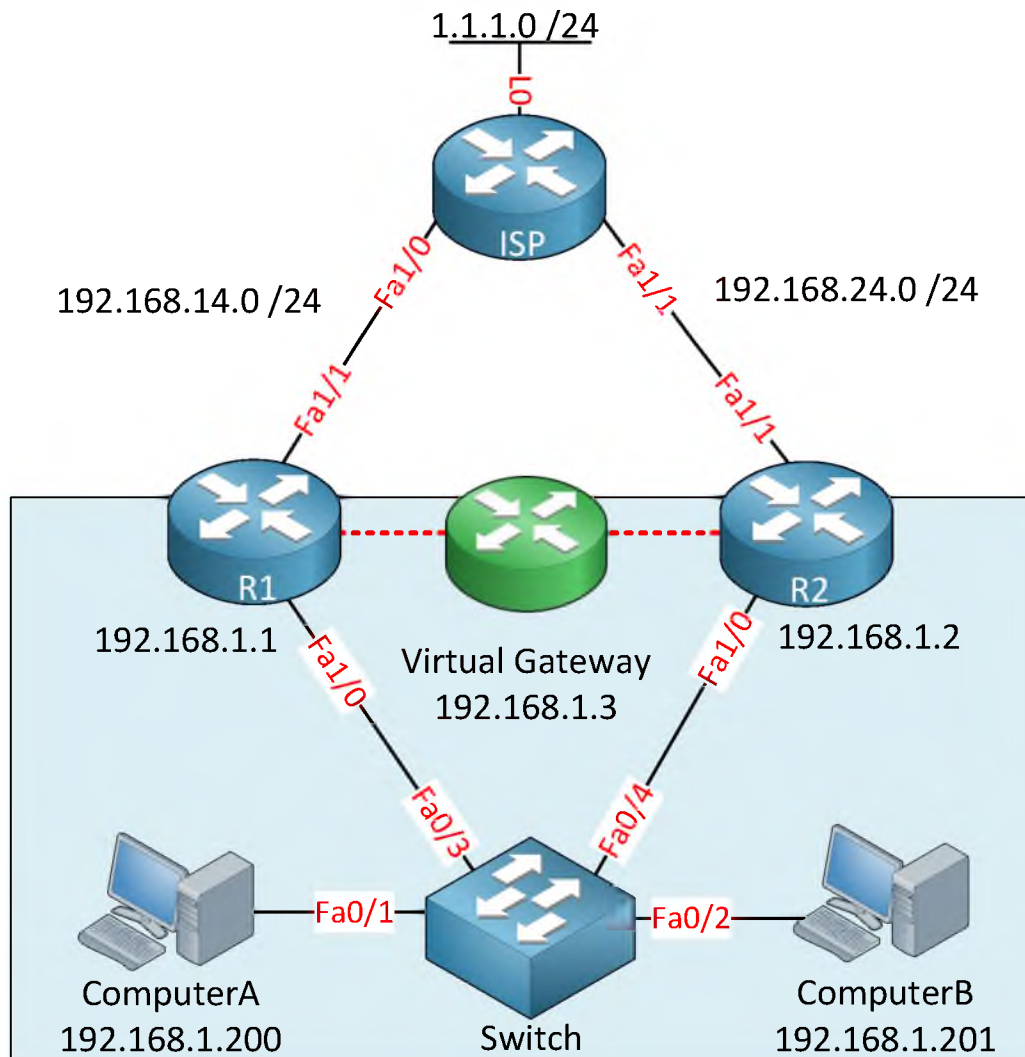
We can shut the interface on R1 so we can see that R2 will take over.

```
R1#
%VRRP-6-STATECHANGE: Fa1/0 Grp 1 state Master -> Init
```

```
R2#
%VRRP-6-STATECHANGE: Fa1/0 Grp 1 state Backup -> Master
```

Same principle...different terminology!

Is it possible to do load balancing when we use HSRP or VRRP? Both protocols elect one device to be the active/master device which will take care of forwarding all the IP packets. It would be a shame not to use R2 at all right?



In the example above I have added ComputerB and I would like it to use R2 as its gateway without losing redundancy. If we pull this off we'll have a 50/50 load share (if both computers would send the same amount of data).

```
R1(config)#interface fa1/0
R1(config-if)#standby 1 ip 192.168.1.3
R1(config-if)#standby 1 priority 150
R1(config-if)#standby 2 ip 192.168.1.4
```

```
R2(config)#interface fa1/0
R2(config-if)#standby 1 ip 192.168.1.3
R2(config-if)#standby 2 ip 192.168.1.4
R2(config-if)#standby 2 priority 150
```

Here's an example for HSRP. I created two groups so we have two virtual IP addresses:

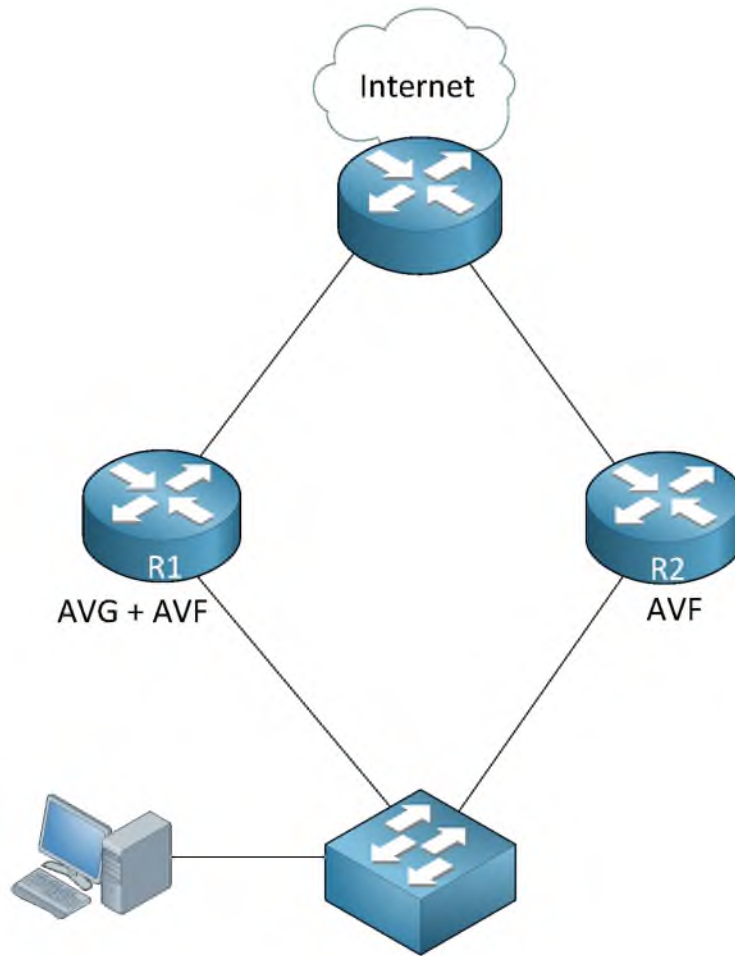
- 192.168.1.3 and 192.168.1.4 are both virtual IP addresses we can use as a gateway.
- R1 has the highest priority (150) for virtual IP address 192.168.1.3.
- R2 has the highest priority (150) for virtual IP address 192.168.1.4.
- ComputerA can use 192.168.1.3 as its default gateway.
- ComputerB can use 192.168.1.4 as its default gateway.
- We now have load sharing and R1 and R2 will be redundant for each other!

Here's the same configuration for VRRP:

```
R1(config)#interface fa1/0
R1(config-if)#vrrp 1 ip 192.168.1.3
R1(config-if)#vrrp 1 priority 150
R1(config-if)#vrrp 2 ip 192.168.1.4
```

```
R2(config-if)#interface fa1/0
R2(config-if)#vrrp 1 ip 192.168.1.3
R2(config-if)#vrrp 2 ip 192.168.1.4
R2(config-if)#vrrp 2 priority 150
```

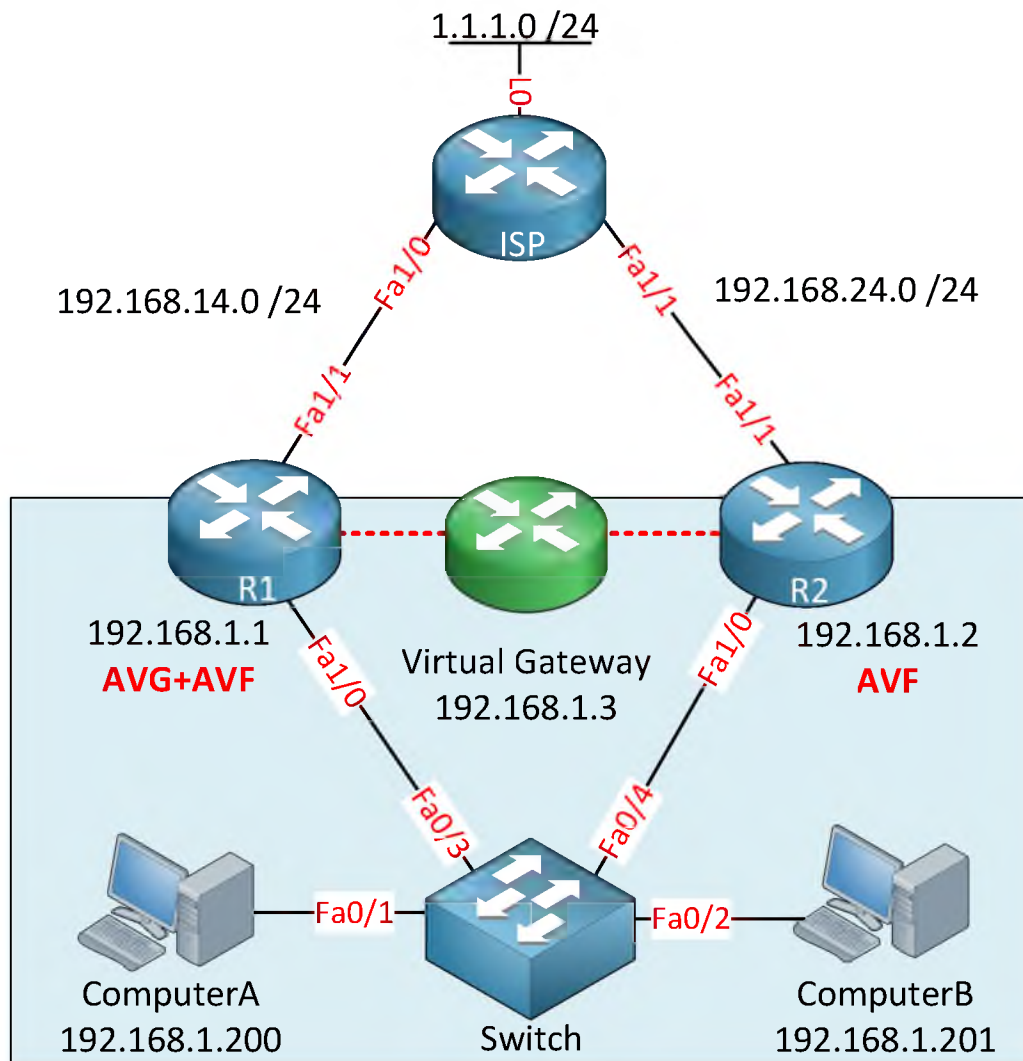
That's all I have on HSRP and VRRP. Let's take a look at the last virtual gateway protocol **GLBP**. GLBP stands for **Gateway Load Balancing Protocol** and one of the key differences is that it can do load balancing without the group configuration that HSRP/VRRP use (what's in a name right?).



All devices running GLBP will elect an **AVG (Active Virtual Gateway)**. There will be only one AVG for a single group running GLBP but other devices can take over this role if the AVG fails. The role of the AVG is to assign a **virtual MAC address** to all other devices running GLBP. All devices will become an **AVF (Active Virtual Forwarder)** including the AVG. Whenever a computer sends an ARP Request the AVG will respond with one of the virtual MAC addresses of the available AVFs. Because of this mechanism all devices running GLBP will be used to forward IP packets.

There are multiple methods for load balancing:

- **Round-robin:** the AVG will hand out the virtual MAC address of AVF1, then AVF2, AVF3 and gets back to AVF1 etc.
- **Host-dependent:** A host will be able to use the same virtual MAC address of an AVF as long as it is reachable.
- **Weighted:** If you want some AVFs to forward more traffic than others you can assign them a different weight.



Let's configure GLBP with the same topology. Make sure you get rid of all your HSRP or VRRP stuff first.

```
R1(config)#interface fa1/0
R1(config-if)#glbp 1 ip 192.168.1.3
R1(config-if)#glbp 1 priority 150
```

```
R2(config-if)#interface fa1/0
R2(config-if)#glbp 1 ip 192.168.1.3
```

I'll enable GLBP on R1 and R2 using the same group number (1). I changed the priority on R1 because I want it to be the AVG.

```
R1#show glbp brief
```

| Interface | Grp | Fwd | Pri | State  | Address        | Active router | Standby     |
|-----------|-----|-----|-----|--------|----------------|---------------|-------------|
| router    |     |     |     |        |                |               |             |
| Fa1/0     | 1   | -   | 150 | Active | 192.168.1.3    | local         | 192.168.1.2 |
| Fa1/0     | 1   | 1   | -   | Active | 0007.b400.0101 | local         | -           |
| Fa1/0     | 1   | 2   | -   | Listen | 0007.b400.0102 | 192.168.1.2   | -           |

```
R2#show glbp brief
```

| Interface | Grp | Fwd | Pri | State   | Address        | Active router | Standby |
|-----------|-----|-----|-----|---------|----------------|---------------|---------|
| Fal/0     | 1   | -   | 100 | Standby | 192.168.1.3    | 192.168.1.1   | local   |
| Fal/0     | 1   | 1   | -   | Listen  | 0007.b400.0101 | 192.168.1.1   | -       |
| Fal/0     | 1   | 2   | -   | Active  | 0007.b400.0102 | local         | -       |

Use the **show glbp brief** command to verify your configuration. There are a couple of things we can see here:

- R1 has become the AVG for group 1. R2 (192.168.1.2) is standby for the AVG role and will take over in case R1 fails.
- Group1 has two AVFs:
  - 1: R1: Virtual MAC address 0007.b400.0101.
  - 2: R2: Virtual MAC address 0007.b400.0102.

The **virtual MAC address that GLBP uses is 0007.b400.XXYY** (where X = GLBP group number and Y = AVF number).

```
C:\Documents and Settings\ComputerA>ipconfig
```

```
Windows IP Configuration
```

```
Ethernet adapter Local Area Connection:
```

```

IP Address. . . . . : 192.168.1.200
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.3

```

```
C:\Documents and Settings\ComputerB>ipconfig
```

```
Windows IP Configuration
```

```
Ethernet adapter Local Area Connection:
```

```

IP Address. . . . . : 192.168.1.201
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.3

```

We can use our computers to check which virtual MAC address they use for their gateway. Make sure both use the same gateway IP address (192.168.1.3).

```
C:\Documents and Settings\ComputerA>arp -a
```

```
Interface: 192.168.1.200--- 0x2
```

| Internet Address | Physical Address  | Type    |
|------------------|-------------------|---------|
| 192.168.1.3      | 00-07-b4-00-01-01 | dynamic |

```
C:\Documents and Settings\ComputerB>arp -a
```

```
Interface: 192.168.1.201-- 0x2
  Internet Address      Physical Address      Type
  192.168.1.3           00-07-b4-00-01-02    dynamic
```

You can see ComputerA uses the virtual MAC address of R1 (00-07-b4-00-01-01) while ComputerB uses the virtual MAC address of R2 (00-07-b4-00-01-02) for the same IP address (192.168.1.3). This is how GLBP will load balance traffic from hosts.

```
R1(config)#interface fa1/0
R1(config-if)#glbp 1 preempt
R1(config-if)#glbp 1 authentication md5 key-string mypass
```

```
R2(config)#interface fa1/0
R2(config-if)#glbp 1 preempt
R2(config-if)#glbp 1 authentication md5 key-string mypass
```

If you want you can configure things like preemption and authentication just like HSRP or VRRP. The configuration is the same but now you use the "glbp" command.

This is everything I wanted to show you about GLBP and this is also the end of this chapter. You have now seen how HSRP, VRRP and GLBP operate and are ready to bring redundancy to your gateways.

If you want to practice the configuration of these protocols you can take a look at some of the pre-built labs I created for all three protocols:

<http://gns3vault.com/Network-Services/hot-standby-routing-protocol.html>

<http://gns3vault.com/Network-Services/vrrp-virtual-router-redundancy-protocol.html>

<http://gns3vault.com/Network-Services/glbp-gateway-load-balancing-protocol.html>

## 15. Distance Vector Routing Protocols

In the “IP Routing” chapter you learned that routers need to have entries in their routing table so they know where to send IP packets and that we can fill this routing table with static routes (doing everything ourselves) or with routing protocols.

Before Cisco changed the exam(s) in 2013, CCNA students had to study 3 different routing protocols:

- RIP (Routing Information Protocol)
- OSPF (Open Shortest Path First)
- EIGRP (Enhanced Interior Gateway Routing Protocol)

Now you only have to learn about OSPF and EIGRP which makes sense since RIP is a very old / outdated protocol and it’s unlikely that you will see it on a network nowadays.

However I really liked RIP in CCNA for two reasons:

- It’s a very simple routing protocol and easy to explain, so it was a good way to introduce students to routing protocols.
- Some of the mechanics behind RIP still apply to EIGRP, so before diving into EIGRP it is easier to play with RIP first...kinda like you should learn riding a bike first before jumping on a motorcycle.

There are two “types” of routing protocols that we use on our internal networks:

- **Distance Vector**
- **Link-State**

Both RIP and EIGRP are distance vector routing protocols, but Cisco likes to call EIGRP an “advanced distance vector” routing protocol. OSPF is a link-state routing protocol (in the next chapter you will learn about OSPF).

You could see this “type” like the *engine* of the routing protocol; it’s how it learns and exchanges information.

This chapter is an introduction to how a distance vector routing protocol operates and even though RIP is no longer on the exam, I’m going to show you a configuration example to turn the theoretical talk into practice. Just keep in mind that you don’t have to worry about RIP for the exam!

Having said that, let’s dive into distance vector! We’ll start by paying attention to the distance vector class. What does the name distance vector mean?

- **Distance:** How far away, in routing world we use metrics which we just discussed.
- **Vector:** Which direction, in routing world we care about which interface and the IP-address of the next router to send it to.

I don’t know if you ever go cycling but here in The Netherlands we have some nice so called mushroom signposts telling you which way to go and how far (in kilometers) the destination is. The same principle applies to distance vector routing protocols.

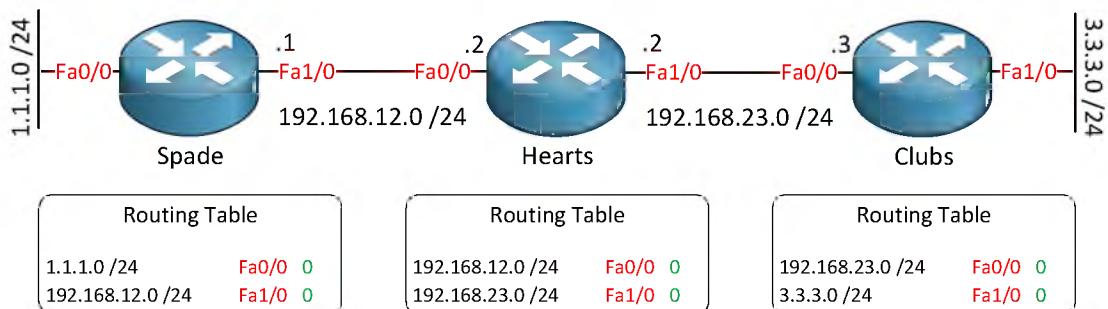




Photo: E. Dronkert (<http://flic.kr/dnet>)

You have to trust the information on the sign; the only thing you know is what direction you should go to (vector) and how far away it is (distance). Hopefully when you get closer to your destination you will find another sign that tells where to go from there...

Enough about cycling, let's see how this concept applies to distance vector routing protocols:

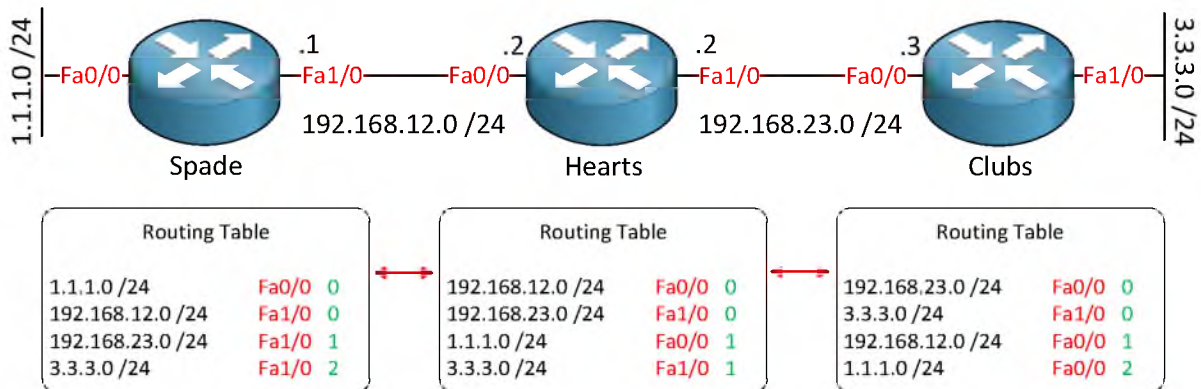




In this picture we have three routers and we are running a distance vector routing protocol. As we start our routers they build a routing table by default but the only thing they know are their directly connected interfaces. You can see that this information is in their routing table. In **red** you can see which interface and in **green** you can see the metric. RIP uses hop count as its metric which is nothing more than counting the number of routers (hops) you have to pass to get to your destination.

Now I'm going to enable distance vector routing, what will happen is that our routers will **copy their routing table** to their directly connected neighbor. Router Spade will copy its routing table to router Hearts. Router Hearts will copy its routing table to router Clubs and the other way around.

If a router receives information about a network it doesn't know about yet, it will add this information to its routing table:



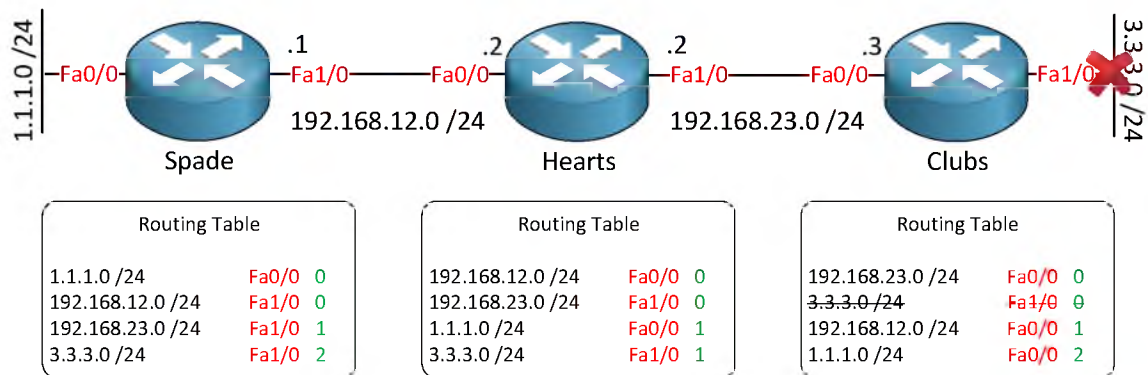
Take a look at router Spade and you will see that it has learned about the 192.168.23.0 /24 and 3.3.3.0 /24 network from router Hearts. You see that it has added the interface (Fa1/0) how to reach these networks (that's the vector part) and you see that it has added the metric (hop count) for these networks (that's the distance part).

192.168.23.0 /24 is one hop away, 3.3.3.0 /24 is two hops away.

Awesome! You also see that router Hearts and router Clubs have filled their routing tables.

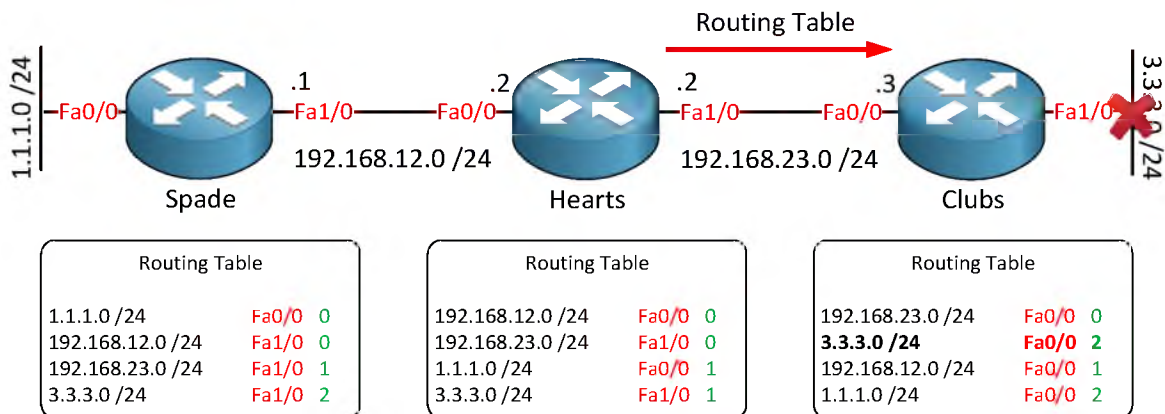
Every 30 seconds our routers will send a full copy of their routing table to their neighbors who can update their own routing table.

So far so good, our routers are working and we know the destination to all of our networks...distance vector routing protocols are vulnerable to some problems however. Let me show you what can go wrong:

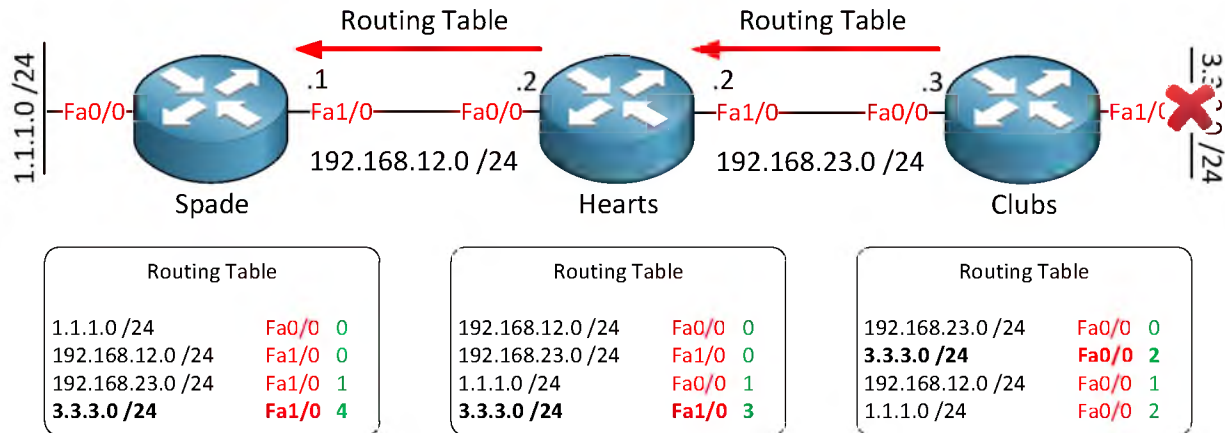


The FastEthernet 1/0 link on router Clubs is going down, so it will change its routing table. Its status went from 0 to Down.

Every 30 seconds our routers send a full copy of their routing table to their neighbors and it just happens to be that it's time for router Hearts to send a copy. Router Hearts sends its full routing table towards router Clubs. What do you think will happen?



Router Clubs gets the routing table from router Hearts and will see that router Hearts is advertising the 3.3.3.0 /24 network with a hop count of 1. That's excellent is what router Clubs thinks....a hop count of 1 is better than having a network that is down. Router Clubs will add this information to its routing table.



A few seconds later it's time for router Clubs to send its routing table to its neighbor router Hearts. Router Hearts will come to the following conclusion:

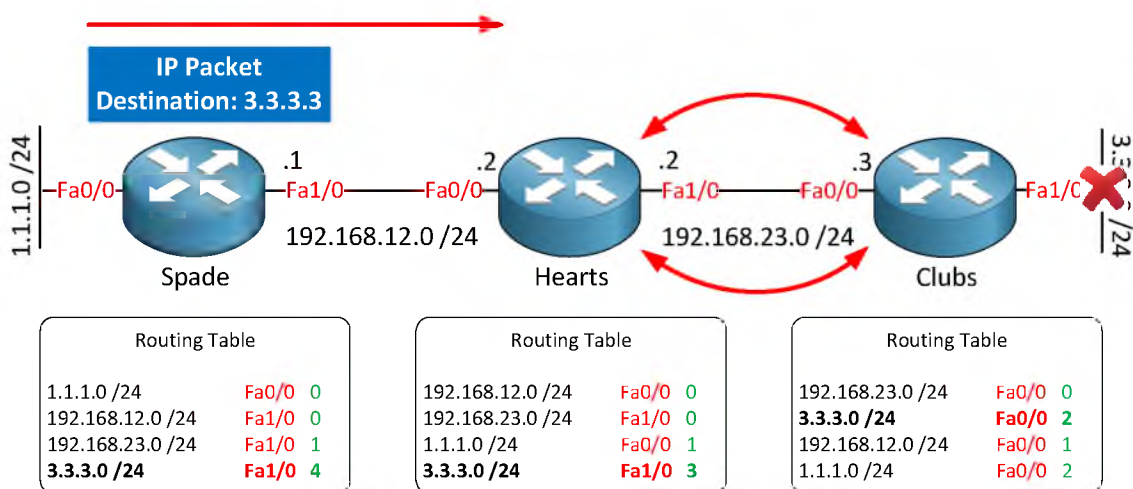
"I can reach 3.3.3.0 /24 by going to router Clubs and my hop count used to be 1. I'm receiving a routing table from router Clubs now and it now says that the hop count is 2...I need to update myself to include this change!"

The hop count on router Hearts is now 3, it received 2 from router Clubs plus it adds the hop towards router Clubs.

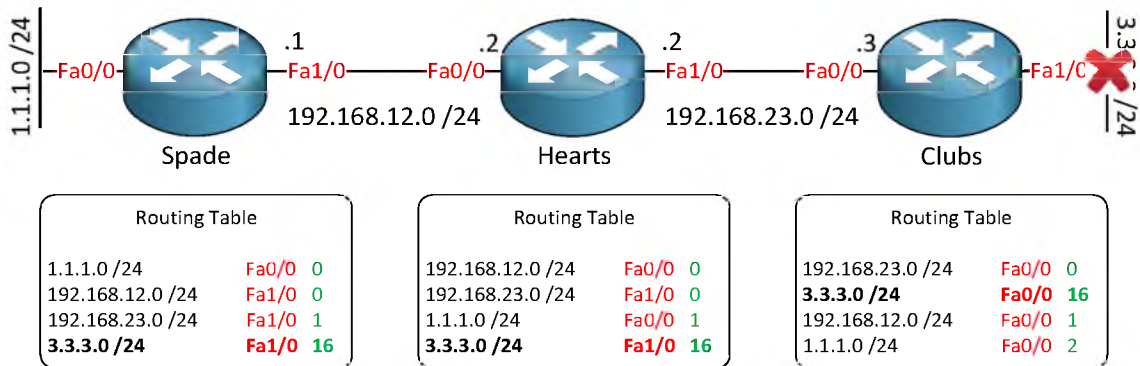
Router Hearts will also send a copy of its routing table towards router Spade who will update itself as well.

Do you see where this is going? These routers are going to keep **updating themselves to infinity**.

What will happen when we send an IP packet to the 3.3.3.0 /24 network?



Look at the routing table of router Hearts and router Clubs, they are pointing to each other. Ladies and Gentleman...we have a routing loop! That's not a good thing, IP packets have a TTL (Time to Live) field however so they won't loop around forever like Ethernet Frames do.



To prevent routers from updating themselves over and over again we have a **maximum**. For RIP this is a **hop count of 16**. 16 is considered unreachable so the maximum number of hops you can have is 15.

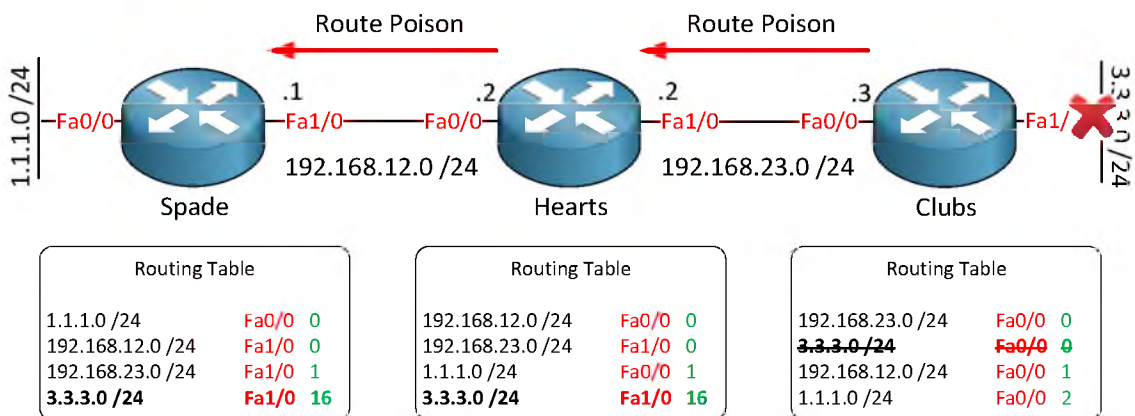
This problem is called **counting to infinity**.

There is something else we use to counter the counting to infinity problem. In our example router Clubs advertised the 3.3.3.0 /24 network towards router Hearts. How useful is it that router Hearts advertises this network towards router Clubs?



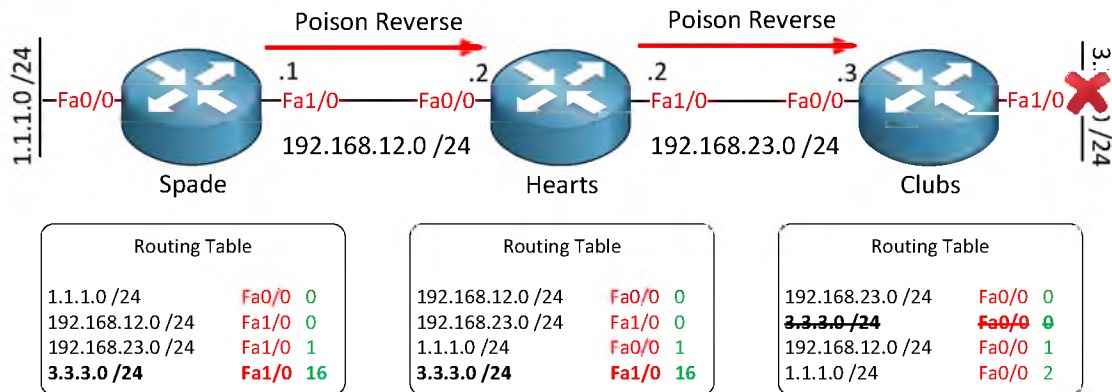
That's like telling someone a joke that you just learned from that person...not very effective (unless the joke is extremely good/lame ☺)

In routing it's not very effective so whatever you learn from your neighbor you are **not** going to advertise back to him. We call this **split horizon**.



Something else we do is that once a network goes down (3.3.3.0 /24 in our example) the router will send a **triggered update** immediately to update its neighbors.

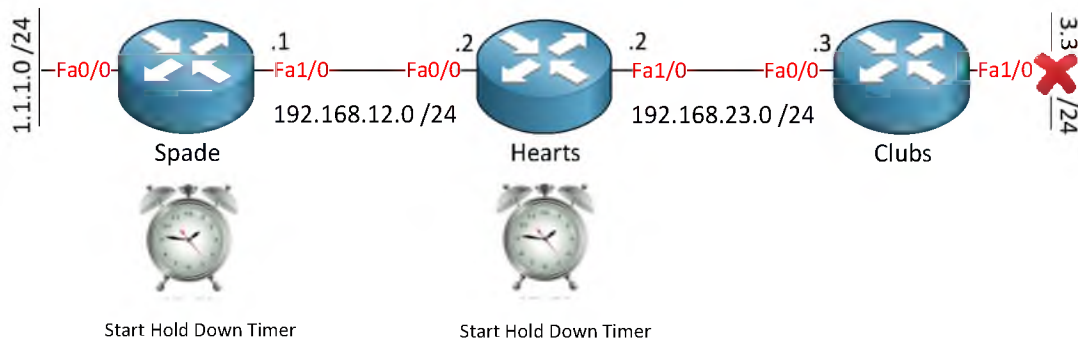
The triggered update will contain the network that is down and an infinite metric (16 in the case of RIP). Sending a update for this network with an infinite metric is called **route poisoning**.



To make sure router Clubs does not update itself via some other router / path in the network, router Hearts will send a **poison reverse** in response to the **route poison** it has received from router Clubs.

I just explained to you what split horizon is..."don't advertise whatever you learned from your neighbor back to them".

## "Route poisoning overrules split horizon"

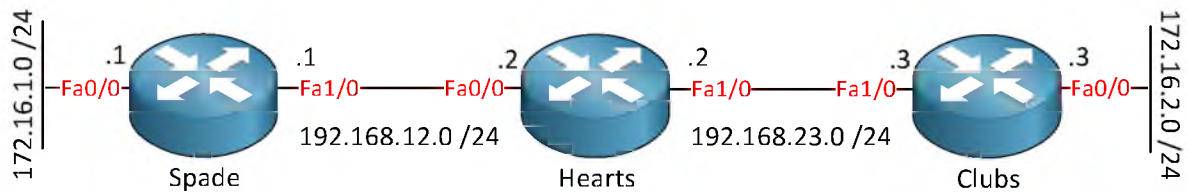


There is one more thing we use with distance vector routing protocols. When router Hearts and Spade find out that our 3.3.3.0 /24 network is down they will also start a **holddown timer**. This holddown timer will run for 180 seconds and it does the following:

- If we receive information about the 3.3.3.0 /24 network from another router with the same or worse metric than we currently have, we ignore this information.
- If we receive information about the 3.3.3.0 /24 network from another router with a better metric, we stop the holddown timer and update our routing table with this new information.
- If we don't receive anything and the holddown timer elapses we remove this network from the routing table.

What do you think? That's quite some theoretical information right?

To demonstrate that this isn't just theory but that distance vector really works like this I will use the following topology to demonstrate this:



Let's start with the configuration of the IP addresses on each router:

```

Spade>enable
Spade#configure terminal
Spade(config)#interface fastEthernet 0/0
Spade(config-if)#no shutdown
Spade(config-if)#ip address 172.16.1.1 255.255.255.0
Spade(config-if)#exit
Spade(config)#interface fastEthernet 1/0
Spade(config-if)#ip address 192.168.12.1 255.255.255.0
Spade(config-if)#no shutdown
  
```

```

Hearts>enable
Hearts#configure terminal
Hearts(config)#interface fastEthernet 0/0
Hearts(config-if)#no shutdown
Hearts(config-if)#ip address 192.168.12.2 255.255.255.0
Hearts(config-if)#exit
Hearts(config)#interface FastEthernet 1/0
Hearts(config-if)#no shutdown
Hearts(config-if)#ip address 192.168.23.2 255.255.255.0
Hearts(config-if)#exit
  
```



```
Clubs>enable
Clubs#configure terminal
Clubs(config)#interface fastEthernet 0/0
Clubs(config-if)#no shutdown
Clubs(config-if)#ip address 172.16.2.3 255.255.255.0
Clubs(config-if)#exit
Clubs(config)#interface fastEthernet 1/0
Clubs(config-if)#no shutdown
Clubs(config-if)#ip address 192.168.23.3 255.255.255.0
Clubs(config-if)#exit
```

Before we configure a routing protocol, let's check the routing tables of our routers:

```
Spade#show ip route
Codes: 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 - S-IS level-2
       ia - IS-IS inter area, * - candidate default,
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C      192.168.12.0/24 is directly connected, FastEthernet1/0
      172.16.0.0/24 is subnetted, 1 subnets
C      172.16.1.0 is directly connected, FastEthernet0/0
```

```
Hearts#show ip route
Codes: 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 - S-IS level-2
       ia - IS-IS inter area, * - candidate default,
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C      192.168.12.0/24 is directly connected, FastEthernet0/0
C      192.168.23.0/24 is directly connected, FastEthernet1/0
```

```
Clubs#show ip route
Codes: 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 - S-IS level-2
       ia - IS-IS inter area, * - candidate default,
       o - ODR, P - periodic downloaded static route
```

Gateway of last resort is not set

172.16.0.0/24 is subnetted, 1 subnets

```
C      172.16.2.0 is directly connected, FastEthernet0/0
C      192.168.23.0/24 is directly connected, FastEthernet1/0
```

Right now our routers only know 1 thing...their directly connected networks.

Let's configure RIP on router Spade and Hearts to see what happens:

```
Spade(config)#router rip
Spade(config-router)#network 192.168.12.0
Spade(config-router)#network 172.16.1.0
```

```
Hearts(config)#router rip
Hearts(config-router)#network 192.168.12.0
Hearts(config-router)#network 192.168.23.0
```

The network command is used to configure what networks you want to advertise in your routing protocol, in the next chapter I'll teach you exactly how it works. Let's check the routing tables of router Spade and Hearts!

```
Spade#show ip route rip
R      192.168.23.0/24 [120/1] via 192.168.12.2, 00:00:16, FastEthernet1/0
```

```
Hearts#show ip route rip
R      172.16.0.0/16 [120/1] via 192.168.12.1, 00:00:21, FastEthernet0/0
```

Router Spade has learned about network 192.168.23.0/24 and router Hearts has learned about network 172.16.0.0/16.

Let's take a closer look at this entry of router Spade:

```
R      192.168.23.0/24 [120/1] via 192.168.12.2, 00:00:16, FastEthernet1/0
```

I'll break it down so you understand what everything means.

```
R      192.168.23.0/24
```

The "R" means that this entry was learned through RIP and 192.168.23.0 /24 is the network that we learned.

```
[120/1]
```



The first number (120) is the administrative distance. The second number (1) is the metric. RIP uses "hop count" as the metric so network 192.168.23.0/24 is 1 hop away.

```
via 192.168.12.2
```

This is the next hop IP address. If we want to reach network 192.168.23.0 /24 we'll send IP packets to 192.168.12.2.

```
00:00:16
```

This is the time since the last update for this entry that we received from our neighbor.

```
FastEthernet1/0
```

This is the outgoing interface. When we want to reach network 192.168.23.0 /24 we'll use this interface for outgoing traffic.

I haven't configured anything on router Clubs yet so let's enable RIP on it so it can join the routing fun:

```
Clubs(config)#router rip
Clubs(config-router)#network 172.16.0.0
Clubs(config-router)#network 192.168.23.0
```

First I'll enable the network commands so it advertises all its networks.

Now let's see what we find in our routing tables:

```
Spade#show ip route rip
R    192.168.23.0/24 [120/1] via 192.168.12.2, 00:00:21, FastEthernet1/0
```

```
Hearts#show ip route rip
R    172.16.0.0/16 [120/1] via 192.168.23.3, 00:00:15, FastEthernet1/0
      [120/1] via 192.168.12.1, 00:00:11, FastEthernet0/0
```

```
Clubs#show ip route rip
R    192.168.12.0/24 [120/1] via 192.168.23.2, 00:00:26, FastEthernet1/0
```

Now this is an interesting output. There are a couple of interesting things here:

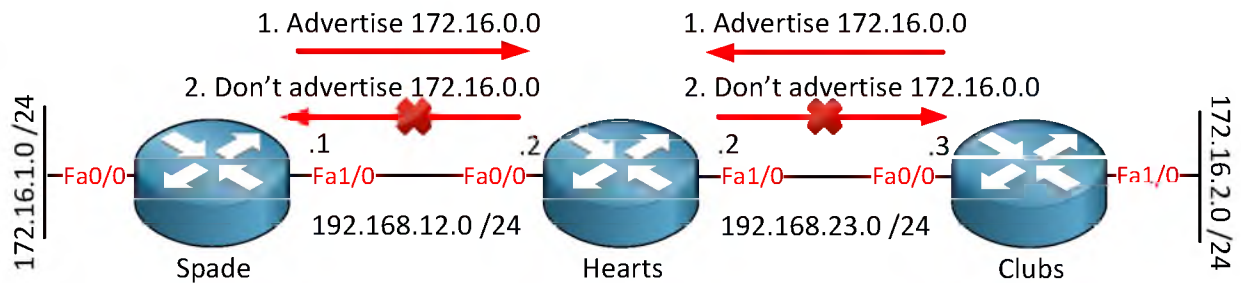
- Router Spade didn't learn about network 172.16.2.0 /24.
- Router Clubs didn't learn about network 172.16.1.0 /24.
- Router Hearts has 172.16.0.0 /16 in its routing table with two next hop IP addresses:
  - 192.168.23.3.
  - 192.168.12.1.

At this moment we are running RIP version 1 which is **classful**. This means that router Spade advertises 172.16.1.0 /24 as 172.16.0.0 and router Clubs advertises 172.16.2.0 /24 as 172.16.0.0.

So what does router Hearts do with this information? Whenever you learn about a **network from two different sources with the same metric, your router will do load-balancing**. For example when router Hearts receives an IP packet meant for 172.16.1.1 it might send it to router Clubs and it will be dropped.

Router Spade and Clubs are not learning about network 172.16.1.0 /24 or 172.16.2.0 /24 but why is this happening?

Let me show you a picture:



Router Spade and Clubs both advertise 172.16.0.0 to router Hearts. Router Hearts stores this information in its routing table but doesn't advertise 172.16.0.0 to router Spade or Clubs because of split-horizon. **You don't advertise to your neighbor what you learned from them...**

So how to fix this? We need to make sure that router Spade and Clubs send a subnet mask with their routing updates. We need a classless routing protocol for this.

There is a very useful command to verify what routing protocols you are using, let me show you:

```
Spade#show ip protocols
Routing Protocol is "rip"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Sending updates every 30 seconds, next due in 21 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Redistributing: rip
  Default version control: send version 1, receive any version
    Interface          Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0      1     1 2
    FastEthernet1/0      1     1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.16.0.0
    192.168.12.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    192.168.12.2      120          00:00:03
  Distance: (default is 120)
```

**Show ip protocols** is a very useful command so I recommend to write it down somewhere. This isn't just for RIP but it shows you information for **any other routing protocol**. We can see that by default we are **running RIP version 1** and that we are sending RIP updates on interface FastEthernet 0/0 and 1/0. You can also see the networks that we are advertising (172.16.0.0 and 192.168.12.0).

To show you the difference between a classful and classless routing protocol I will change the RIP version from version 1 (classful) to version 2 (classless):

```
Spade(config)#router rip
Spade(config-router)#version 2
Spade(config-router)#no auto-summary
```

```
Hearts(config)#router rip
Hearts(config-router)#version 2
Hearts(config-router)#no auto-summary
```

```
Clubs(config)#router rip
Clubs(config-router)#version 2
Clubs(config-router)#no auto-summary
```



*Make sure you understand the difference between classful and classless routing protocols. Even though RIP is no longer on the exam, you still might get questions about distance vector and classful/classless behavior...*

We are now running RIP version 2 which is classless. Let's see if there's a difference in our routing tables:

```
Spade#show ip route rip
      172.16.0.0/24 is subnetted, 2 subnets
R       172.16.2.0 [120/2] via 192.168.12.2, 00:00:24, FastEthernet1/0
R       192.168.23.0/24 [120/1] via 192.168.12.2, 00:00:24, FastEthernet1/0
```

```
Hearts#show ip route rip
      172.16.0.0/24 is subnetted, 2 subnets
R       172.16.1.0 [120/1] via 192.168.12.1, 00:00:08, FastEthernet0/0
R       172.16.2.0 [120/1] via 192.168.23.3, 00:00:26, FastEthernet1/0
```

```
Clubs#show ip route rip
R       192.168.12.0/24 [120/1] via 192.168.23.2, 00:00:16, FastEthernet1/0
      172.16.0.0/24 is subnetted, 2 subnets
R       172.16.1.0 [120/2] via 192.168.23.2, 00:00:16, FastEthernet1/0
```

This is looking better. You can see that router Hearts has learned about network 172.16.1.0 /24 and 172.16.2.0 /24. This is because Spade and Clubs are advertising 172.16.1.0 /24 and 172.16.2.0 /24, not 172.16.0.0 anymore. Router Spade and Clubs also have learned about each other's networks.

Our routers now know about all the networks out there, if we want we can also see what is happening in **real-time** on our routers by using a **debug** command. This is a great way to see how our routers are learning about networks:

```
Hearts#debug ip rip
RIP protocol debugging is on
```

Debug IP rip is how we enable it.

You will see messages like these:

```
Hearts#  
RIP: received v2 update from 192.168.12.1 on FastEthernet0/0  
172.16.1.0/24 via 0.0.0.0 in 1 hops
```

Here's what you see when router Hearts receives a RIP update from router Spade. We learn about network 172.16.1.0 /24.

```
RIP: sending v2 update to 224.0.0.9 via FastEthernet0/0 (192.168.12.2)  
RIP: build update entries  
172.16.2.0/24 via 0.0.0.0, metric 2, tag 0  
192.168.23.0/24 via 0.0.0.0, metric 1, tag 0
```

This is the RIP update that router Hearts will send towards router Spade. It carries network 172.16.2.0 /24 and 192.168.23.0 /24.

You have now seen how distance vector routing protocols operate, the problems that might occur and the solutions to fix these issues. RIP is no longer on the exam but you still have to understand the theory behind distance vector.

In the upcoming chapters you will learn how modern routing protocols like OSPF and EIGRP operate.

## 16. OSPF – Link-state routing protocol

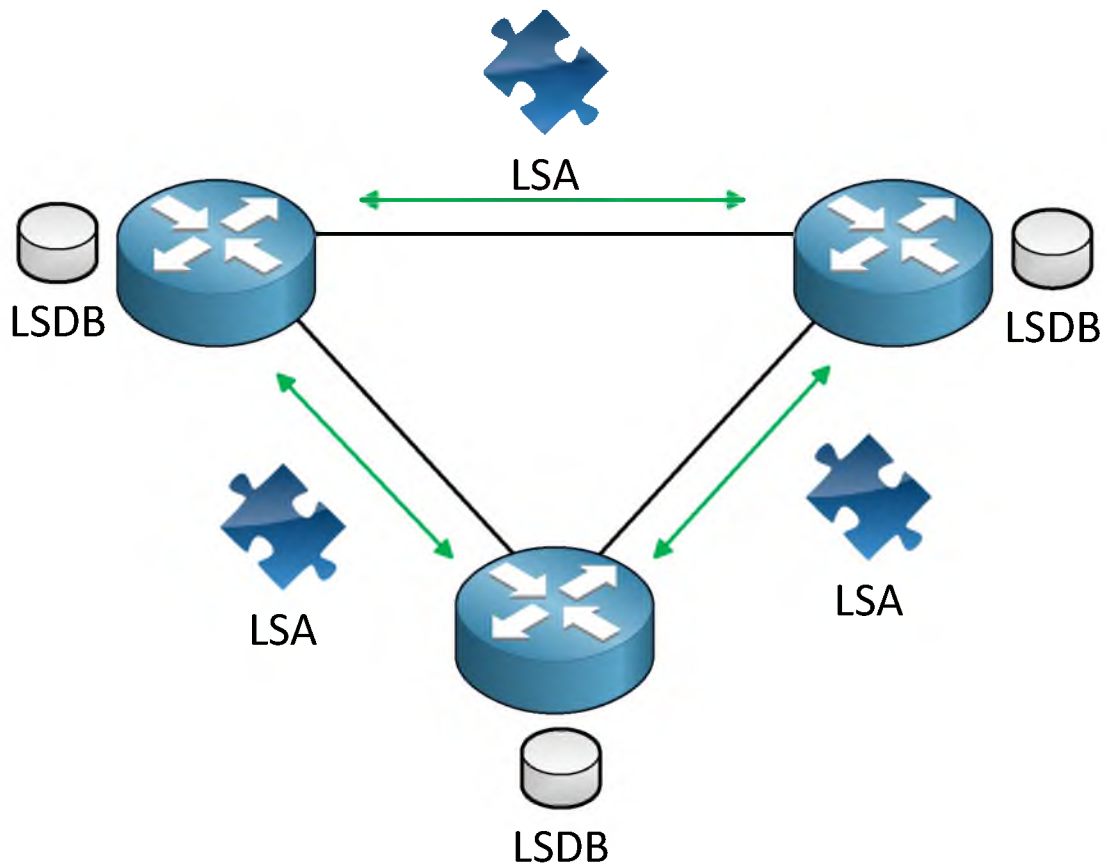
Another type of routing protocol we have is the link-state routing protocol. This class works completely different compared to the distance vector routing protocols. OSPF is a **link-state** routing protocol.

In this chapter we'll take a look at how link-state routing protocols work and how to configure OSPF.



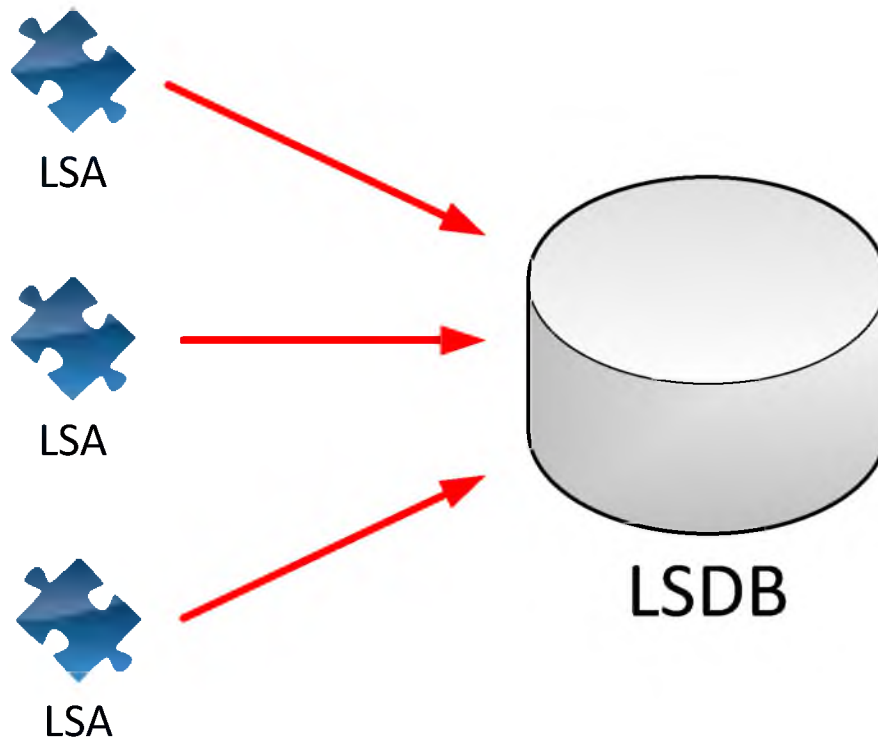
I don't know about you but I love my navigation system. The good thing about them is you can just drive and there is no need to look for traffic signs, the bad thing is that I'm absolutely lost when it's not working. I'm bad at reading maps (or maybe I just don't like it) and if I had to find my way to some street in any big city I'm doomed.

Link-state routing protocols are like your navigation system, they have a complete map of the network. If you have a full map of the network you can just calculate the shortest path to all the different destinations out there. This is cool because if you know about all the different paths it's impossible to get a loop since you know everything! The downside is that this is more CPU intensive than a distance vector routing protocol. It's just like your navigation system...if you calculate a route from New York to Los Angeles it's going to take a bit longer than when you calculate a route from one street to another street in the same city.



Let's take a good look at link-state and what it exactly means:

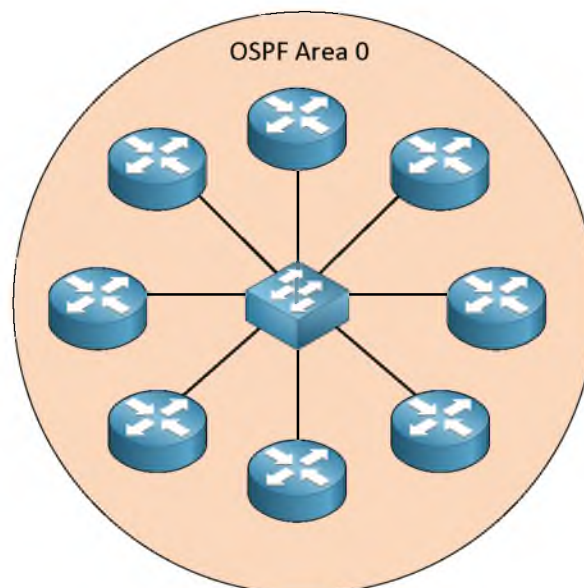
- Link: That's the interface of our router.
- State: Description of the interface and how it's connected to neighbor routers.



Link-state routing protocols operate by sending **link-state advertisements (LSA)** to all other link-state routers.

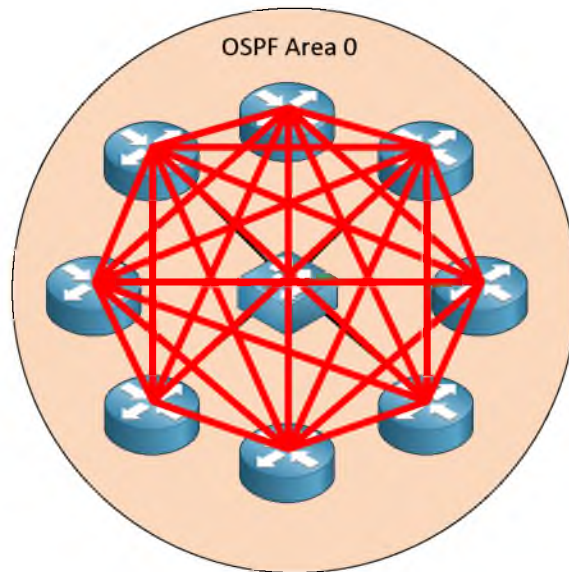
All the routers need to have these link-state advertisements so they can build their **link-state database** or **LSDB**. Basically all the link-state advertisements are a piece of the puzzle which builds the LSDB.

If you have a lot of OSPF routers it might not be very efficient that each OSPF router floods its LSAs to all other OSPF routers. Let me show you an example:

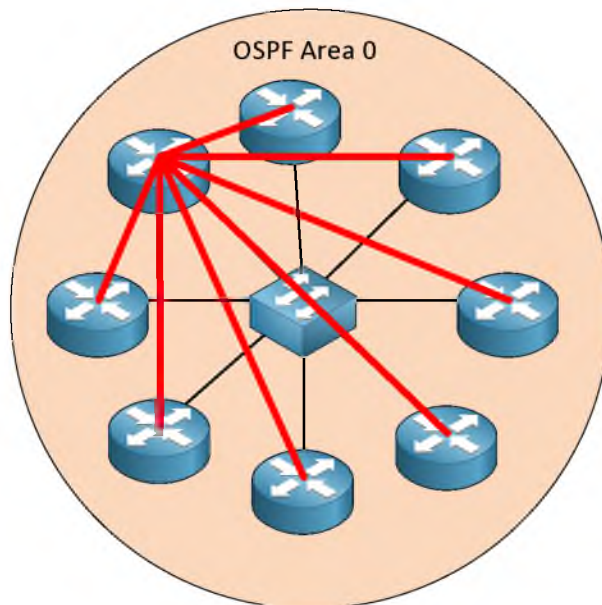




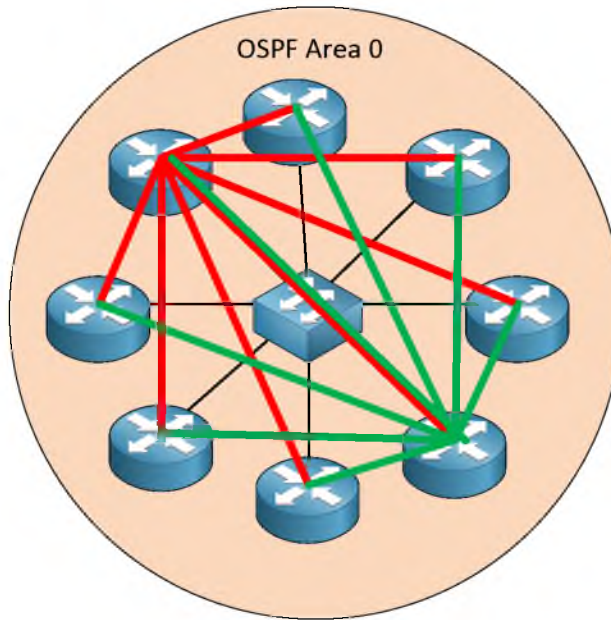
Above we have a network with 8 OSPF routers connected on a switch. Each of those routers is going to become OSPF neighbors with all of the other routers...sending hello packets, flooding LSAs and building the LSDB. This is what will happen:



We will get a full-mesh of OSPF neighbors. Each router will flood LSAs to all other routers so we will have a lot of OSPF traffic. Is there any way to make this a bit more efficient?



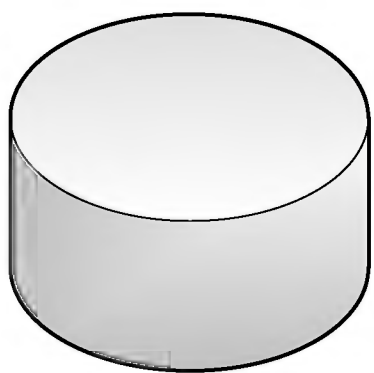
What if all our OSPF routers would just send their stuff to a single OSPF router who will then forward it to all the other OSPF routers? All our OSPF routers will know about all the routing information out there but we will have far less OSPF traffic. OSPF uses something called a **DR (Designated Router)**. All our OSPF routers will only form a "full" neighbor adjacency with the DR and not with all the other routers.



Since bad stuff can happen to our networks we want to have backup for our DR. If it crashes the **BDR (Backup Designated Router)** will take over. All our OSPF routers will only form **full neighbor adjacencies** with the DR and BDR and not with all other routers. This sounds efficient right?

We only use a DR/BDR on a **multi-access** network. An example of a multi-access network is using a switch. There is no need to do this on a **point-to-point** link. There is only one other router on the other side so there is no reason to select a DR/BDR.

The LSDB is our full picture of the network, in network terms we call this the **topology**. You could compare the LSDB to having a full map of your country.



=



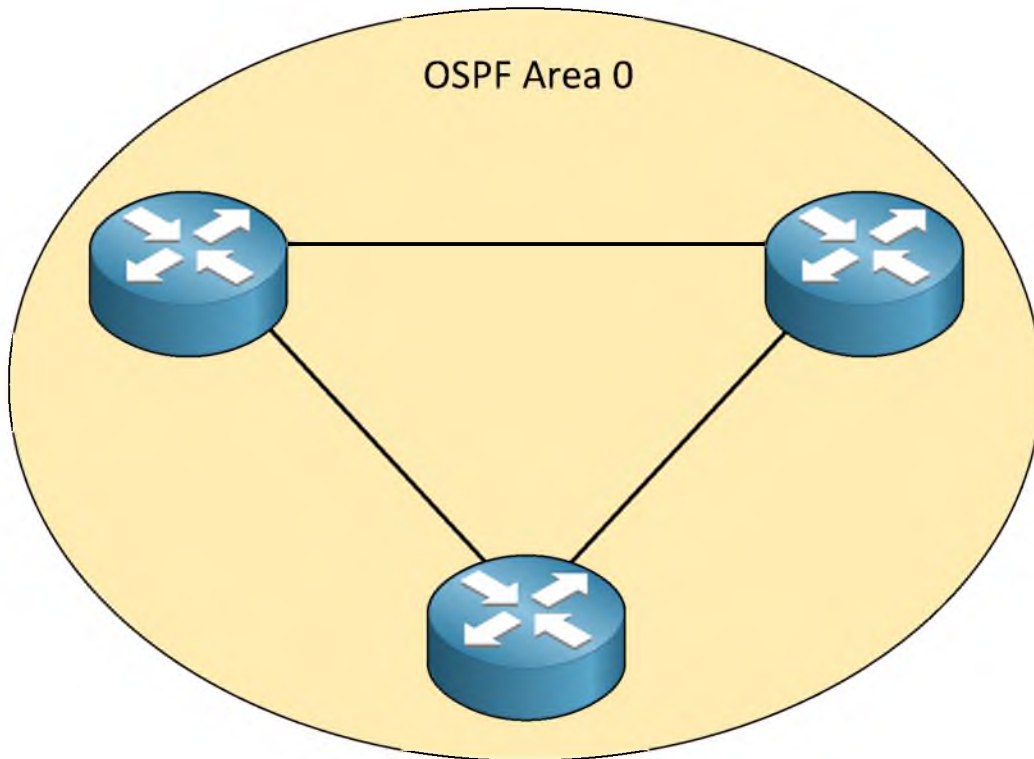
LSDB

Once every router has a complete map we can start calculating the shortest path to all the different destinations by using the **shortest-path first (SPF) algorithm**. The BEST

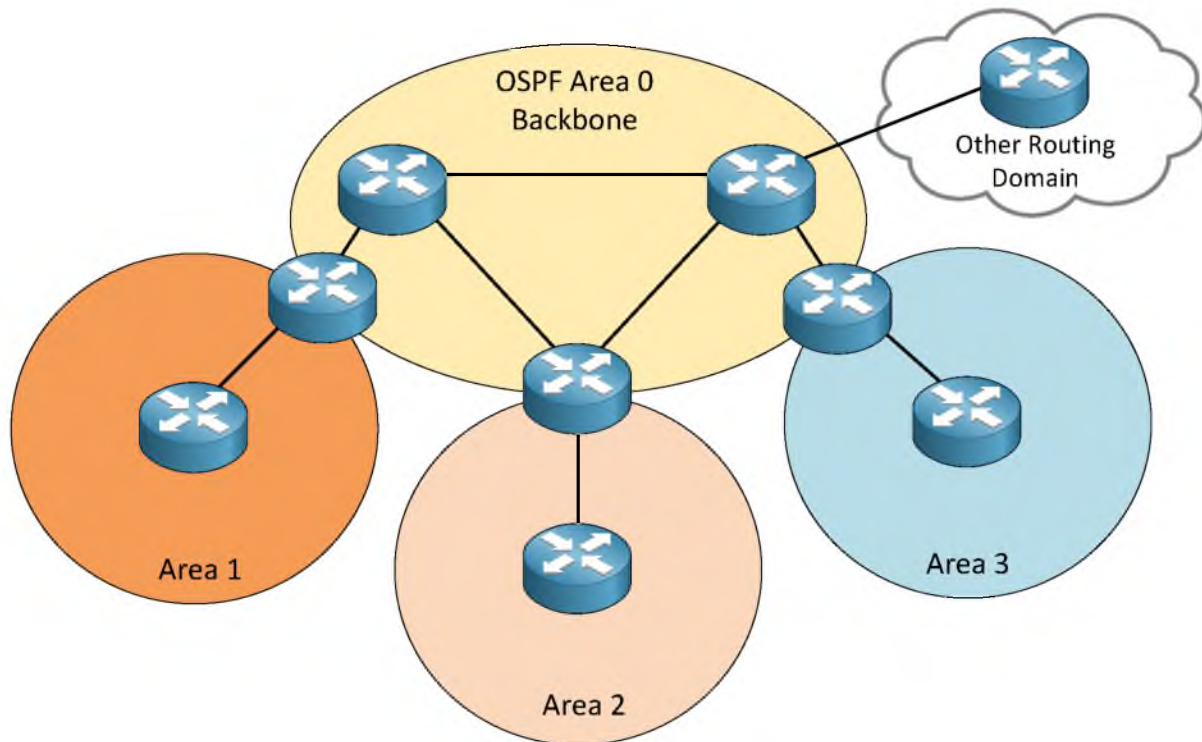
information goes into the routing table. Calculating the shortest path is like using your navigation system, it will look at the map and look at all the different ways of getting to the destination and only show you the best way of getting there.

There is only one link-state routing protocol we are going to discuss which is OSPF (Open Shortest Path First). There is another link-state routing protocol called IS-IS but it has been completely removed from CCNA, CCNP and even CCIE by Cisco.

Enough of my link-state routing protocol introduction let's take a look at OSPF and see how it operates.



OSPF works with the concepts of **areas** and by default you will always have a single area, normally this is area 0 or also called the **backbone** area.



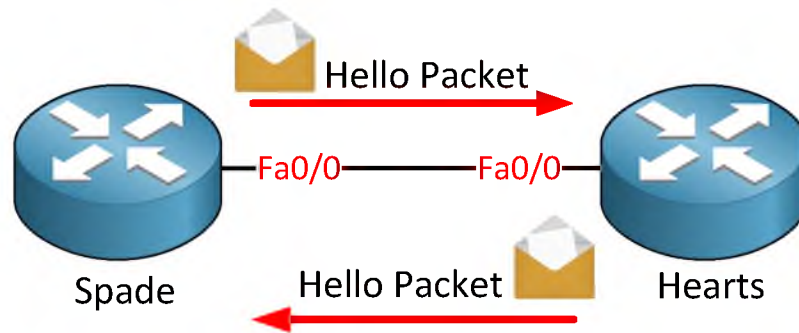
You can have multiple areas however as in the picture above, we have area 1,2 and 3. All of these areas **must connect** to the backbone area. If you want to go from area 1 to area 2 you **must** go through the backbone area to get there. It's impossible to go from area 1 directly to area 2; you always have to pass the backbone area! Same thing if you want to go from area 3 to area 2...you must cross the backbone area.

So why do we work with areas? Remember what I just explained about your navigation system. If you tell your navigation system to calculate from New York to Los Angeles it will take much longer than calculating a route from one street to another street in the same city. This calculating is called the **shortest path first** or **SPF** algorithm and the same thing apply to OSPF. Our routers only have a full picture of the network topology within the area, the smaller your map the faster your SPF algorithm works!

Keep in mind the SPF algorithm is from the 70's and OSPF was invented somewhere in the 80's...we didn't have fancy Core 2 Duo / Quad and I7's back then.

In the picture you also see on the top right something called "other routing domain". This could be another network running another routing protocol (perhaps EIGRP) and it's possible to import and export routes from EIGRP into OSPF or the other way around, this is called **redistribution**.

- Routers in the backbone area (area 0) are called backbone routers.
- Routers between 2 areas (like the one between area 0 and area 1) are called **area border routers** or **ABR**.
- Routers that run OSPF and are connected to another network that runs another routing protocol (for example EIGRP) are called **autonomous system border routers** or **ASBR**.



OSPF works differently than distance vector routing protocols, first of all it's a link-state but it also doesn't just send the link-state-advertisements around. Routers have to become neighbors first, once we have become neighbors we are going to exchange link-state advertisements.

Once you configure OSPF your router will start sending hello packets. If you also receive hello packets from the other router you will become neighbors.



Hello Packet

Router ID  
Hello / Dead Interval \*  
Neighbors  
Area ID \*  
Router Priority  
DR IP Address  
BDR IP Address  
Authentication  
Password \*  
Stub Area Flag \*

There's one catch however, there are a couple of fields in the hello packet and many of them have to match otherwise you won't become neighbors.



Let's walk through the items in the hello packet and see what they are about:

- **Router ID:** Each OSPF router needs to have a unique ID which is the highest IP address on any active interface. More about this later.
- **Hello / Dead Interval:** Every X seconds we are going to send a hello packet, if we don't hear any hello packets from our network for X seconds we declare you "dead" and we are no longer neighbors. These values have to match on both sides in order to become neighbors.
- **Neighbors:** All other routers who are your neighbors are specified in the hello packet.
- **Area ID:** This is the area you are in. This value has to match on both sides in order to become neighbors.
- **Router Priority:** This value is used to determine who will become designated or backup designated router.
- **DR and BDR IP address:** Designated and Backup Designated router, these are outside the scope of the CCNA so I'm going to skip it.
- **Authentication password:** You can use clear text and MD5 authentication for OSPF which means every packet will be authenticated. Obviously you need the same password on both routers in order to make things work.
- **Stub area flag:** Besides area numbers OSPF has different area types, this is outside the scope of the CCNA so I'm going to skip it. Both routers have to agree on the area type in order to become neighbors.



Each OSPF router needs to have a unique router ID which is based on the highest IP address on any active interface, there is a catch here however.

On any Cisco router you can create loopback interfaces which is like a virtual interface. You can configure an IP address on it and when you try to ping it you will always get a response.

If you have a loopback interface on your OSPF router then this IP address will be used as the router ID even when it's not the highest active IP address. Why does OSPF do this? Well it makes sense...your loopback interface will never go down unless your router crashes.

Using a loopback interface you can do two things:

- Advertise the IP address on the loopback interface in OSPF.
- Don't advertise the IP address on the loopback interface in OSPF.

What's the difference? Well if you advertise it other routers will be able to reach and ping the IP address on this loopback interface or even use it to telnet into the router. If you don't then well you can't...it's as easy as that.

Everything is well, we have configured OSPF...we have become neighbors with a bunch of routers and they have exchanged link-state advertisements. Our routers have built their LSDB and they have a full topology picture of our network. Next step is to run the SPF algorithm and see what the shortest path to our destination is.

Remember **metrics**? The metric is what the routing protocol uses in order to determine the best path. OSPF uses a metric called **cost** which is based on the bandwidth of an interface, it works like this:

$$\text{Cost} = \text{Reference Bandwidth} / \text{Interface Bandwidth}$$

The reference bandwidth is a default value on Cisco routers which is a 100Mbit interface. You divide the reference bandwidth by the bandwidth of the interface and you'll get the cost.

Example: If you have a 100 Mbit interface what will the cost be?

Cost = Reference bandwidth / Interface bandwidth.

100 Mbit / 100 Mbit = COST 1

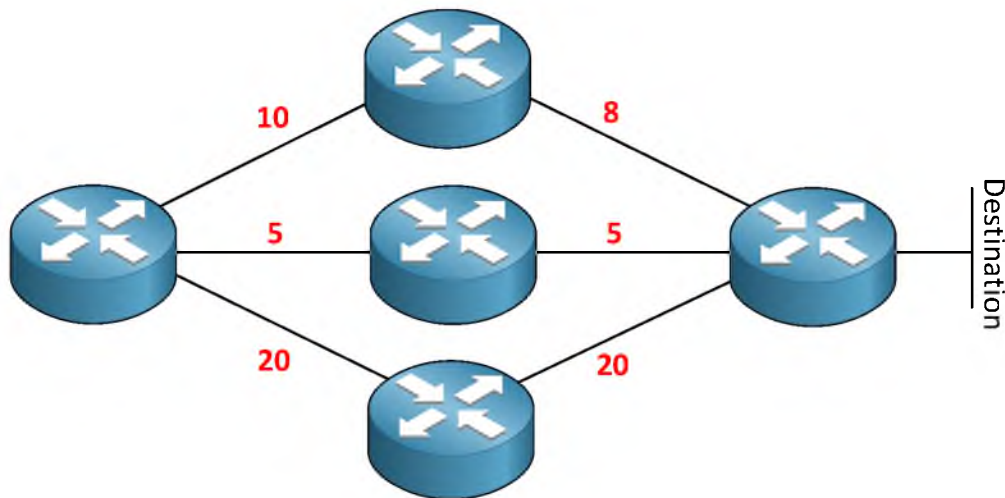
Example: If you have a 10 Mbit interface what will the cost be?

100 Mbit / 10 Mbit = COST 10

Example: If you have a 1 Mbit interface what will the cost be?

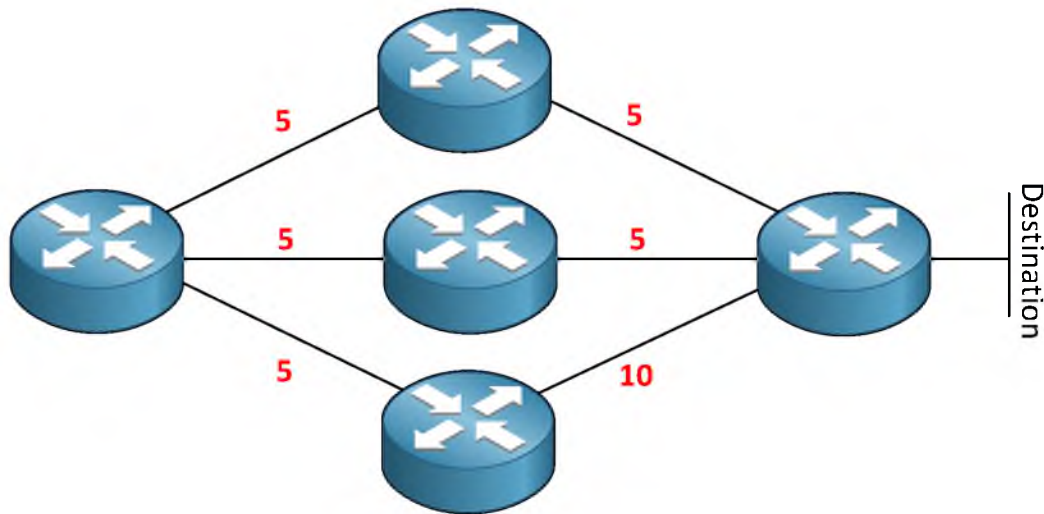
100 Mbit / 1 Mbit = COST 100

The **lower** the cost the better the path is.



Look at the picture above, we are sitting in the router on the left and running the SPF algorithm looking for the shortest path to our destination. Which one are we going to use?

Using the router on top we would have a cost of 10+8 which is 18. The path in the middle is 5+5 = 10. The router on the bottom we would have a cost of 20+20 = 40. The path in the middle obviously has the lowest cost so this is the path we are going to use!



Look at this picture: As you can see the path through the router on top and the middle router have the same cost ( $5+5 = 10$ ). What are we going to do here? The path is equal.

The answer is **load balancing**! We are going to use both paths and OSPF will load balance among them 50/50. Some things worth knowing about OSPF load balancing:

- Paths must have an equal cost.
- 4 equal cost paths will be placed in routing table.
- Maximum of 16 paths.
- To make paths equal cost, change the "cost" of a link

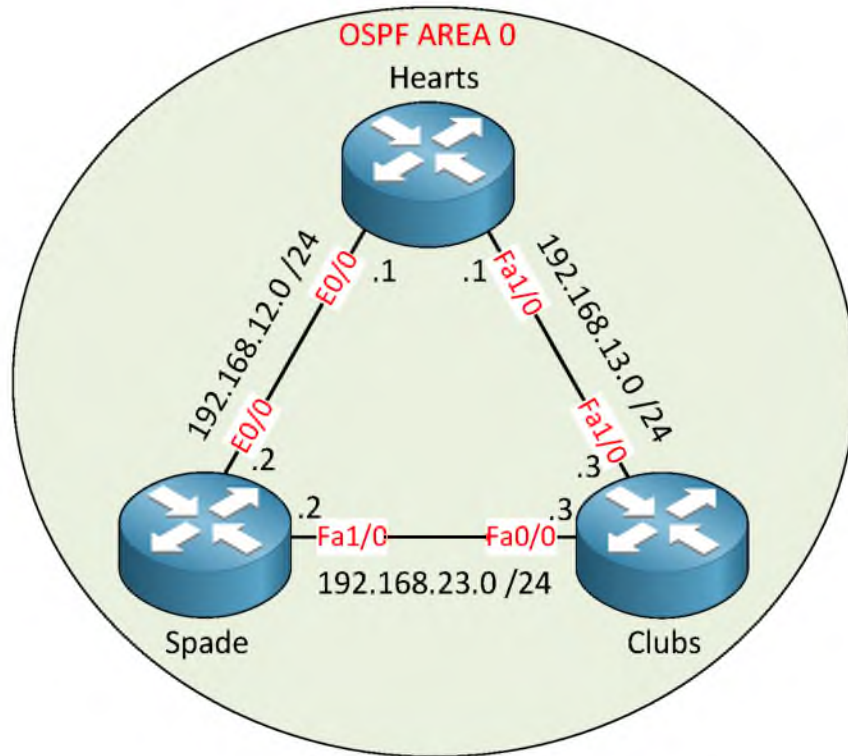
If a path is not equal we can make it so by manually changing the cost or bandwidth of an interface.

One last thing I'd like to tell you about OSPF is authentication:

- OSPF can do MD5 authentication.
- OSPF can do clear text authentication.
- You can enable authentication for the entire area or a single interface.



Now you know how OSPF works, let's walk through the configuration together to see what it's like on some real routers:



This is the topology that we'll use. All routers are in OSPF Area 0. Note that the link between router Spade and Clubs is an Ethernet (10Mbit) link. All other links are FastEthernet (100Mbit) interfaces.

We'll start with the configuration on router Spade:

```
Spade(config)#router ospf 1
Spade(config-router)#network 192.168.23.0 0.0.0.255 area 0
```

I need to use the **router ospf** command to get into the OSPF configuration. The number "1" is a process ID and you can choose any number you like. It doesn't matter and if you want you can use a different number on each router.

The second step is to use the network command. This command requires some further explanation:

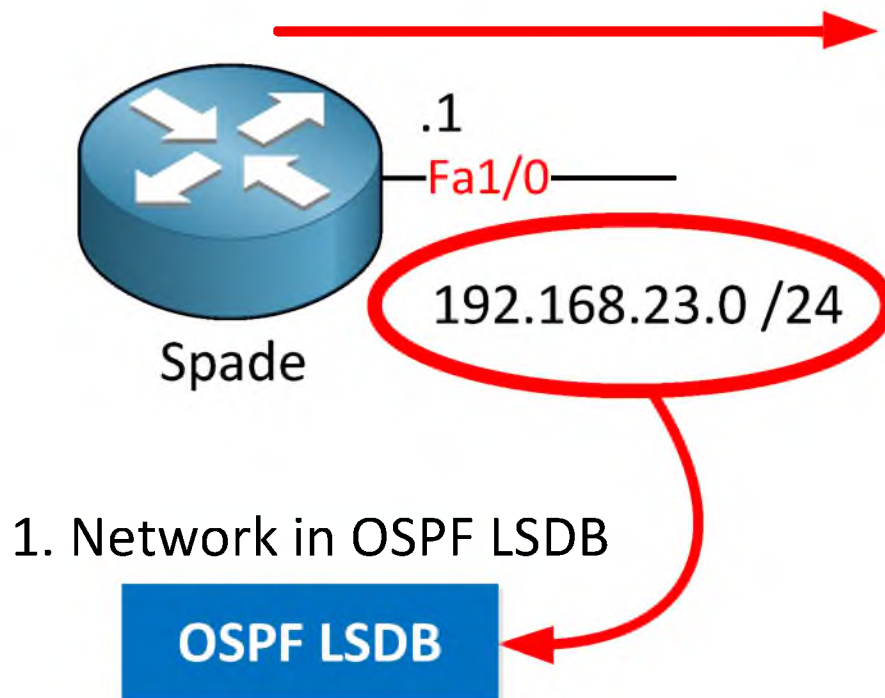
```
network 192.168.23.0 0.0.0.255 area 0
```

This command actually does two things:

- Advertise the networks that fall within this range in OSPF.
- Activate OSPF on the interface(s) that fall within this range. This means that OSPF will send hello packets on the interface.

Let me zoom in on router Spade so I can explain this command a bit more:

## 2. Send OSPF Hello Packets



## 1. Network in OSPF LSDB

By typing in the network command I am advertising the 192.168.23.0 /24 network into the OSPF LSDB (Link State Database) so that it can be exchanged with other OSPF routers, but I'm also sending OSPF hello packets on the interface that falls within the 192.168.23.0 /24 range (in this case that's Fa1/0).

In the example above that means that router Spade will try to become OSPF neighbors with router Clubs.



*When an interface is connected to a switch with computers, you probably don't want to send hello packets there since there is no OSPF router anyway. To block the hello packets but still advertise the network in OSPF you can use the **passive-interface** command on the interface.*

Let's look at the network command again:

```
network 192.168.23.0 0.0.0.255 area 0
```

Behind 192.168.23.0 you can see it says 0.0.0.255. This is not a subnet mask but a **wildcard mask**. A wildcard mask is a **reverse subnet mask**.

Let me give you an example:

| Subnet mask   | 255      | 255      | 255      | 0        |
|---------------|----------|----------|----------|----------|
|               | 11111111 | 11111111 | 11111111 | 00000000 |
| Wildcard mask | 0        | 0        | 0        | 255      |
|               | 00000000 | 00000000 | 00000000 | 11111111 |

When I say reverse subnet mask I mean that the binary 1s and 0s of the wildcard mask are flipped compared to the subnet mask. A subnet mask of 255.255.255.0 is the same as wildcard mask 0.0.0.255. Don't worry about this too much for now as I'll explain wildcard masks to you in detail when we talk about access-lists!

OSPF uses areas so you need to specify the area:

```
area 0
```

In our example we have configured single area OSPF. All routers belong to area 0. We will also enable OSPF on router Clubs:

```
Clubs(config)#router ospf 1
Clubs(config-router)#network 192.168.23.0 0.0.0.255 area 0
```

This network command will store network 192.168.23.0 /24 in the OSPF LSDB and router Clubs will start sending OSPF hello packets to router Spade.

After typing in my network command you'll see this message in the console:

```
Clubs# %OSPF-5-ADJCHG: Process 1, Nbr 192.168.23.2 on FastEthernet0/0 from
LOADING to FULL, Loading Done
```

```
Spade# %OSPF-5-ADJCHG: Process 1, Nbr 192.168.23.3 on FastEthernet1/0 from
LOADING to FULL, Loading Done
```

Great! It seems that router Clubs and Spade have become OSPF neighbors. There's another command we can use to verify that we have become neighbors:

```
Clubs#show ip ospf neighbor
```

| Neighbor ID  | Pri | State    | Dead Time | Address      | Interface       |
|--------------|-----|----------|-----------|--------------|-----------------|
| 192.168.23.2 | 1   | FULL/BDR | 00:00:36  | 192.168.23.2 | FastEthernet0/0 |

```
Spade#show ip ospf neighbor
```

| Neighbor ID  | Pri | State   | Dead Time | Address      | Interface       |
|--------------|-----|---------|-----------|--------------|-----------------|
| 192.168.23.3 | 1   | FULL/DR | 00:00:32  | 192.168.23.3 | FastEthernet1/0 |

**Show ip ospf neighbor** is a great command to see if your router has OSPF neighbors. When the state is **full** you know that the routers have successfully become neighbors.

Each OSPF router has a router ID and we check it with the show ip protocols command:

```
Spade#show ip protocols
Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.23.2
```

```
Clubs#show ip protocols
Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.23.3
```

Above you see the router ID of router Spade and Clubs. They used their highest active IP address as the router ID. Let's create a loopback on router Spade to see if the router ID changes...

```
Spade(config)#interface loopback 0
Spade(config-if)#ip address 2.2.2.2 255.255.255.0
```

This is how you create a loopback interface. You can pick any number that you like it really doesn't matter.

```
Spade#show ip protocols
Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.23.2
```

The router ID still the same. We need to reset the OSPF process before the change will take effect, this is how you do it:

```
Spade#clear ip ospf process
Reset ALL OSPF processes? [no]: yes
```

Use **clear ip ospf process** to reset OSPF. Let's see if there is a difference:

```
Spade#show ip protocols
Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 2.2.2.2
```

We can also change the router ID manually. Let me demonstrate this on router Clubs:

```
Clubs#show ip protocols
Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.23.3
```

Right now it's 192.168.23.3...

```
Clubs(config-router)#router-id 3.3.3.3
Reload or use "clear ip ospf process" command, for this to take effect
```

```
Clubs#clear ip ospf process
Reset ALL OSPF processes? [no]: yes
```

The router is friendly enough to warn me to reload or clear the OSPF process. Let's verify our configuration:

```
Clubs#show ip protocols
Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 3.3.3.3
```

As you can see above the router ID is now 3.3.3.3.



*Changing the router ID isn't something you would normally do. IP addresses on your router have to be unique so your OSPF router ID will also be unique. Understanding how OSPF selects a router ID is something you have to understand for the exam however.*

Right now we have an OSPF neighbor adjacency between router Spade and Clubs. Let's configure our routers so that Spade/Hearts and Hearts/Clubs also become OSPF neighbors:

```
Spade(config)#router ospf 1
Spade(config-router)#network 192.168.12.0 0.0.0.255 area 0
```

```
Hearts(config)#router ospf 1
Hearts(config-router)#network 192.168.12.0 0.0.0.255 area 0
Hearts(config-router)#network 192.168.13.0 0.0.0.255 area 0
```

```
Clubs(config)#router ospf 1
Clubs(config-router)#network 192.168.13.0 0.0.0.255 area 0
```

I'll advertise all networks in OSPF. Before we check the routing table it's a good idea to see if our routers have become OSPF neighbors:

```
Spade#show ip ospf neighbor
```

| Neighbor ID  | Pri | State    | Dead Time | Address      | Interface       |
|--------------|-----|----------|-----------|--------------|-----------------|
| 192.168.13.1 | 1   | FULL/BDR | 00:00:31  | 192.168.12.1 | Ethernet0/0     |
| 3.3.3.3      | 1   | FULL/DR  | 00:00:38  | 192.168.23.3 | FastEthernet1/0 |

```
Hearts#show ip ospf neighbor
```

| Neighbor ID | Pri | State    | Dead Time | Address      | Interface       |
|-------------|-----|----------|-----------|--------------|-----------------|
| 3.3.3.3     | 1   | FULL/BDR | 00:00:33  | 192.168.13.3 | FastEthernet1/0 |
| 2.2.2.2     | 1   | FULL/DR  | 00:00:30  | 192.168.12.2 | Ethernet0/0     |

```
Clubs#show ip ospf neighbor
```

| Neighbor ID  | Pri | State    | Dead Time | Address      | Interface       |
|--------------|-----|----------|-----------|--------------|-----------------|
| 192.168.13.1 | 1   | FULL/DR  | 00:00:37  | 192.168.13.1 | FastEthernet1/0 |
| 2.2.2.2      | 1   | FULL/BDR | 00:00:30  | 192.168.23.2 | FastEthernet0/0 |

Excellent our routers have become OSPF neighbors and the state is **full** which means they are done exchanging information.

Let's check the routing tables:

```
Spade#show ip route ospf
```

```
O    192.168.13.0/24 [110/2] via 192.168.23.3, 00:09:45, FastEthernet1/0
```

Router Spade has one entry, it's for network 192.168.13.0 /24. What exactly do we see here?

- The "O" stands for OSPF. This entry was learned through OSPF.
- 192.168.13.0 /24 is the network that we learned. This is the link between router Hearts and Clubs.
- The "110" is the administrative distance of OSPF.
- The "2" is the metric. OSPF uses cost as a metric. To reach this network we have a total cost of 2.
- "via" is the next hop IP address where we send our traffic to. This is router Clubs for this network.

How did OSPF exactly come up with this cost? Let's take a detailed look:

```
Spade#show ip ospf interface fa1/0
```

```
FastEthernet1/0 is up, line protocol is up
  Internet Address 192.168.23.2/24, Area 0
  Process ID 1, Router ID 2.2.2.2, Network Type BROADCAST, Cost: 1
```

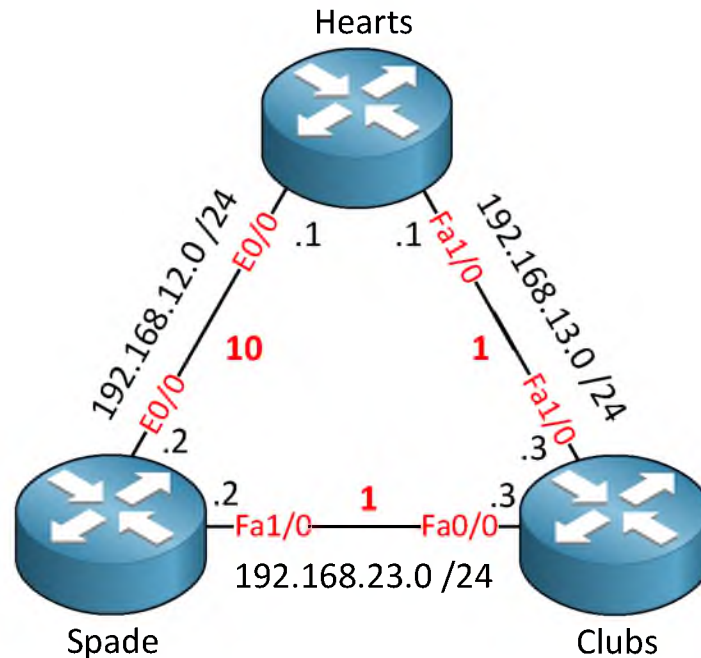
You can use the **show ip ospf interface** command to check the cost of a certain interface. As you can see a FastEthernet interface has a cost of 1.

```
Spade#show ip ospf interface e0/0
```

```
Ethernet0/0 is up, line protocol is up
  Internet Address 192.168.12.2/24, Area 0
  Process ID 1, Router ID 2.2.2.2, Network Type BROADCAST, Cost: 10
```

An Ethernet interface is slower and it has a cost of 10.

Let's draw a picture to illustrate this:



In the picture above I have added the cost of the interface. From router Spade I can reach network 192.168.13.0 /24 by going through router Clubs or Hearts. This is what the cost will be:

- Through Clubs: 1+1 = cost 2.
- Through Hearts: 10+1 = cost 11.

Obviously the path through router Clubs has the lowest cost. As an experiment we can shut the FastEthernet0/0 interface on router Clubs to see if Spade can find another path:

```
Clubs(config)#interface fastEthernet 0/0
Clubs(config-if)#shutdown
```

```
Spade#show ip route ospf
O 192.168.13.0/24 [110/11] via 192.168.12.1, 00:01:20, Ethernet0/0
```

Now you can see that router Spade will reach network 192.168.13.0 /24 through router Hearts and it has a total cost of 11. Before we continue let's enable the interface again:

```
Clubs(config)#interface fastEthernet 0/0
Clubs(config-if)#no shutdown
```

Let's verify our routing table:

```
Spade#show ip route ospf
O 192.168.13.0/24 [110/2] via 192.168.23.3, 00:00:01, FastEthernet1/0
```

Now it's using the FastEthernet1/0 interface again.





What if I wanted to force OSPF to use the slower Ethernet0/0 interface without shutting the FastEthernet interface? It's possible to manually change the cost, let me show you how:

```
Spade(config)#interface fastEthernet 1/0
Spade(config-if)#ip ospf cost 50
```

Use the **ip ospf cost** command to change the cost. When I set it to 50 the FastEthernet interface isn't attractive anymore.

```
Spade#show ip ospf interface fastEthernet 1/0
FastEthernet1/0 is up, line protocol is up
  Internet Address 192.168.23.2/24, Area 0
  Process ID 1, Router ID 2.2.2.2, Network Type BROADCAST, Cost: 50
```

I can verify that the cost is now 50.

```
Spade#show ip route ospf
O    192.168.13.0/24 [110/11] via 192.168.12.1, 00:01:20, Ethernet0/0
```

And as a result OSPF will prefer the slower Ethernet interface...

Let's get rid of this cost command and take a look at the other routing tables!

```
Spade(config)#interface fastEthernet 1/0
Spade(config-if)#no ip ospf cost 50
```

```
Hearts#show ip route ospf
O    192.168.23.0/24 [110/2] via 192.168.13.3, 00:00:15, FastEthernet1/0
```

Router Hearts has a single entry for the 192.168.23.0 /24 network through router Clubs. This is the shortest path and it has a cost of 2.

```
Clubs#show ip route ospf
O    192.168.12.0/24 [110/11] via 192.168.23.2, 00:01:14, FastEthernet0/0
    [110/11] via 192.168.13.1, 00:01:14, FastEthernet1/0
```

Router Clubs has two interesting entries. It has learned about the 192.168.12.0 /24 network and it can reach it through 192.168.23.2 (router Spade) or through 192.168.13.1 (router Hearts). The cost of both paths is 11. OSPF will do **load-balancing** to reach this network.

Do you remember our loopback0 interface on router Spade? We used it for the router ID but we can also advertise it in OSPF or any other routing protocol, let me show you:

```
Spade#show ip interface loopback 0
Loopback0 is up, line protocol is up
  Internet address is 2.2.2.2/24
```

As you can see it's a normal interface with an IP address and subnet mask on it.

Let's advertise it:

```
Spade(config)#router ospf 1
Spade(config-router)#network 2.2.2.0 0.0.0.255 area 0
```

I'll use the above network command to advertise it into OSPF. Let's check the routing tables:

```
Hearts#show ip route ospf
    2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/3] via 192.168.13.3, 00:00:26, FastEthernet1/0
O       192.168.23.0/24 [110/2] via 192.168.13.3, 00:00:26, FastEthernet1/0
```

Router Hearts will reach it by going through router Clubs. The total cost is 3:

1 (FastEthernet) + 1 (FastEthernet) + 1 (Loopback) = 3.

```
Clubs#show ip route ospf
O       192.168.12.0/24 [110/11] via 192.168.23.2, 00:01:42, FastEthernet0/0
        [110/11] via 192.168.13.1, 00:01:42, FastEthernet1/0
    2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/2] via 192.168.23.2, 00:01:42, FastEthernet0/0
```

Router Clubs has a total cost of 2:

1 (FastEthernet) + 1 (Loopback) = 2.

The great thing about loopback interfaces is that they are reachable just like normal interfaces:

```
Clubs#ping 2.2.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/8 ms
```

You can ping them just like any other interface.



*Loopback interfaces have many more uses but for CCNA it's great to use them to quickly advertise networks into your routing protocols.*

We can also advertise a default route into OSPF. This might be useful if your router is connected to the Internet and you want to advertise this to other routers, this is how you do it:

```
Spade(config)#router ospf 1
Spade(config-router)#default-information originate always
```

You need to use the **default-information originate** command. If you don't already have a default route in your routing table then you need to add the **always** keyword. Let's see if the default route has been advertised:

```
Hearts#show ip route ospf
      2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/3] via 192.168.13.3, 00:00:50, FastEthernet1/0
O       192.168.23.0/24 [110/2] via 192.168.13.3, 00:00:50, FastEthernet1/0
O*E2 0.0.0.0/0 [110/1] via 192.168.13.3, 00:00:50, FastEthernet1/0
```

```
Clubs#show ip route ospf
O       192.168.12.0/24 [110/11] via 192.168.23.2, 00:00:45, FastEthernet0/0
      [110/11] via 192.168.13.1, 00:00:45, FastEthernet1/0
      2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/2] via 192.168.23.2, 00:00:45, FastEthernet0/0
O*E2 0.0.0.0/0 [110/1] via 192.168.23.2, 00:00:45, FastEthernet0/0
```

As you can see router Hearts and Clubs have learned the default route from router Spade.

Let's continue our OSPF configuration. I want to show you how to do plaintext and MD5 authentication. I'll start by configuring plaintext authentication between router Spade and Clubs:

```
Spade(config)#interface fastEthernet 1/0
Spade(config-if)#ip ospf authentication
Spade(config-if)#ip ospf authentication-key secret
```

```
Clubs(config)#interface fastEthernet 0/0
Clubs(config-if)#ip ospf authentication
Clubs(config-if)#ip ospf authentication-key secret
```

First you need to use the **ip ospf authentication** command to enable plaintext authentication on the interface. Secondly we need to configure a password using the **ip ospf authentication-key** command.

Once you configure authentication on one router you'll see the neighbor adjacency going down for a moment until you configure the other router.

There is a useful debug command you can use to verify if authentication has been enabled or not:

```
Clubs#debug ip ospf packet
OSPF packet debugging is on
```

Debug ip ospf packet will show you an overview of the OSPF packets that you are receiving, it looks like this:

```
Clubs#
OSPF: rcv. v:2 t:1 l:48 rid:192.168.13.1
      aid:0.0.0.0 chk:7D95 aut:0 auk: from FastEthernet1/0
```

This is a packet that we received from router Hearts. The "aut:0" means that this packet is not authenticated. This is correct because we didn't configure authentication between router Clubs and Hearts.

```
OSPF: rcv. v:2 t:1 l:48 rid:2.2.2.2
      aid:0.0.0.0 chk:3339 aut:1 auk: from FastEthernet0/0
```

This packet is from router Spade and you can see it says "aut:1". This means we have enabled plaintext authentication. Let's disable debug before we continue:

```
Clubs#no debug all
All possible debugging has been turned off
```

Let's configure MD5 authentication between router Hearts and Clubs:

```
Clubs(config)#interface fastEthernet 1/0
Clubs(config-if)#ip ospf authentication message-digest
Clubs(config-if)#ip ospf message-digest-key 1 md5 mykey
```

```
Hearts(config)#interface fastEthernet 1/0
Hearts(config-if)#ip ospf authentication message-digest
Hearts(config-if)#ip ospf message-digest-key 1 md5 mykey
```

First we tell OSPF to use MD5 with the **ip ospf authentication message-digest** command. Secondly the **ip ospf message-digest-key** tells OSPF to use MD5 key 1 (you can pick any number you like as long it's the same on both routers) and password "mykey".

If you enable the debug you can see that it's working:

```
Hearts#debug ip ospf packet
OSPF packet debugging is on
```

```
Hearts#
OSPF: rcv. v:2 t:1 l:48 rid:3.3.3.3
      aid:0.0.0.0 chk:0 aut:2 keyid:1 seq:0x3C7EE6DC from FastEthernet1/0
```

```
Hearts#no debug all
All possible debugging has been turned off
```

In the output above you can see it says "auth:2" which means MD5 authentication. You can also see the key-id.

In the examples above I enabled authentication per interface. It's also possible to do this for the entire area...this might save you some time if you have a router with many interfaces. You can do it like this:

```
Clubs(config-if)#router ospf 1
Clubs(config-router)#area 0 authentication
```

Or in case you want MD5 authentication:

```
Clubs(config-if)#router ospf 1
Clubs(config-router)#area 0 authentication message-digest
```

If you want you can change the OSPF timers so it responds faster to changes in the network. This is how you do it:

```
Hearts#show ip ospf interface fastEthernet 1/0
FastEthernet1/0 is up, line protocol is up
Internet Address 192.168.13.1/24, Area 0
Process ID 1, Router ID 192.168.13.1, Network Type BROADCAST, Cost: 1
Transmit Delay is 1 sec, State DR, Priority 1
Designated Router (ID) 192.168.13.1, Interface address 192.168.13.1
Backup Designated router (ID) 3.3.3.3, Interface address 192.168.13.3
Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
```

Above you see the default timers. Every 10 seconds an hello packet is sent and if we don't receive any hello packets for 40 seconds then we will declare our neighbor "dead".

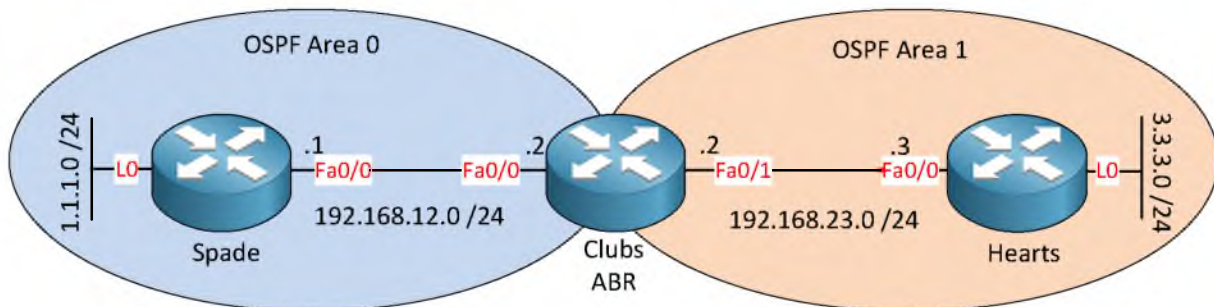
If we want we can change these tuners, I'll do this between router Spade and Clubs:

```
Spade(config-if)#interface fastEthernet 1/0
Spade(config-if)#ip ospf hello-interval 5
Spade(config-if)#ip ospf dead-interval 15
```

```
Clubs(config)#interface fastEthernet 0/0
Clubs(config-if)#ip ospf hello-interval 5
Clubs(config-if)#ip ospf dead-interval 15
```

Use the **ip ospf hello-interval** and **ip ospf dead-interval** to change these timers. As I explained in the beginning of this chapter, these values have to match on both ends!

I haven't shown you multi-area OSPF yet, this might sound exciting but it's really easy to configure. You only have to change the area number when you are configuring the network commands. Let me show you with the following topology:



In this topology, router Spade is in area 0 and router Hearts is in area 1. Router Clubs is in both areas and is the area border router. Router Spade and Hearts each have a loopback0 interface with a network that we will advertise in OSPF.

Let me show you the configuration:

```
Spade(config)#router ospf 1
Spade(config-router)#network 1.1.1.0 0.0.0.255 area 0
Spade(config-router)#network 192.168.12.0 0.0.0.255 area 0
```

Router Spade advertises its loopback0 interface and the FastEthernet 0/0 interface into area 0.

```
Clubs(config)#router ospf 1
Clubs(config-router)#network 192.168.12.0 0.0.0.255 area 0
Clubs(config-router)#network 192.168.23.0 0.0.0.255 area 1
```

Router Clubs advertises its FastEthernet 0/0 interface in area 0 and the FastEthernet 0/1 interface in area 1. This turns it into an ABR.

```
Hearts(config)#router ospf 1
Hearts(config-router)#network 3.3.3.0 0.0.0.255 area 1
Hearts(config-router)#network 192.168.23.0 0.0.0.255 area 1
```

And finally router Hearts advertises both its interfaces in area 1. Let's verify that we have neighbors:

```
Spade#show ip ospf neighbor
```

| Neighbor ID     | Pri | State    | Dead Time | Address      | Interface |
|-----------------|-----|----------|-----------|--------------|-----------|
| 192.168.23.2    | 1   | FULL/BDR | 00:00:38  | 192.168.12.2 |           |
| FastEthernet0/0 |     |          |           |              |           |

```
Clubs#show ip ospf neighbor
```

| Neighbor ID     | Pri | State    | Dead Time | Address      | Interface |
|-----------------|-----|----------|-----------|--------------|-----------|
| 1.1.1.1         | 1   | FULL/DR  | 00:00:36  | 192.168.12.1 |           |
| FastEthernet0/0 |     |          |           |              |           |
| 3.3.3.3         | 1   | FULL/BDR | 00:00:34  | 192.168.23.3 |           |
| FastEthernet0/1 |     |          |           |              |           |

```
Hearts#show ip ospf neighbor
```

| Neighbor ID     | Pri | State   | Dead Time | Address      | Interface |
|-----------------|-----|---------|-----------|--------------|-----------|
| 192.168.23.2    | 1   | FULL/DR | 00:00:38  | 192.168.23.2 |           |
| FastEthernet0/0 |     |         |           |              |           |

This is looking good, as you can see router Clubs has two OSPF neighbors. Now the big question, what do the routing tables look like?

```
Spade#show ip route ospf
 3.0.0.0/32 is subnetted, 1 subnets
O IA   3.3.3.3 [110/21] via 192.168.12.2, 00:04:24, FastEthernet0/0
O IA 192.168.23.0/24 [110/20] via 192.168.12.2, 00:05:36, FastEthernet0/0
```

Router Spade has learned two networks and if you take a close look you can see it says “**O IA**”. The **IA means Inter-Area** and these are the networks from another area.

```
Clubs#show ip route ospf
 1.0.0.0/32 is subnetted, 1 subnets
O      1.1.1.1 [110/11] via 192.168.12.1, 00:05:35, FastEthernet0/0
 3.0.0.0/32 is subnetted, 1 subnets
O      3.3.3.3 [110/11] via 192.168.23.3, 00:04:27, FastEthernet0/1
```

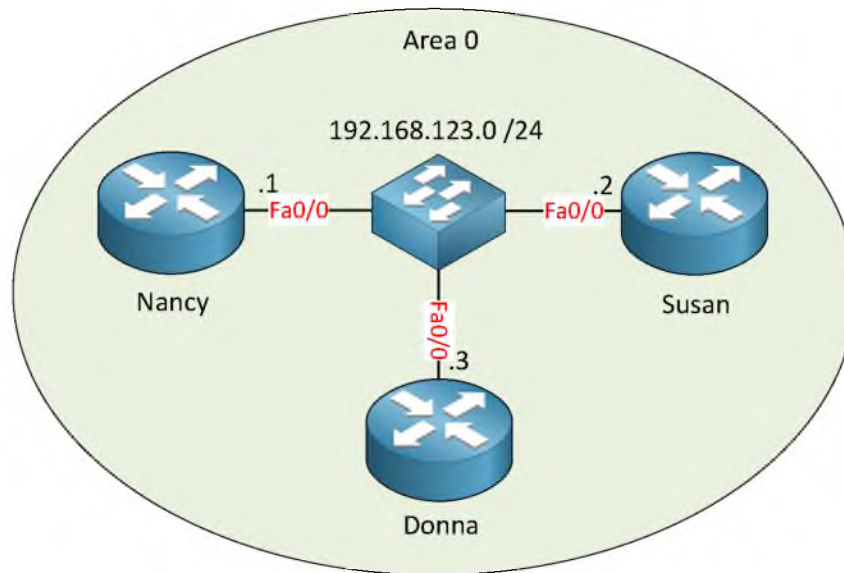
Router Clubs has learned about the loopback interfaces of router Spade and Hearts. As you can see these are normal “O” entries because router Clubs is in area 0 and area 1.

```
Hearts#show ip route ospf
O IA 192.168.12.0/24 [110/20] via 192.168.23.2, 00:04:31, FastEthernet0/0
 1.0.0.0/32 is subnetted, 1 subnets
O IA   1.1.1.1 [110/21] via 192.168.23.2, 00:04:31, FastEthernet0/0
```

Router Hearts looks similar to router Clubs. It has learned about the networks in area 0 and these show up as “O IA”.

That’s all there is to multi-area OSPF, just make sure you use the correct area number when you configure the network command!

Last but not least I want to show you how OSPF selects the DR and BDR. I will use a different topology to demonstrate this:



Here's an example of a network with 3 OSPF routers on a FastEthernet network. They are connected to the same switch (multi-access network) so there will be a DR/BDR election. OSPF has been configured so all routers have become OSPF neighbors, let's take a look:

```
Nancy#show ip ospf neighbor
```

| Neighbor ID   | Pri | State    | Dead Time | Address       | Interface       |
|---------------|-----|----------|-----------|---------------|-----------------|
| 192.168.123.2 | 1   | FULL/BDR | 00:00:32  | 192.168.123.2 | FastEthernet0/0 |
| 192.168.123.3 | 1   | FULL/DR  | 00:00:31  | 192.168.123.3 | FastEthernet0/0 |

From router Nancy's perspective, router Susan is the BDR and Donna is the DR.

```
Donna#show ip ospf neighbor
```

| Neighbor ID   | Pri | State        | Dead Time | Address       | Interface       |
|---------------|-----|--------------|-----------|---------------|-----------------|
| 192.168.123.1 | 1   | FULL/DROTHER | 00:00:36  | 192.168.123.1 | FastEthernet0/0 |
| 192.168.123.2 | 1   | FULL/BDR     | 00:00:39  | 192.168.123.2 | FastEthernet0/0 |

When a router is not the DR or BDR it's called a **DROTHER**. I have no idea if we have to pronounce it like "BROTHER with a D" or "DR-OTHER" ☺ Here we can see that router Nancy is a DROTHER.

```
Susan#show ip ospf neighbor
```

| Neighbor ID   | Pri | State        | Dead Time | Address       | Interface       |
|---------------|-----|--------------|-----------|---------------|-----------------|
| 192.168.123.1 | 1   | FULL/DROTHER | 00:00:31  | 192.168.123.1 | FastEthernet0/0 |
| 192.168.123.3 | 1   | FULL/DR      | 00:00:32  | 192.168.123.3 | FastEthernet0/0 |

And router Susan (the BDR) sees the DR and DROTHER.

Of course we can change which router becomes the DR/BDR by playing with the **priority**. Let's turn router Nancy in the DR:



```
Nancy(config)#interface fastEthernet 0/0
Nancy(config-if)#ip ospf priority 200
```

You change the priority if you like by using the **ip ospf priority** command:

- The default priority is 1.
- A priority of 0 means you will never be elected as DR or BDR.
- You need to use **clear ip ospf process** before this change takes effect.

```
Nancy#show ip ospf neighbor
```

| Neighbor ID          | Pri      | State          | Dead Time | Address       | Interface       |
|----------------------|----------|----------------|-----------|---------------|-----------------|
| 192.168.123.2        | 1        | FULL/BDR       | 00:00:31  | 192.168.123.2 | FastEthernet0/0 |
| <b>192.168.123.3</b> | <b>1</b> | <b>FULL/DR</b> | 00:00:32  | 192.168.123.3 | FastEthernet0/0 |

As you can see router Donna is still the DR, we need to reset the OSPF neighbor adjacencies so that we'll elect the new DR and BDR.

```
Donna#clear ip ospf process
Reset ALL OSPF processes? [no]: yes
Susan#clear ip ospf process
Reset ALL OSPF processes? [no]: yes
```

I'll reset all the OPSF neighbor adjacencies.

```
Nancy#show ip ospf neighbor
```

| Neighbor ID   | Pri | State        | Dead Time | Address       | Interface       |
|---------------|-----|--------------|-----------|---------------|-----------------|
| 192.168.123.2 | 1   | FULL/DROTHER | 00:00:36  | 192.168.123.2 | FastEthernet0/0 |
| 192.168.123.3 | 1   | FULL/BDR     | 00:00:30  | 192.168.123.3 | FastEthernet0/0 |

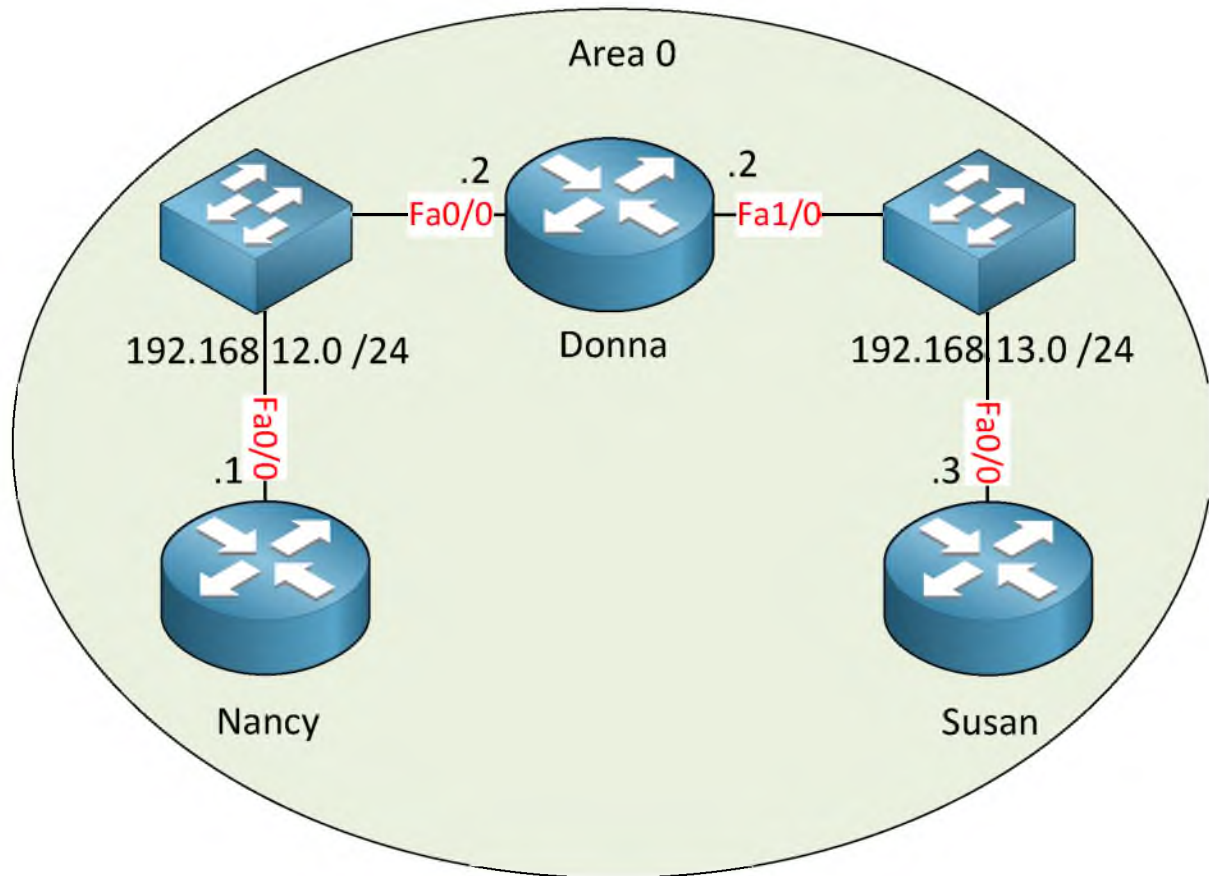
Now you can see router Nancy is the DR because the other routers are DROTHER and BDR.

```
Donna#show ip ospf neighbor
```

| Neighbor ID          | Pri        | State          | Dead Time | Address       | Interface       |
|----------------------|------------|----------------|-----------|---------------|-----------------|
| <b>192.168.123.1</b> | <b>200</b> | <b>FULL/DR</b> | 00:00:30  | 192.168.123.1 | FastEthernet0/0 |
| 192.168.123.2        | 1          | FULL/DROTHER   | 00:00:31  | 192.168.123.2 | FastEthernet0/0 |

Or we can confirm it from router Donna, you'll see that router Nancy is the DR and that the priority is 200. Something you need to be aware of is that the **DR/BDR election is per multi-access segment...not per area!**.

Let me give you an example:

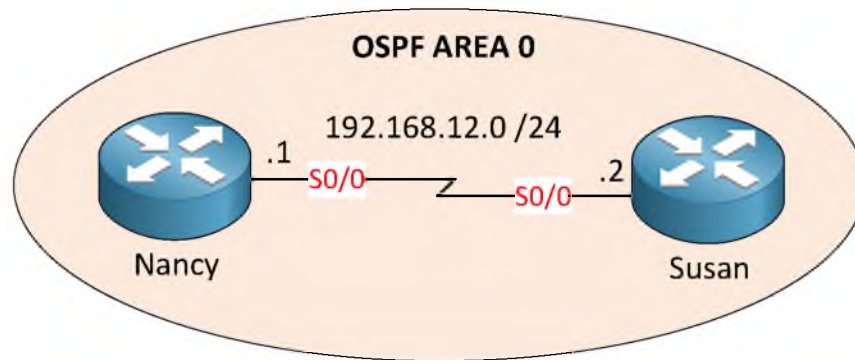


In the example above we have 2 multi-access segments. Between router Donna and Nancy, and between router Donna and Susan. For each segment there will be a DR/BDR election.

```
Donna#show ip ospf neighbor
```

| Neighbor ID  | Pri | State   | Dead Time | Address      | Interface       |
|--------------|-----|---------|-----------|--------------|-----------------|
| 192.168.13.3 | 200 | FULL/DR | 00:00:36  | 192.168.23.3 | FastEthernet1/0 |
| 192.168.12.1 | 200 | FULL/DR | 00:00:37  | 192.168.12.1 | FastEthernet0/0 |

In the example above you can see that router Nancy is the DR for the 192.168.12.0/24 segment and router Susan is the DR for the 192.168.13.0/24 segment.



```
Nancy#show ip ospf neighbor
```

| Neighbor ID  | Pri | State          | Dead Time | Address      | Interface |
|--------------|-----|----------------|-----------|--------------|-----------|
| 192.168.12.2 | 0   | <b>FULL/</b> - | 00:00:36  | 192.168.12.2 | Serial0/0 |

```
Susan#show ip ospf neighbor
```

| Neighbor ID  | Pri | State          | Dead Time | Address      | Interface |
|--------------|-----|----------------|-----------|--------------|-----------|
| 192.168.12.1 | 0   | <b>FULL/</b> - | 00:00:34  | 192.168.12.1 | Serial0/0 |

Here's an example of a point-to-point link. You can see that we have a neighbor but we didn't do an election for DR or BDR. Makes sense because there is always only one router on the other side.

And that's already the end of our OSPF chapter! See if you can configure some routers with OSPF using the commands that I just showed you. Play around with changing the cost, configuring authentication and tune the timers. You can also take a look at one of my labs that I created specifically for CCNA:

<http://gns3vault.com/OSPF/ospf-single-area.html>

## 17. EIGRP – Cisco’s Hybrid Routing Protocol

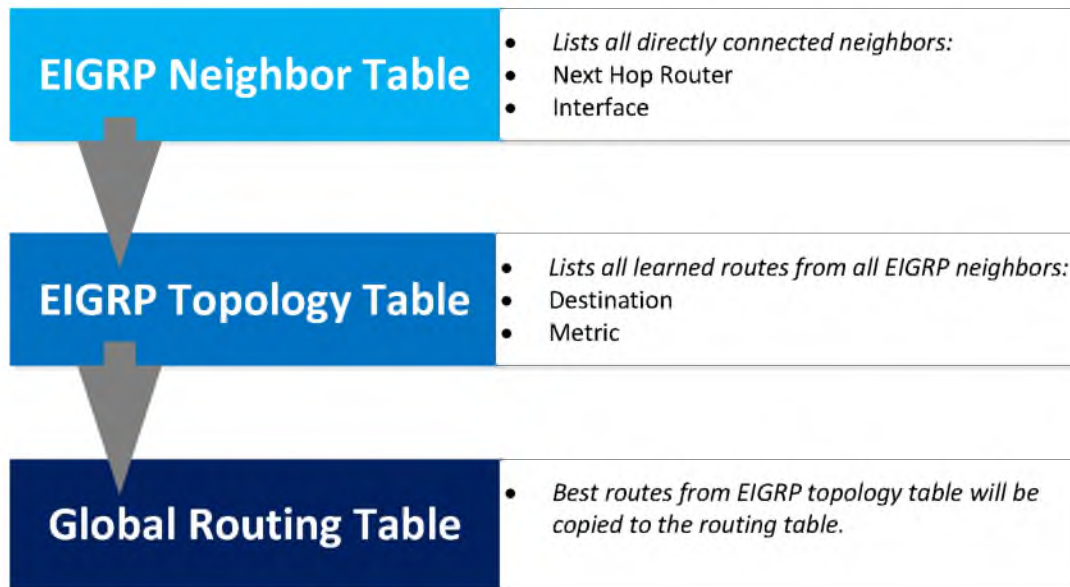
You have seen learned about distance vector routing protocols and have seen OSPF our link-state routing protocol. There is one more IGP to check which is **EIGRP**.

EIGRP stands for Enhanced Interior Gateway Routing Protocol and is a routing protocol created by Cisco. This means you can run it only on Cisco hardware, other vendors like Juniper don’t support it. EIGRP is called a hybrid or advanced distance vector protocol and most of the rules we have seen in the distance vector chapter also apply here:

- Split Horizon
- Route Poisoning
- Poison Reverse

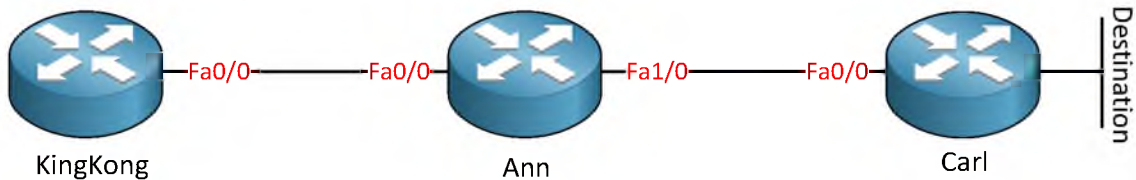
If you are unsure about these terms I’d highly recommend checking out the distance vector chapter again before you go continue with EIGRP.

Having said that let’s get started with EIGRP!



EIGRP routers will start sending hello packets to other routers just like OSPF does, if you send hello packets and you receive them you will become neighbors. EIGRP neighbors will exchange routing information which will be saved in the topology table. The best path from the topology table will be copied in the routing table.

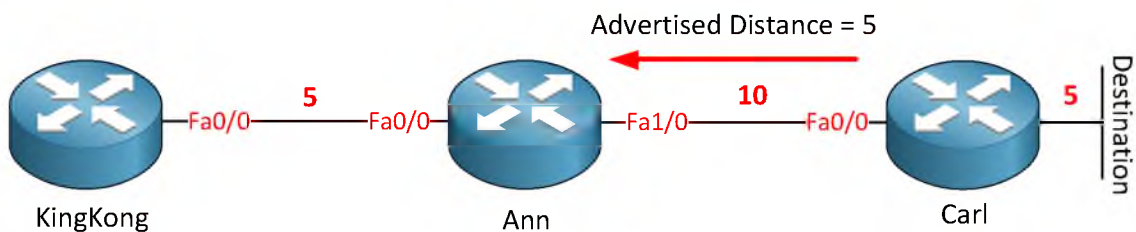
Selecting the best path with EIGRP works a bit different than other routing protocols so let’s see it in action:



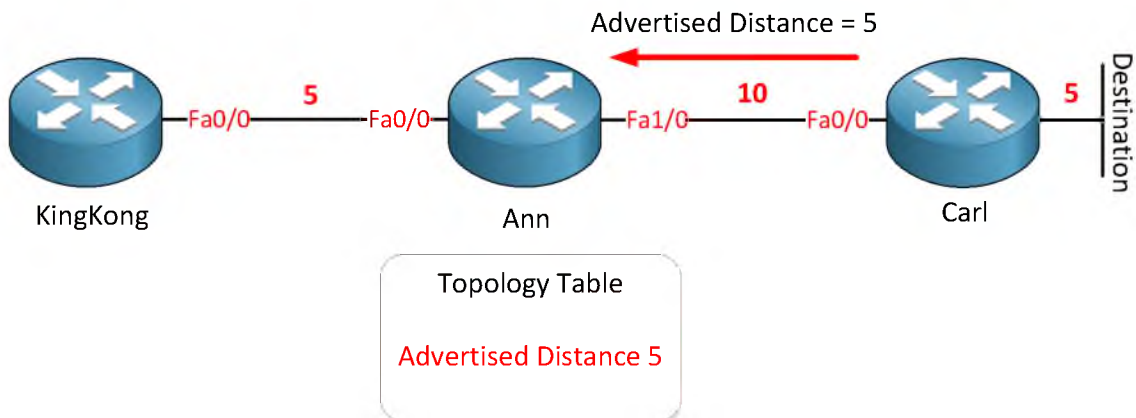
We have three routers named KingKong, Ann and Carl. We are going to calculate the best path to the destination which is behind router Carl.

EIGRP uses a rich set of metrics namely **bandwidth, delay, load and reliability** which we will cover later. These values will be put into a formula and each link will be assigned a metric. The lower these metrics the better.

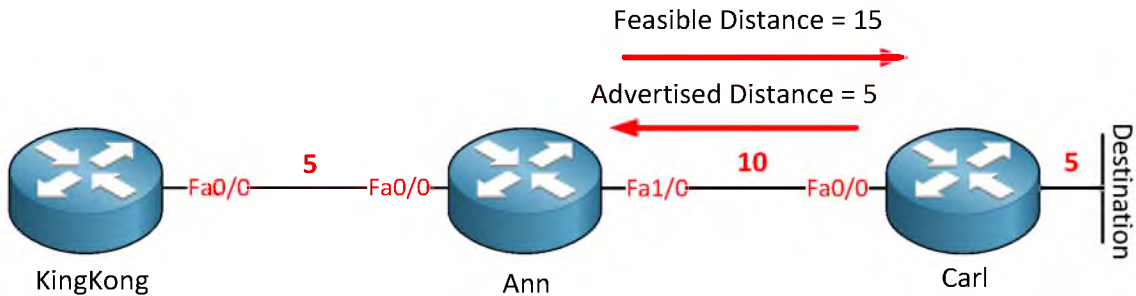
In the picture above I have assigned some values on the interfaces, if you would look on a real EIGRP router you'll see the numbers are very high and a bit annoying to work with.



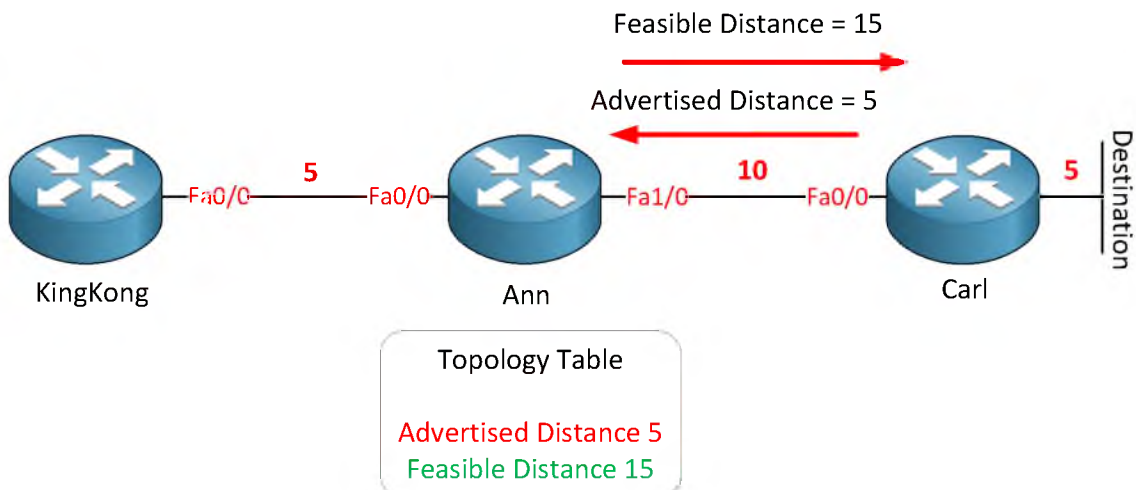
Router Carl will advertise to router Ann its metric towards the destination. Basically router Carl is saying to router Ann: "It costs me 5 to get there". This is called the **advertised distance**.



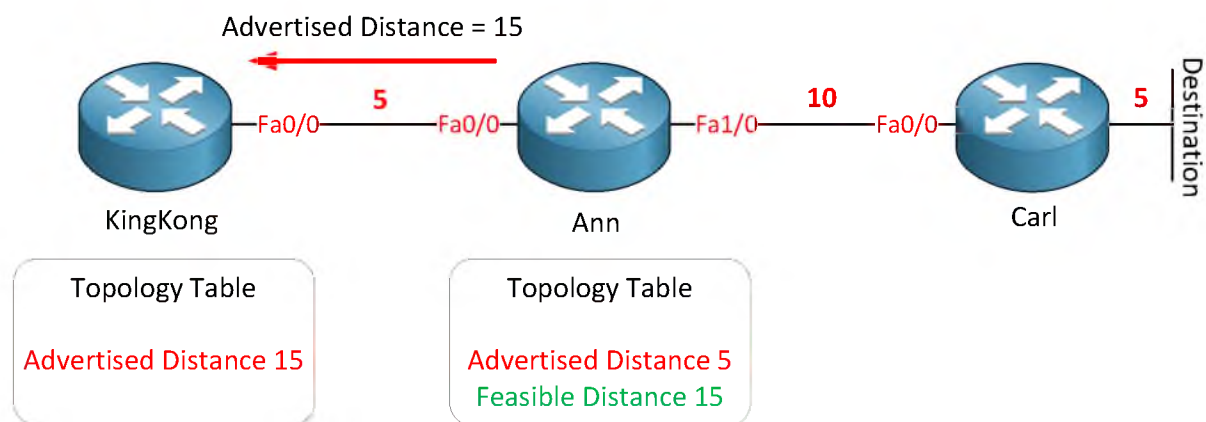
Router Ann has a topology table and in this topology table it will save this metric, the advertised distance to reach this destination is 5.



We are not done yet since there is something else that router Ann will save in its topology table. We know the advertised distance is 5 since this is what router Carl told us. We also know the metric of the link between router Ann and router Carl since this is directly connected. Router Ann now knows the metric for the total path to the destination, this total path is called the **feasible distance** and it will be saved in the topology table.

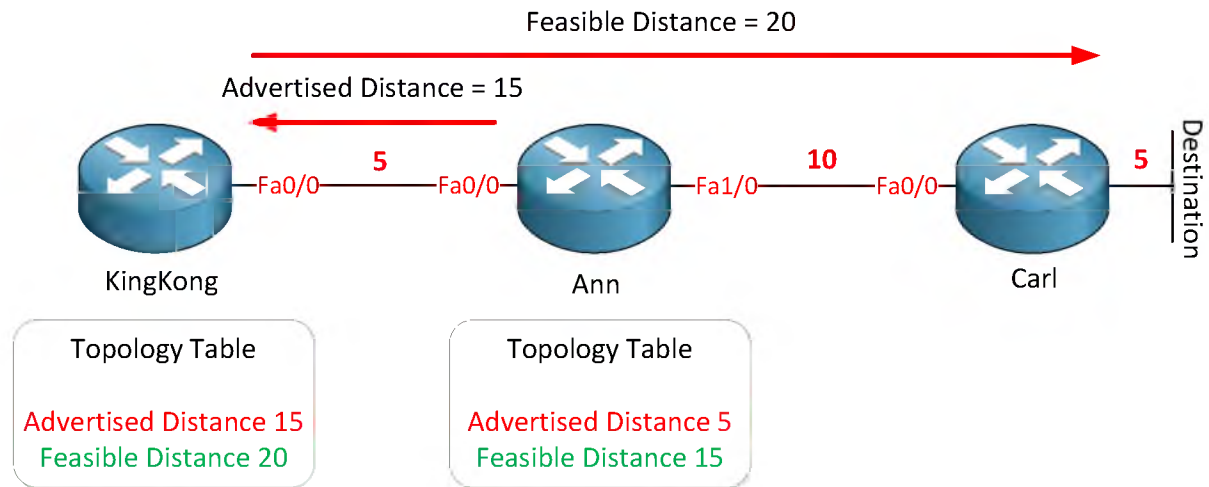


You have now learned two important concepts of EIGRP. The advertised distance, your neighbor tells you how far it is for him to reach the destination and the feasible distance which is your total distance to get to the destination.



We are not done yet since router KingKong is also running EIGRP. Router Ann is sending its feasible distance towards router KingKong which is 15. Router KingKong will save this

information in the topology table as the advertised distance. Router Ann is “telling” router KingKong the distance is 15.



Router KingKong now knows how far the destination is away for Router Ann and since we know the metric for the link between router KingKong and Ann it can also calculate the total distance which is called the feasible distance, this information is saved in the topology table.

Are you following me so far? Let me describe these terms once again but in plain English:

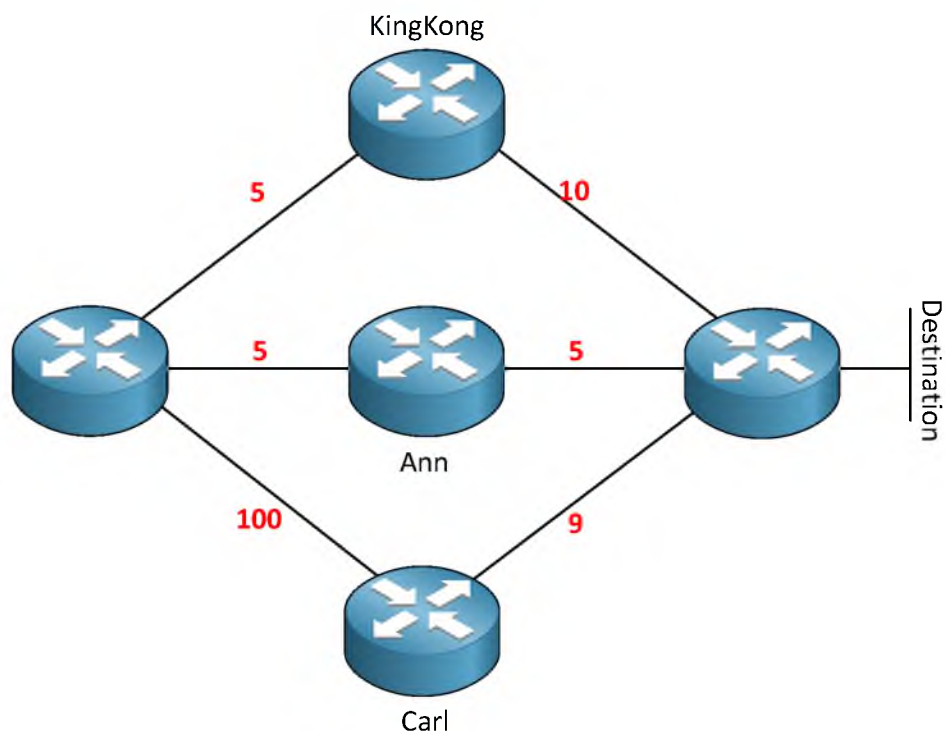
- Advertised distance: How far the destination is away for your neighbor.
- Feasible distance: The total distance to the destination.

## The best path to the destination is called the **successor**!

The successor will be copied from the topology table to the routing table.

With EIGRP however it's possible to have a backup path which we call the **feasible successor**. How do we find out if we have a feasible successor? Let's find out:





In the example above we have a couple of routers running EIGRP; we are sitting on the router without a name on the left side and would like to know two things:

- Which path is the successor (the best path)?
- Do we have any feasible successors? (backup paths)

Let's fill in the following table to find out:

|                 | Advertised Distance | Feasible distance |  |
|-----------------|---------------------|-------------------|--|
| <b>KingKong</b> |                     |                   |  |
| <b>Ann</b>      |                     |                   |  |
| <b>Carl</b>     |                     |                   |  |

If you want to try your new-learned EIGRP skills try to fill in the advertised and feasible distance by yourself in the table above.

Router KingKong is telling us the destination is 10 away, router Ann tells us its 5 away and router Carl tells us its 9 away. We can now fill in the advertised distance part of the table:

|                 | Advertised Distance | Feasible distance |  |
|-----------------|---------------------|-------------------|--|
| <b>KingKong</b> | 10                  |                   |  |
| <b>Ann</b>      | 5                   |                   |  |
| <b>Carl</b>     | 9                   |                   |  |



Since we know our directly connected links we can add this to the advertised distance and we'll have our feasible distance.

|                 | Advertised Distance | Feasible distance |  |
|-----------------|---------------------|-------------------|--|
| <b>KingKong</b> | 10                  | 15                |  |
| <b>Ann</b>      | 5                   | 10                |  |
| <b>Carl</b>     | 9                   | 109               |  |

The path with the lowest feasible distance will be the successor (router Ann) so now we answered the first question.

|                 | Advertised Distance | Feasible distance |           |
|-----------------|---------------------|-------------------|-----------|
| <b>KingKong</b> | 10                  | 15                |           |
| <b>Ann</b>      | 5                   | 10                | SUCCESSOR |
| <b>Carl</b>     | 9                   | 109               |           |

You will find the successor in the routing table.

To answer the second question "do we have a feasible successor (backup path)?" we need to learn another formula:

**Advertised distance of feasible successor < Feasible distance of successor.**

This is where I get to see glazed eyes and flabbergasted students so let's do it in plain English one more time:

A router can become a backup path if he is closer to the destination than the total distance of your best path.

I think that sounds a bit better right? Let's try it and see if router KingKong or router Carl is suitable as a backup path:

The advertised distance of router KingKong is 10 which is equal to the feasible distance of router Ann which is also 10. It has to be lower...equal is not good enough so router KingKong will NOT be a feasible successor.

The advertised distance of router Carl is 9 which is lower than the feasible distance of router Ann which is 10. Router Carl will be a valid feasible successor and used as a backup path!

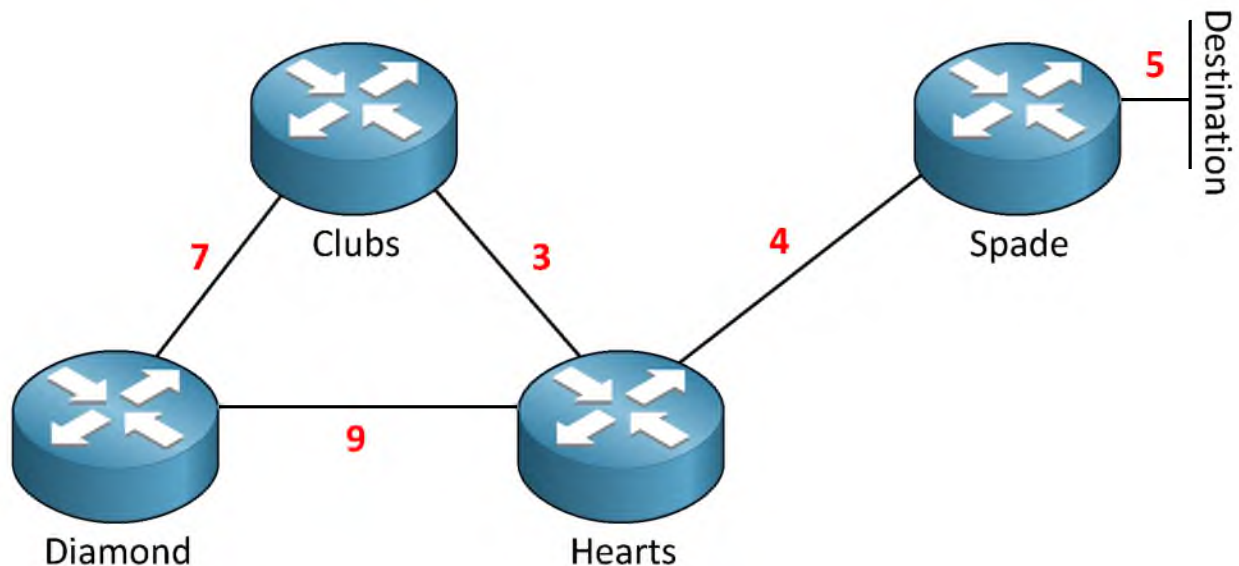
|                 | Advertised Distance | Feasible distance |                       |
|-----------------|---------------------|-------------------|-----------------------|
| <b>KingKong</b> | 10                  | 15                |                       |
| <b>Ann</b>      | 5                   | 10                | SUCCESSOR             |
| <b>Carl</b>     | 9                   | 109               | FEASIBLE<br>SUCCESSOR |

Excellent so router Ann is our successor and router Carl is a feasible successor. You will find both entries in the EIGRP topology but you will only find the successor in the routing table. If you lose the successor because of a link failure EIGRP will copy/paste the feasible successor in the routing table. This is what makes EIGRP a FAST routing table...but only if you have feasible successor in the routing table.

Now look closely to the feasible distance of router Carl and router KingKong...what do you see? The metric for router Carl is FAR worse than the one for router KingKong. Does this make any sense? Did the Cisco EIGRP engineers make a horrible mistake here by using non-optimal backup paths?

Nope this is perfectly the way it should be! Keep in mind EIGRP at heart is a distance vector protocol. It doesn't know what the complete network looks like...it's not a link-state routing protocol like OSPF which DOES have a complete map of the network. Distance vector routing protocols only know which way to go (vector) and how far away the destination is (distance).

The formula you just witnessed to determine EIGRP feasible successors is how EIGRP can guarantee you that the backup path is 100% loop-free! I know this is difficult to grasp by reading text so let's do another example:



Keep in mind that EIGRP is by nature a distance vector routing protocol, we see the topology but EIGRP does not!

We are looking at the EIGRP topology table of router Hearts and we want to reach the destination behind router Spade, let's fill in the table (try it yourself if you want a good exercise):

|                | Advertised Distance | Feasible distance |  |
|----------------|---------------------|-------------------|--|
| <b>Spade</b>   |                     |                   |  |
| <b>Clubs</b>   |                     |                   |  |
| <b>Diamond</b> |                     |                   |  |

1. Router Spade will advertise the destination network to router Hearts.
2. Router Hearts will advertise the network to router Clubs and Diamond.
3. Router Clubs will advertise the network to router Diamond.
4. Router Diamond will advertise the network to router Clubs.
5. Router Clubs will advertise this network back to router Hearts.
6. Router Diamond will advertise this network back to router Hearts.

|                | Advertised Distance | Feasible distance |  |
|----------------|---------------------|-------------------|--|
| <b>Spade</b>   | 5                   |                   |  |
| <b>Clubs</b>   | 25                  |                   |  |
| <b>Diamond</b> | 19                  |                   |  |

Here we have the advertised distance, our neighbors are telling us how far it is for them to reach the destination network. Next step is to fill in the feasible successors.

How did I get the numbers in the advertised distance table? Let's look at all the routers:

Router Spade is easy. The destination has a distance of 5 as seen in the topology picture. This will be advertised to router Hearts and placed in its topology table.

Router Clubs will learn the destination network through router Hearts and router Diamond. Router Hearts will advertise a distance of  $5+4 = 9$  to router Clubs. So why didn't I place "9" in the advertised distance field in the table? Good question! Remember split-horizon? Don't advertise to your neighbor whatever you learned from them....router Clubs is not sending information about this network back to router Hearts. To be more specific: **whatever you learn on an interface you don't advertise back out of the same interface.**

How did I get to 25? Let's break it down:

Router Spade will advertise a distance of 5 towards router Hearts. Router Hearts will advertise  $5+4 = 9$  towards router Diamond.

Router Diamond will advertise  $5+4+9 = 18$  towards router Clubs. Finally router Clubs will advertise  $5+4+9+7 = 25$  towards router Hearts. Split Horizon doesn't apply here since router Clubs learned about the destination on another interface (router Diamond).

The same thing applies for the advertised distance of 19 for router Diamond:

1. Router Spade advertises a distance of 5 to router Hearts.
2. Router Hearts advertises a distance of  $5+4 = 9$  to router Clubs.
3. Router Clubs advertises a distance of  $5+4+3 = 12$  to router Diamond.
4. Router Diamond advertises a distance of  $5+4+3+7 = 19$  to router Hearts.

|                | Advertised Distance | Feasible distance |  |
|----------------|---------------------|-------------------|--|
| <b>Spade</b>   | 5                   | 9                 |  |
| <b>Clubs</b>   | 25                  | 28                |  |
| <b>Diamond</b> | 19                  | 28                |  |

Router Hearts has learned the advertised distance from its neighbors and knows about its own directly connected interfaces so you can fill in the feasible distance. Last step is to pick our successor.

|                | Advertised Distance | Feasible distance |           |
|----------------|---------------------|-------------------|-----------|
| <b>Spade</b>   | 5                   | 9                 | SUCCESSOR |
| <b>Clubs</b>   | 25                  | 28                |           |
| <b>Diamond</b> | 19                  | 28                |           |

Router Spade has the lowest feasible distance so it will become the successor...excellent! Let's do the feasible successor check and see if there is a backup path:

Advertised distance of feasible successor < Feasible distance of successor.

The advertised distance of router Clubs (25) and Diamond (19) are higher than the feasible distance of router Spade (9) so they won't become feasible successors. This makes sense right? If these routers become backup paths we would have a loop!

If your neighbor is closer to the destination than your total path you at least know it's not getting to the destination by sending packets **through your router**. Perhaps it's not the best path but it's absolutely 100% loop-free!

The next thing I want to show you are the EIGRP metrics. You have learned that OSPF uses cost and EIGRP has yet something else for metrics...actually 4 things:

- Bandwidth
- Delay
- Load
- Reliability

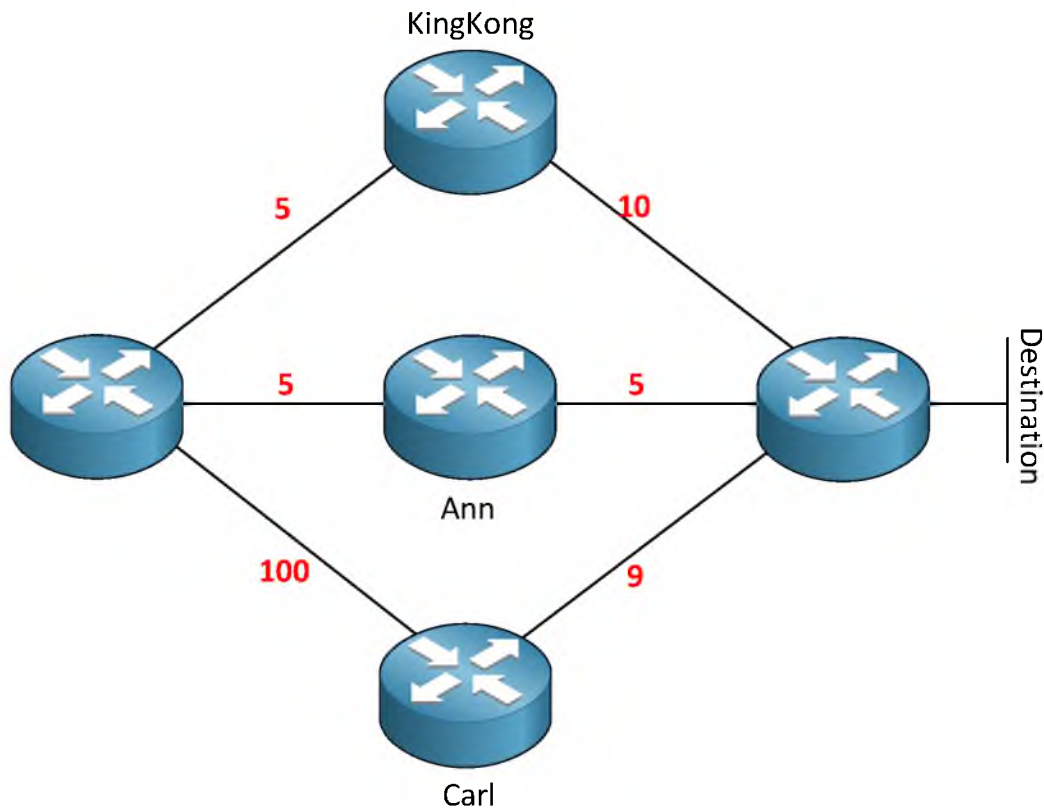
Bandwidth and Delay are static values; a FastEthernet link is 100 Mbit and has a delay of 100usec. Ethernet for example is 10 Mbit and has a delay of 1000usec. These values don't change.

Load is how busy your link is and reliability is how reliable your link is by looking at errors. Load and reliability are values that are dynamic because they can change over time.

By default EIGRP only uses bandwidth and delay. Load and reliability are disabled by default. This makes sense because we want routing protocols to be as quiet and reliable as can be.

You don't want EIGRP routers sending updates every now and then because a link is suddenly busy. Routing should be predictable and after picking the best path you want EIGRP only to send hello packets to maintain neighbors...nice and quiet.

EIGRP has another trick in its hat. OSPF can do load-balancing but the paths have to be equal. EIGRP can do something cool...unequal load-balancing! Even better it will share traffic in a proportional way, if you have a feasible successor that has a feasible distance which is 5 times worse than the successor traffic will be shared in a 5:1 way.



|                 | Advertised Distance | Feasible distance |                       |
|-----------------|---------------------|-------------------|-----------------------|
| <b>KingKong</b> | 10                  | 15                |                       |
| <b>Ann</b>      | 5                   | 10                | SUCCESSOR             |
| <b>Carl</b>     | 9                   | 109               | FEASIBLE<br>SUCCESSOR |

This is our first example where we found out the successor and feasible successor. If you look at the routing table you will only find the successor there. Now we are going to change things so we'll see the feasible successor in the routing table as well so it will load-balance.

You can do this by using the **variance** command. The variance command works as a multiplier:

- Our successor has a feasible distance of 10.
- Our feasible successor has a feasible distance of 109.

In order to load-balance our feasible successor needs to have a lower feasible distance than the successor X multiplier.

If we set the variance at 2, this is what we get:

Feasible distance of successor is  $10 \times 2$  (multiplier) = 20.

109 is higher than 20 so we don't do any load-balancing.

If we set the variance at 5, this is what we get:

Feasible distance of successor is  $10 \times 5$  (multiplier) = 50.

109 is still higher than 50 so still no load-balancing here.

Now I'm going to set the variance at 11 and this is what we get:

Feasible distance of successor is  $10 \times 11 = 110$ .

109 is lower than 110 so now we will put the feasible successor in the routing table and start load-balancing!

Are we ever going to use the route through router KingKong? No we won't since it's not a feasible successor!

We are almost done with EIGRP there is one more thing. EIGRP can do authentication but it works different than OSPF since we need to configure a key-chain and it only supports MD5.

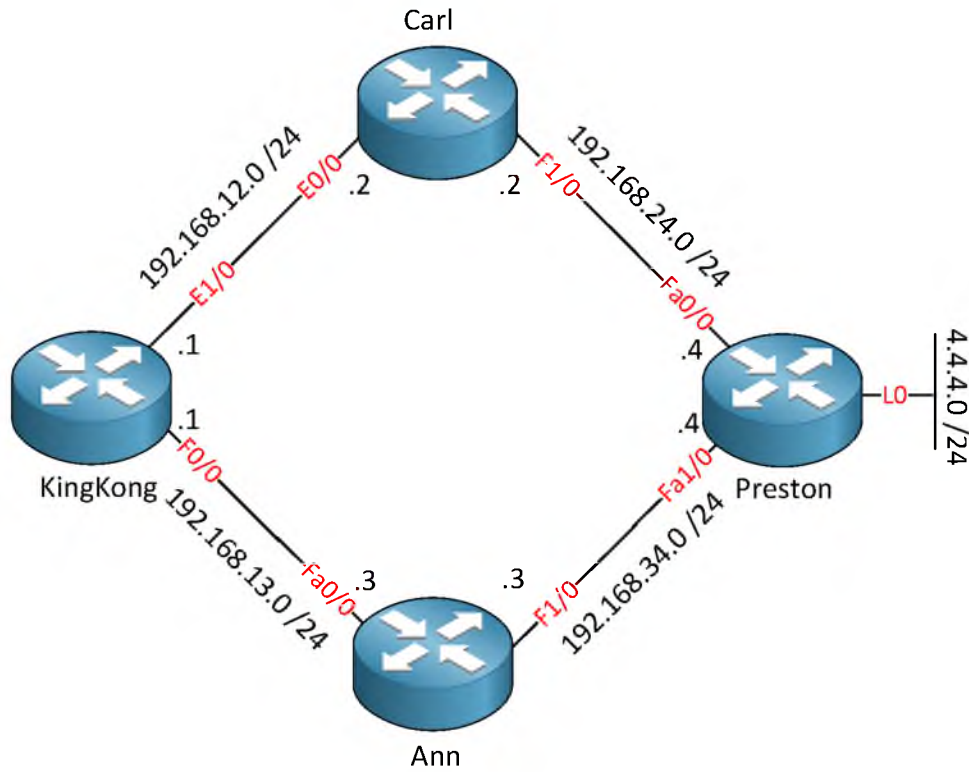


First you need to configure a key-chain and you can give it any name you want. Every key-chain can have a bunch of keys; each key will have a key-ID.

- The Key-chain can be **different** per router.
- The key-ID has to be **the same** on every router.

Finally you need to set a key-string per key-ID which is the password; this of course has to be the same. If you like you can also configure time duration for the different keys.

Now you have an idea how EIGRP works, let's walk through all the commands together on some routers:



In the topology above I have 4 routers. All interfaces are FastEthernet with the exception of the link between router KingKong and Carl, that's where we use an Ethernet link. Behind router Preston there is a loopback interface.

Let's start by configuring EIGRP between router KingKong and Ann:

```

KingKong(config)#router eigrp 1
KingKong(config-router)#no auto-summary
KingKong(config-router)#network 192.168.13.0
  
```

```

Ann(config)#router eigrp 1
Ann(config-router)#no auto-summary
Ann(config-router)#network 192.168.13.0
  
```

The "1" is the AS number and it has to be **the same** on all routers! We require the **no auto-summary** command because on most IOS versions EIGRP will behave classful by default and we want it to be classless.

After typing in these commands this is what you will see:

```
KingKong#
%DUAL-5-NBRCHANGE: IP-EIGRP(0) 1: Neighbor 192.168.13.3 (FastEthernet0/0) is
up: new adjacency
```

```
Ann#
%DUAL-5-NBRCHANGE: IP-EIGRP(0) 1: Neighbor 192.168.13.1 (FastEthernet0/0) is
up: new adjacency
```

Our routers have become EIGRP neighbors.

We can also verify this with a command:

```
KingKong#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                Interface      Hold Uptime    SRTT    RTO  Q  Seq
                               (sec)          (ms)        Cnt  Num
0   192.168.13.3            Fa0/0         12 00:11:58 1275   5000  0   3
```

```
Ann#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                Interface      Hold Uptime    SRTT    RTO  Q  Seq
                               (sec)          (ms)        Cnt  Num
0   192.168.13.1            Fa0/0         14 00:11:47   15    200  0   3
```

Use **show ip eigrp neighbors** to verify that we have a working EIGRP neighbor adjacency. This seems to be the case.

Let's configure all the network commands so all routers become EIGRP neighbors and advertise their networks:

```
KingKong(config)#router eigrp 1
KingKong(config-router)#network 192.168.12.0
```

```
Ann(config)#router eigrp 1
Ann(config-router)#network 192.168.34.0
```

```
Carl(config)#router eigrp 1
Carl(config-router)#no auto-summary
Carl(config-router)#network 192.168.12.0
Carl(config-router)#network 192.168.24.0
```

```
Preston(config)#router eigrp 1
Preston(config-router)#no auto-summary
Preston(config-router)#network 192.168.24.0
Preston(config-router)#network 192.168.34.0
Preston(config-router)#network 4.0.0.0
```

These network commands will activate EIGRP on all interfaces and advertise all networks that we have. Let's verify our work:



```
KingKong#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                Interface      Hold Uptime    SRTT    RTO  Q  Seq
                               (sec)          (ms)          Cnt  Num
1   192.168.12.2            Et1/0         14 00:20:08    12     200  0  15
0   192.168.13.3            Fa0/0         11 00:43:34    428    2568  0  12
```

```
Carl#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                Interface      Hold Uptime    SRTT    RTO  Q  Seq
                               (sec)          (ms)          Cnt  Num
1   192.168.24.4            Fa1/0         12 00:19:22    12     200  0  14
0   192.168.12.1            Et0/0         13 00:20:17    10     200  0  15
```

```
Ann#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                Interface      Hold Uptime    SRTT    RTO  Q  Seq
                               (sec)          (ms)          Cnt  Num
1   192.168.34.4            Fa1/0         14 00:19:29     8     200  0  13
0   192.168.13.1            Fa0/0         12 00:43:53     9     200  0  14
```

```
Preston#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                Interface      Hold Uptime    SRTT    RTO  Q  Seq
                               (sec)          (ms)          Cnt  Num
1   192.168.34.3            Fa1/0         12 00:19:39    658    3948  0  13
0   192.168.24.2            Fa0/0         14 00:19:42    537    3222  0  16
```

Each router has two EIGRP neighbors so this is looking good. Now let me show you the routing table of router KingKong:

```
KingKong#show ip route eigrp
4.0.0.0/24 is subnetted, 1 subnets
D      4.4.4.0 [90/158720] via 192.168.13.3, 00:21:48, FastEthernet0/0
D      192.168.24.0/24 [90/33280] via 192.168.13.3, 00:21:53, FastEthernet0/0
D      192.168.34.0/24 [90/30720] via 192.168.13.3, 00:21:50, FastEthernet0/0
```

The first thing you might notice is that you see a "D" for the EIGRP entries. The reason that you see a "D" and not an "E" is that the last one is already taken for EGP, an old routing protocol that we don't use anymore. "D" stands for "dual" which is the mechanism behind EIGRP. Let's take a closer look at one of these entries:

```
D      4.4.4.0 [90/158720] via 192.168.13.3, 00:21:48, FastEthernet0/0
```

What exactly do we see here?

- 4.4.4.0 is the network that we have learned.
- 90 is the administrative distance for EIGRP.
- 158720 is the metric, this is the feasible distance that you see.
- Via 192.168.13.3 is the next hop IP address. In this case it's router Ann.

The metric that EIGRP uses isn't as easy as OSPF; the numbers to work with are very large.

Why did router KingKong decide to use the link through router Ann in order to get to network 4.4.4.0 /24? The answer is in the EIGRP topology table:

```
KingKong#show ip eigrp topology
IP-EIGRP Topology Table for AS(1)/ID(192.168.12.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
       r - reply Status, s - sia Status

P 4.4.4.0/24, 1 successors, FD is 158720
    via 192.168.13.3 (158720/156160), FastEthernet0/0
    via 192.168.12.2 (412160/156160), Ethernet1/0
```

Use **show ip eigrp topology** to see the EIGRP topology table. This is an important command so let me describe what we have here...

```
P 4.4.4.0/24, 1 successors, FD is 158720
```

The first line tells us that we have a successor. We already knew this because we saw it in the routing table. FD is the feasible distance to reach 4.4.4.0 /24.

```
via 192.168.13.3 (158720/156160), FastEthernet0/0
```

Above you see two important numbers:

- 158720 is the feasible distance.
- 156160 is the advertised distance.

```
via 192.168.12.2 (412160/156160), Ethernet1/0
```

Above you see the information for the feasible successor (router Carl):

- 412160 is the feasible distance.
- 156160 is the advertised distance.

The reason that this is a feasible successor is because the advertised distance (156160) is lower than the feasible distance of the successor (158720)

So to summarize what we have seen:

- Router KingKong knows how to reach network 4.4.4.0 /24.
- The path through router Ann is the successor because the feasible distance is 158720.
- The path through router Carl is the feasible successor because the advertised distance (156160) is lower than the feasible distance of the successor (158720).

Right now only the successor is in the routing table but we activate load balancing so that the feasible successor will also be used.

This is how we do it:

```
KingKong#show ip eigrp topology
IP-EIGRP Topology Table for AS(1)/ID(192.168.12.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
       r - reply Status, s - sia Status

P 4.4.4.0/24, 1 successors, FD is 158720
    via 192.168.13.3 (158720/156160), FastEthernet0/0
    via 192.168.12.2 (412160/156160), Ethernet1/0
```

The feasible successor will be used if its feasible distance is lower than the feasible distance multiplied with a multiplier.

$158720 \times 3 = 476160 < 412160$ .

```
KingKong(config)#router eigrp 1
KingKong(config-router)#variance 3
```

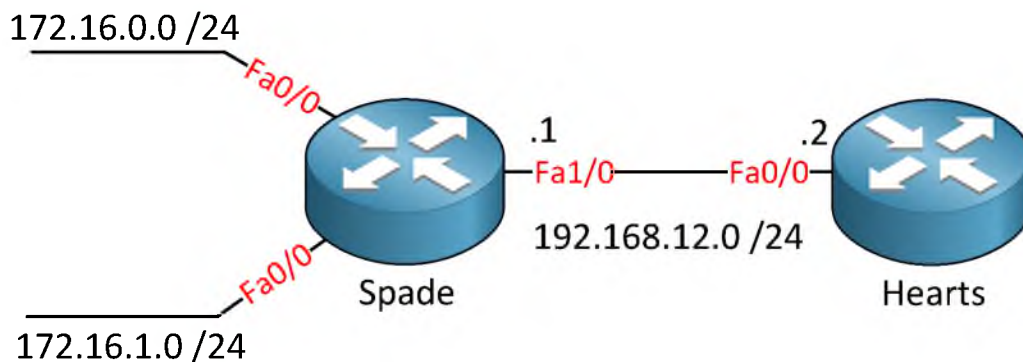
I need to use the **variance** command to set this multiplier.

Let's check our routing table now:

```
KingKong#show ip route eigrp
 4.0.0.0/24 is subnetted, 1 subnets
D    4.4.4.0 [90/158720] via 192.168.13.3, 00:00:37, FastEthernet0/0
      [90/412160] via 192.168.12.2, 00:00:37, Ethernet1/0
```

Great! Now we see both entries in the routing table. You have now seen how EIGRP selects the successor, feasible successor and how to enable load balancing over feasible successors.

EIGRP can also do summarization, let me give you an example:



Let me show you the EIGRP configuration without summarization first:

```
Spade(config)#router eigrp 1
Spade(config-router)#no auto-summary
Spade(config-router)#network 192.168.12.0
Spade(config-router)#network 172.16.0.0
```

```
Hearts(config)#router eigrp 1
Hearts(config-router)#no auto-summary
Hearts(config-router)#network 192.168.12.0
```

Just a basic EIGRP configuration. This is what the routing table of router Hearts looks like:

```
Hearts#show ip route eigrp
      172.16.0.0/24 is subnetted, 2 subnets
D       172.16.0.0 [90/30720] via 192.168.12.1, 00:02:28, FastEthernet0/0
D       172.16.1.0 [90/30720] via 192.168.12.1, 00:02:28, FastEthernet0/0
```

Two entries as expected. Now let's create that summary:

```
Spade(config)#interface fastEthernet 2/0
Spade(config-if)#ip summary-address eigrp 1 172.16.0.0 255.255.254.0
```

Use the **ip summary-address eigrp** command to specify the summary.

Now the routing table will look like this:

```
Hearts#show ip route eigrp
      172.16.0.0/23 is subnetted, 1 subnets
D       172.16.0.0 [90/30720] via 192.168.12.1, 00:00:53, FastEthernet0/0
```

The two networks disappear and a single entry will appear on router Hearts. Something will change on router Spade as well:

```
Spade#show ip route eigrp
      172.16.0.0/16 is variably subnetted, 3 subnets, 2 masks
D       172.16.0.0/23 is a summary, 00:01:38, Null0
```

EIGRP will create an entry in the routing table for the summary and it's pointing to the null0 interface. It does this to protect itself against routing loops. When you use summaries it's possible that other routers will send traffic to you for networks that you have no clue about where they are. When this happens the traffic will be forwarded to the null0 interface and dropped.

Next I'd like to show you how to configure authentication for EIGRP. Let's do this between router KingKong and Ann:

```
KingKong(config)#key chain MYCHAIN
KingKong(config-keychain)#key 1
KingKong(config-keychain-key)#key-string BANANA
```

```
Ann(config)#key chain MYCHAIN
Ann(config-keychain)#key 1
Ann(config-keychain-key)#key-string BANANA
```

First we create the key chain. Secondly we choose a key number and assign the password to it. Now we have to activate it:

```
KingKong(config)#interface fastEthernet 0/0
KingKong(config-if)#ip authentication mode eigrp 1 md5
KingKong(config-if)#ip authentication key-chain eigrp 1 MYCHAIN
```

```
Ann(config)#interface fastEthernet 0/0
Ann(config-if)#ip authentication mode eigrp 1 md5
Ann(config-if)#ip authentication key-chain eigrp 1 MYCHAIN
```

Use the **ip authentication mode eigrp** command to enable MD5 authentication and **ip authentication key-chain eigrp** to select the correct key chain.

That's it for EIGRP! If you want some practical experience on EIGRP there are a couple of labs you can try now:

<http://gns3vault.com/EIGRP/eigrp-beginner.html>

The EIGRP beginner lab will teach you all the basics for EIGRP.

<http://gns3vault.com/EIGRP/eigrp-md5-authentication.html>

This lab will teach you how to configure the key-chain and different keys for EIGRP.

<http://gns3vault.com/EIGRP/eigrp-intermediate.html>

If you can beat this lab by yourself you'll have plenty of knowledge about EIGRP for the CCNA exam.

## 18. Security: Keeping the bad guys out.

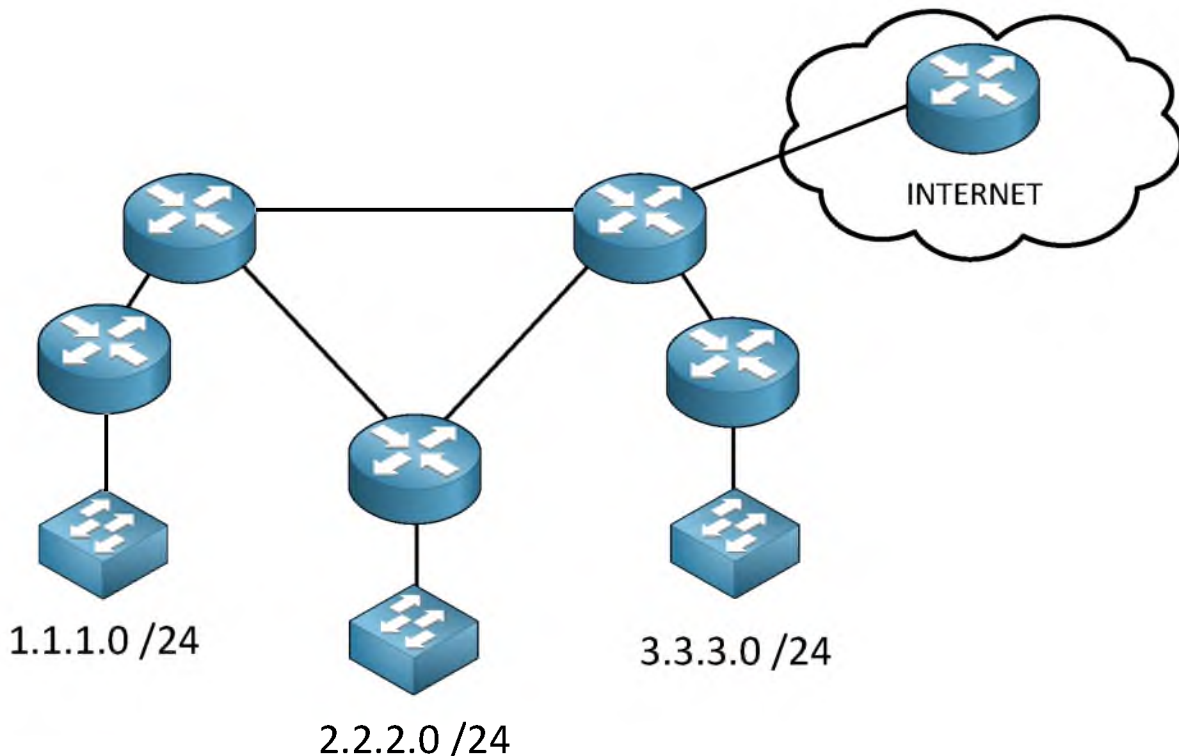
In a perfect world where we can trust anyone and nobody makes a mistake we don't need security. In real life however bad things happen to our network so we'll need to protect it.

By default all IP packets on a router will be routed, there are no restrictions. In the first part of this chapter we will start by looking at access-lists to see how we can block certain IP packets from entering or leaving our routers. In the second part of this chapter we will look at VPNs to see how we can encrypt and authenticate our traffic.

Access-lists work on the network (layer 3) and the transport (layer 4) layer and can be used for two different things:

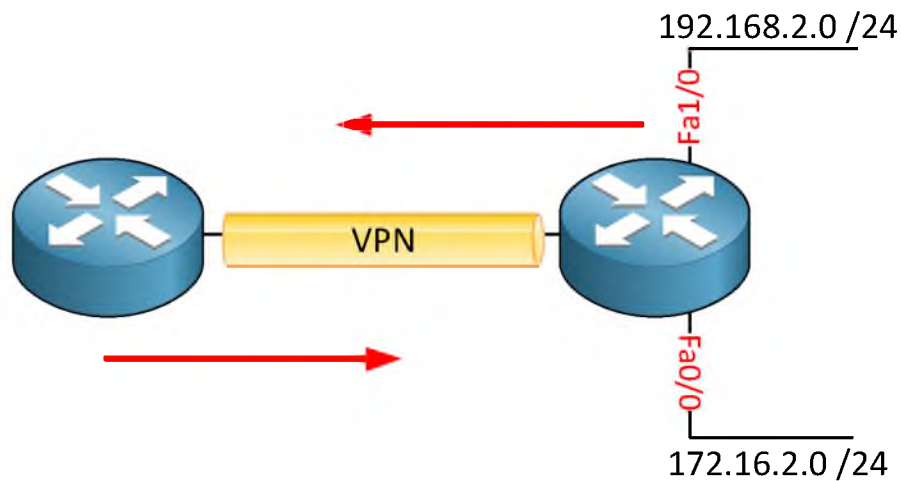
- **Filtering**
- **Classification**

Let's start with filtering:



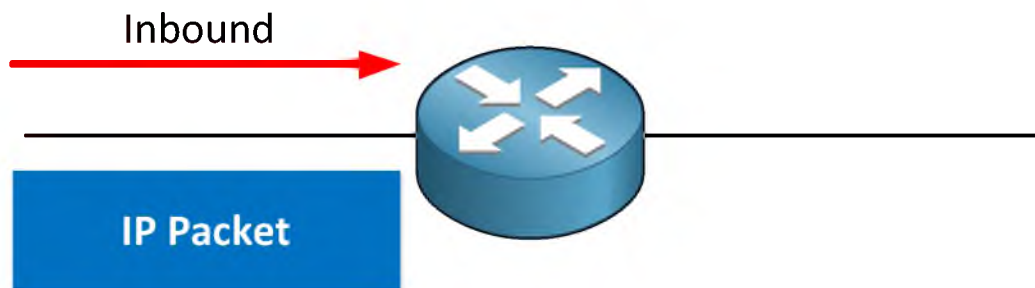
**Filtering** is used to permit or deny traffic reaching certain parts of our network. Without filtering traffic can go anywhere, if you look at the picture above you probably don't want IP packets from the internet to freely enter your network. You can also use an access-list to block IP packets from 3.3.3.0 /24 going to 1.1.1.0 /24 or something else.

**Classification** does not drop IP packets like filtering does but we use it to “select” traffic. Let’s take a look at an example:

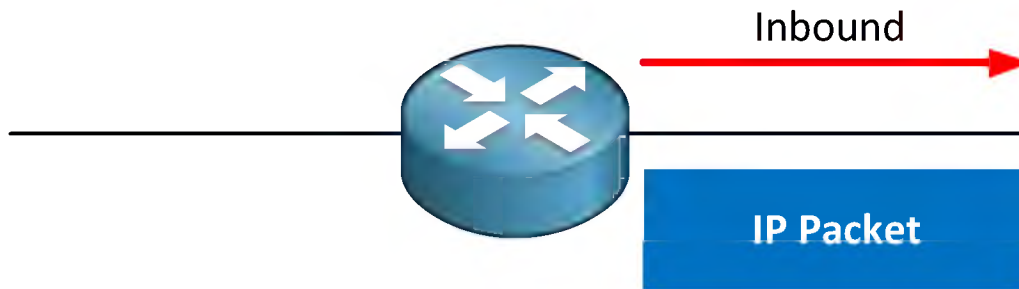


In the picture above we have a VPN that encrypts traffic between the two routers. Whenever we create a VPN we can use an access-list to “select” what traffic should be encrypted. Perhaps I want traffic from network 192.168.2.0 /24 to be encrypted but traffic from 172.16.2.0 /24 not. We can use an access-list to “select” traffic, this is called classification.

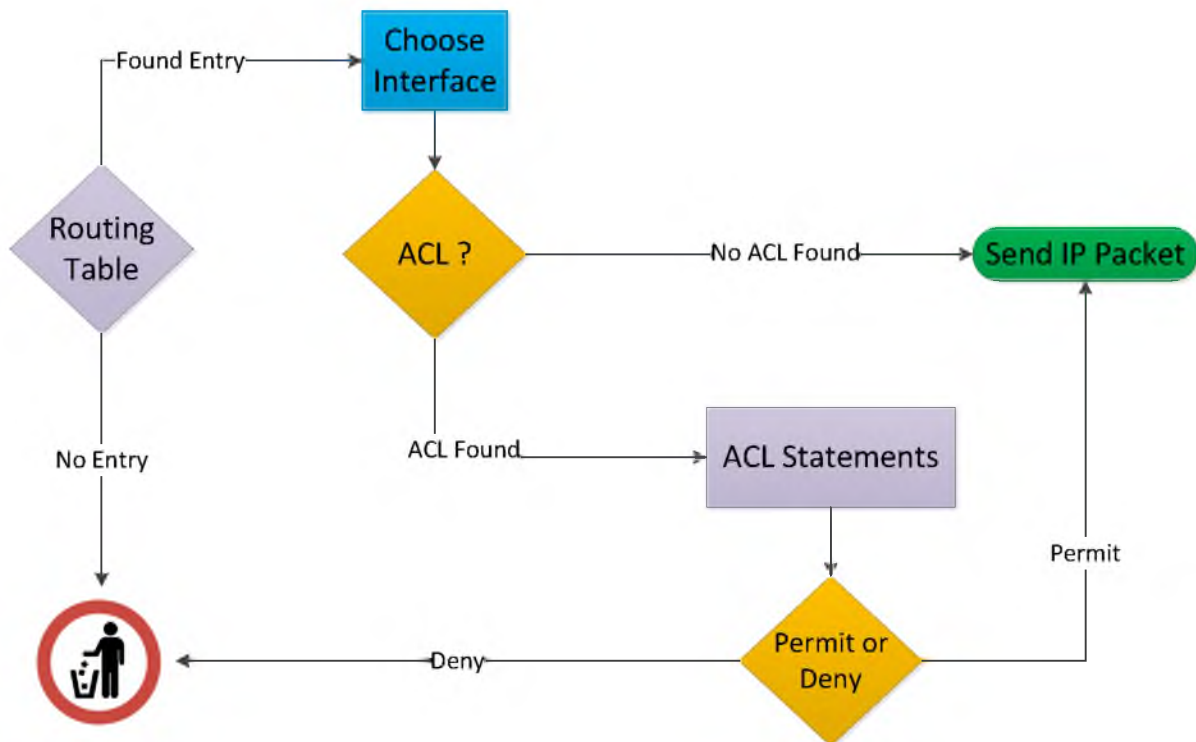
Let’s take a closer look at filtering. After creating an access-list there are 3 spots where you can place them:



You can put them **inbound** on the interface which means that all packets that reach your router will hit the access-list and will have to be checked against the access-list.



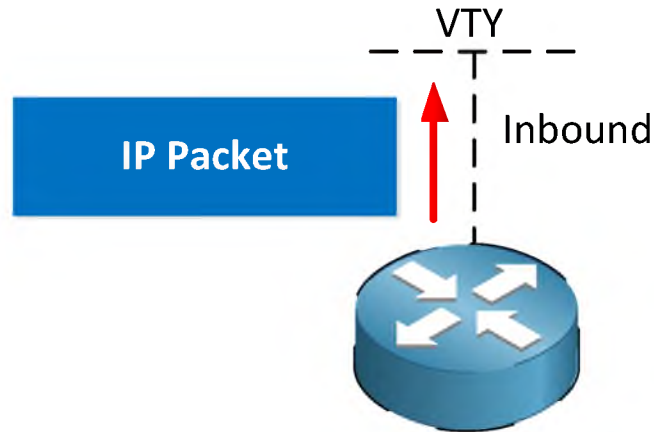
Another option is to put the access-list **outbound**. In this case IP packets will go through the router and once they are leaving the interface they will be checked against the access-list. When you place an access-list outbound, this is what your router will do:



1. IP Packets will enter your router.
2. Your router will check if it knows about the destination by looking in its routing table.
3. If there is no entry in the routing table the IP packet will be discarded.
4. If there is an entry in the routing table it will select the correct outgoing interface.
5. If there is no access-list the IP packet will be sent out of the interface.
6. If there is an access-list we'll have to check our IP packet and compare it with the access-list.
7. If the IP packet is permitted it will be forwarded, otherwise it will be discarded and go to IP heaven.



The third option is applying it to the VTY line. We can use this to secure telnet and/or SSH traffic.



Let me give you an example of what an access-list looks like:

```
Router#show access-lists
Standard IP access list 1
 10 permit 192.168.1.0, wildcard bits 0.0.0.255
 20 permit 192.168.2.0, wildcard bits 0.0.0.255
 30 permit 172.16.0.0, wildcard bits 0.0.255.255
```

Access-lists work by using **statements**. In the picture above you can see access-list number 1 has 3 statements, number 10, 20 and 30. Whenever a packet hits the access-list this is what will happen:

- Access-lists are processed **top-down** so we first check if the packet matches statement 10.
- If it doesn't match statement 10, we'll check if it matches statement 20.
- If it doesn't match statement 20, we'll check if it matches statement 30.
- If it doesn't match statement 30, the packet will be **dropped**.

If a packet **does match** a certain statement then there is immediate action. The packet will either be **permitted** (forward) or **denied** (discarded). For example, if we have a packet that matches statement 10 then the router will **not check** if it "also" matches statement 20.

At the bottom of every access-list there is a **deny any** which means if you didn't explicitly permit something it will be dropped anyway. You **don't see** this deny any but it's there!



Access-lists work in a similar way as if you are at the airport. If you are looking at the departures sign you'll check all the information for your flight number.

Once you have spotted your flight number you will take action by going to the correct gate and you won't look if you see your flight number somewhere else on the display.

Don't forget about the **deny any** at the bottom of this access-list! It's just like checking your flight number on the departure sign, if you don't see your flight you are **going nowhere!**

There are two types of access-lists we can use:

- **Standard** access-lists
- **Extended** access-lists

Let's start with the standard access-list:



The **standard access-list** is very basic since it can only check for **source IP addresses**. You can't do anything more specific than that.



Our **extended access-list** gives us many more options. Not only can you check for **source** and **destination** IP addresses but you can also match on transport layer (layer 4) information like TCP or UDP port numbers.

Does this mean that standard access-lists suck? Well no since sometimes source IP addresses are all we care about...If you want an access-list to select which networks should be translated with your VPN then a standard access-list will do the trick just fine.

How can we recognize the standard and extended access-list? Let me show you the table below:

| IPv4 ACL Type   | Number / Identification |
|-----------------|-------------------------|
| <b>Standard</b> | 1-99 or 1300-1999       |
| <b>Extended</b> | 100-199 or 2000-2699    |
| <b>Named</b>    | Pick any name you like  |

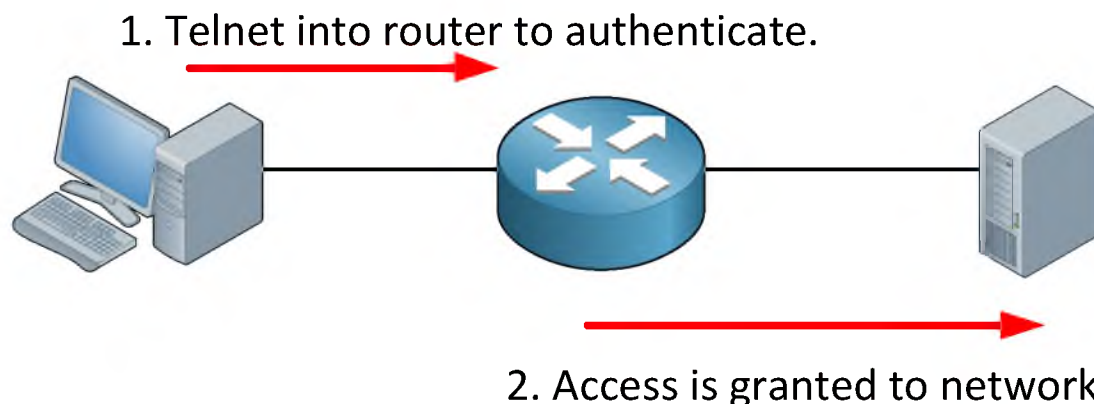
If you want a standard access-list you need to use a number between 1-99 or 1300-1999. For the extended access-list you need to pick a number between 100-199 or 2000-2699.

If you don't like numbers you can also use **named access-lists** by choosing a name, this works for both standard and extended access-lists.

Before we continue let me give you some guidelines when setting up access-lists:

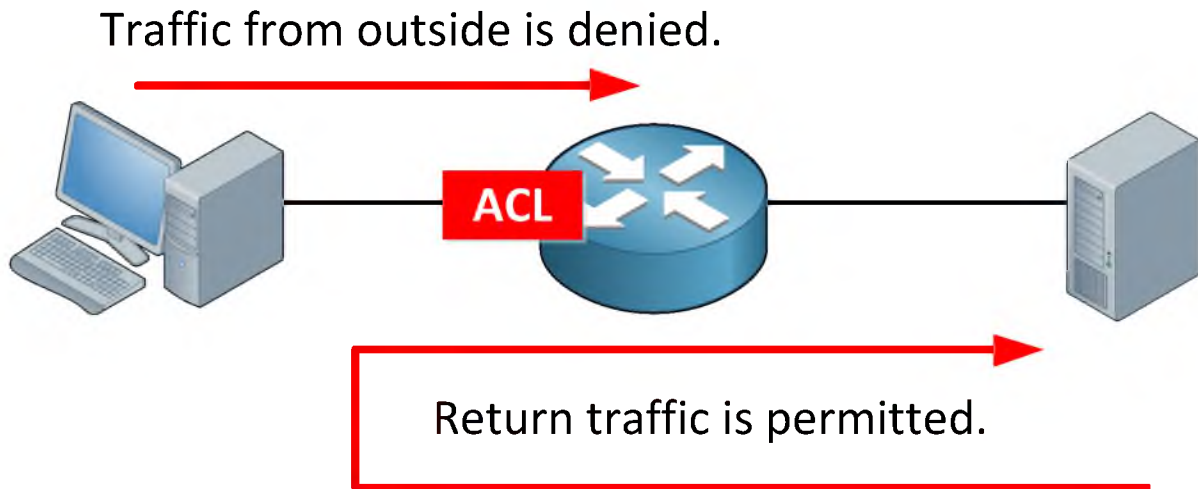
- First we create an access-list globally and then we assign it to an interface.
- You can only have a single ACL per direction, so it's impossible to have 2 inbound access-lists.
- Put the most specific statements at the top of your access-list because if a packet matches a statement the router doesn't check if it matches any other statements.
- Don't forget the last statement is deny any. You don't see it but it's there.

What other cool tricks can we do with access-lists? We can use an access-list for user authentication. Here's an example:



We have **dynamic access-lists** (also known as Lock-and-Key). If our user sitting behind the computer wants to access the server she will have to authenticate to the router by using telnet first before you get access to the server. Does this sound like a good idea? If you ask me you want users and routers to be as far away from each other as possible...besides there are far better solutions for this (802.1X and RADIUS).

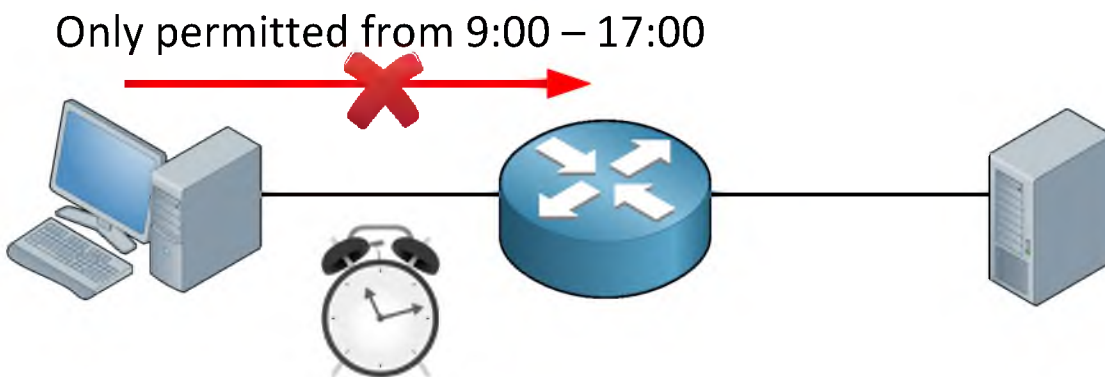
When you are creating access-lists you have to be careful not to drop any legitimate traffic. Here's a good example:



Look at the router above, we have an access-list inbound on the left interface so the computer on the left is unable to reach our internal network with the server on the right side.

Excellent but now the server on the right side wants to communicate with the computer on the outside. Traffic going outside will be permitted but our return-traffic will be dropped by the access-list. If you use a **reflexive access-list** it will make sure that "return" traffic is allowed back into the network. The reflexive access-list will automatically create temporary statements in our access-list to allow the return traffic.

Anything else that is useful?



You can also use **time-based access-lists** which only apply during weekdays, 9:00 – 17:00 or any other time/date you like.

Now you know what we can do with access-lists, let's take a closer look at the access-list itself:

```
Router#show access-lists
Standard IP access list 1
 10 permit 192.168.1.0, wildcard bits 0.0.0.255
 20 permit 192.168.2.0, wildcard bits 0.0.0.255
 30 permit 172.16.0.0, wildcard bits 0.0.255.255
```

This is the same example of an access-list I showed you before, look at the first line that we have.

```
10 permit 192.168.1.1 wildcard bits 0.0.0.255.
```

Access-lists don't use **subnet masks but wildcard bits**. If you read the OSPF chapter I explained you that wildcard bits are reverse subnet masks. This means that in binary a "0" will be replaced by a "1" and vice versa.

Let me show you some examples:

Subnet mask 255.255.255.0 would be 0.255.255.255 as the wildcard mask. To explain this I need to show you some binary:

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>255</b> | 1   | 1  | 1  | 1  | 1 | 1 | 1 | 1 |

This is the the first octet of the subnet mask (255.255.255.0) in binary, as you can see all values have a 1 making the decimal number 255.

| Bits     | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|----------|-----|----|----|----|---|---|---|---|
| <b>0</b> | 0   | 0  | 0  | 0  | 0 | 0 | 0 | 0 |

This is also the first octet but now with wildcard bits. If you want the wildcard-equivalent you need to flip the bits, if there's a 1 you need to change it into a 0. That's why we now have the decimal number 0.

Let me show you another subnet mask...let's take 255.255.255.128. What would be the wildcard-equivalent of this? We know the 255.255.255.X part so I'm only showing you the .128 part.

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>128</b> | 1   | 0  | 0  | 0  | 0 | 0 | 0 | 0 |

That's the last octet of our subnet mask, let's flip the bits:

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>127</b> | 0   | 1  | 1  | 1  | 1 | 1 | 1 | 1 |

Just flip the bits over and this is the wildcard-equivalent. This gives us the decimal number 127.

The subnet mask 255.255.255.128 will be 0.0.0.127 as a wildcard.

Just one more!

Subnet mask 255.255.255.224...what is the wildcard?

| Bits       | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|-----|----|----|----|---|---|---|---|
| <b>224</b> | 1   | 1  | 1  | 0  | 0 | 0 | 0 | 0 |

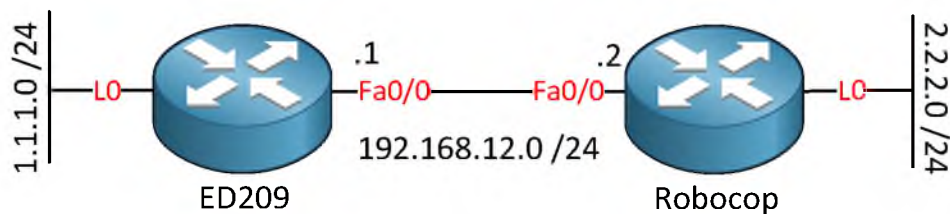
Flip those bits!

| Bits      | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|-----------|-----|----|----|----|---|---|---|---|
| <b>31</b> | 0   | 0  | 0  | 1  | 1 | 1 | 1 | 1 |

Our wildcard will be 0.0.0.31

Are you following me? It will take some practice by configuring access-lists to become familiar with these wildcards.

Let's configure some access-lists so I can demonstrate to you how this works on a real router! I will be using the following topology for this:



Two routers and each router has a loopback interface. I will use two static routes so that the routers can reach each other's loopback interface:

```
ED209(config)#ip route 2.2.2.0 255.255.255.0 192.168.12.2
```

```
Robocop(config)#ip route 1.1.1.0 255.255.255.0 192.168.12.1
```



*If you choose to use a routing protocol to advertise networks, be careful that your access-list doesn't block your RIP, EIGRP or OSPF traffic...*

Now let's start with a standard access-list! I'll create something on router Robocop that only permits traffic from network 192.168.12.0 /24:

```
Robocop(config)#access-list 1 permit 192.168.12.0 0.0.0.255
```

This single permit entry will be enough. Keep in mind at the bottom of the access-list is a "deny any". We don't see it but it's there. Let's apply this access-list inbound on router Robocop:

```
Robocop(config)#interface fastEthernet 0/0
Robocop(config-if)#ip access-group 1 in
```

Use the **ip access-group** command to apply it to an interface. I applied it inbound with the **in** keyword.

```
Robocop#show ip interface fastEthernet 0/0
FastEthernet0/0 is up, line protocol is up
  Internet address is 192.168.12.2/24
  Broadcast address is 255.255.255.255
  Address determined by setup command
  MTU is 1500 bytes
  Helper address is not set
  Directed broadcast forwarding is disabled
  Outgoing access list is not set
  Inbound access list is 1
```

You can verify that the access-list has been applied with the **show ip interface** command. Above you see that access-list 1 has been applied inbound.

Now let's generate some traffic...

```
ED209#ping 192.168.12.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.12.2, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
```

Our ping is successful; let's check the access-list:

```
Robocop#show access-lists
Standard IP access list 1
  10 permit 192.168.12.0, wildcard bits 0.0.0.255 (27 matches)
```

As you can see the access-list shows the number of matches per statement. We can use this to verify our access-list. Let me show you something useful when you are playing with access-lists:

```
ED209#ping 192.168.12.2 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.12.2, timeout is 2 seconds:
Packet sent with a source address of 1.1.1.1
U.U.U
Success rate is 0 percent (0/5)
```

When you send a ping you can use the **source** keyword to select the interface.



The source IP address of this IP packet is now 1.1.1.1 and you can see these pings are failing because the access-list drops them.

```
Robocop#show access-lists
Standard IP access list 1
 10 permit 192.168.12.0, wildcard bits 0.0.0.255 (27 matches)
```

You won't see them with the show access-list command because the "deny any" is dropping them.

What if I wanted something different? Let's say I want to deny traffic from network 192.168.12.0 /24 but permit all other networks? I can do something like this:

```
Robocop(config)#access-list 2 deny 192.168.12.0 0.0.0.255
Robocop(config)#access-list 2 permit any
```

I'll create a new access-list and the first statement will deny network 192.168.12.0 /24. The second statement is a permit any.

Because of this permit any nothing will ever hit the invisible "deny any" with the exception of 192.168.12.0 /24. Let's apply the new access-list:

```
Robocop(config-if)#no ip access-group 1 in
Robocop(config-if)#ip access-group 2 in
```

Now it's active, let's give it a test run:

```
ED209#ping 2.2.2.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
U.U.U
Success rate is 0 percent (0/5)
```

```
Robocop#show access-lists 2
Standard IP access list 2
 10 deny 192.168.12.0, wildcard bits 0.0.0.255 (11 matches)
 20 permit any
```

These pings are hitting the first statement and are dropped....

```
ED209#ping 2.2.2.2 source loopback 0
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
Packet sent with a source address of 1.1.1.1
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
```

```
Robocop#show access-lists 2
Standard IP access list 2
 10 deny 192.168.12.0, wildcard bits 0.0.0.255 (11 matches)
```



```
20 permit any (15 matches)
```

And pings from the loopback0 interface of router ED209 are hitting the second statement and are allowed.

If I want to remove a statement from an access-list, you will await a nice surprise:

```
Robocop(config)#no access-list 2 deny 192.168.12.0 0.0.0.255
```

Let's say I want to remove the statement above. I'll type **no access-list** and this is what you'll discover:

```
Robocop#show access-lists 2
```

The whole access-list is gone...ouch! You can't use **no access-list** to remove a statement. Your router will just accept "no access-list 2" and remove the whole access-list. Fun to discover in a lab, not so much fun on a production network. I'll show you how to deal with this in a bit.

Besides applying an access-list inbound or outbound you can also apply them to the VTY lines. This is useful if you want to secure telnet or SSH access to your router.

Let's configure router ED209 so telnet access is only allowed from network 192.168.12.0 /24:

```
ED209(config)#access-list 3 permit 192.168.12.0 0.0.0.255
ED209(config)#line vty 0 4
ED209(config-line)#access-class 3 in
```

Above you can see that I created access-list 3 but I used the **access-class** command on the VTY lines. On interfaces we use the "access-group" command but on VTY lines you need to use "access-class" to apply them.

Let's try to use telnet:

```
Robocop#telnet 192.168.12.1
Trying 192.168.12.1 ... Open

Password required, but none set

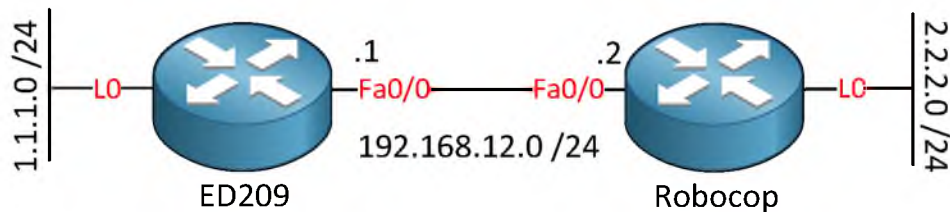
[Connection to 192.168.12.1 closed by foreign host]
```

It says "open" which means that it connects. The connection is closed because I didn't configure a password for telnet but the access-list should work:

```
ED209#show access-lists
Standard IP access list 3
 10 permit 192.168.12.0, wildcard bits 0.0.0.255 (2 matches)
```

You can see that the packets have matched the statement in access-list 3.

Now you have seen the standard access-list, let me show you how to configure the extended access-list. We will use the same topology:



Using the extended access-list we can create far more complex statements. Let's say we have the following requirement:

- Traffic from network 1.1.1.0 /24 is allowed to connect to the HTTP server on router Robocop, but they are only allowed to connect to IP address 2.2.2.2.
- All other traffic has to be denied.

Now we need to translate this to an extended access-list statement. Basically they look like this:

[source] + [ source port] to [destination] + [destination port]

Let's walk through the configuration together:

```

Robocop(config)#access-list 100 ?
deny      Specify packets to reject
dynamic   Specify a DYNAMIC list of PERMITs or DENYs
permit    Specify packets to forward
remark    Access list entry comment
  
```

First of all we need to select a permit or deny. By the way you can also use a **remark**. You can use this to add a comment to your access-list statements. I'll select the permit...

```

Robocop(config)#access-list 100 permit ?
<0-255>   An IP protocol number
ahp       Authentication Header Protocol
eigrp     Cisco's EIGRP routing protocol
esp       Encapsulation Security Payload
gre       Cisco's GRE tunneling
icmp      Internet Control Message Protocol
igmp      Internet Gateway Message Protocol
ip        Any Internet Protocol
ipinip    IP in IP tunneling
nos       KA9Q NOS compatible IP over IP tunneling
ospf      OSPF routing protocol
pcp       Payload Compression Protocol
pim       Protocol Independent Multicast
tcp       Transmission Control Protocol
udp       User Datagram Protocol
  
```

Now we have a lot more options. Since I want something that permits HTTP traffic we'll have to select TCP. Let's continue:

```
Robocop(config)#access-list 100 permit tcp ?
```

```
A.B.C.D Source address
any      Any source host
host     A single source host
```

Now we have to select a **source**. I can either type in a network address with a wildcard or I can use the **any** or **host** keyword. These two keywords are “shortcuts”, let me explain:

- If you type “0.0.0.0 255.255.255.255” you have all networks. Instead of typing this we can use the **any** keyword.
- If you type something like “2.2.2.2 0.0.0.0” we are matching a single IP address. Instead of typing the “0.0.0.0” wildcard we can use the keyword **host**.

I want to select network 1.1.1.0 /24 as the source so this is what we will do:

```
Robocop(config)#access-list 100 permit tcp 1.1.1.0 0.0.0.255 ?
```

```
A.B.C.D Destination address
any      Any destination host
eq       Match only packets on a given port number
gt       Match only packets with a greater port number
host     A single destination host
lt       Match only packets with a lower port number
neq      Match only packets not on a given port number
range    Match only packets in the range of port numbers
```

Besides selecting the source we can also select the **source port number**. Keep in mind that when I connect from router ED209 to router Robocop’s HTTP server that my source port number will be **random** so I’m not going to specify a source port number here.

```
Robocop(config)#access-list 100 permit tcp 1.1.1.0 0.0.0.255 host 2.2.2.2 ?
```

```
ack      Match on the ACK bit
dscp     Match packets with given dscp value
eq      Match only packets on a given port number
established Match established connections
fin      Match on the FIN bit
fragments Check non-initial fragments
gt       Match only packets with a greater port number
log      Log matches against this entry
log-input Log matches against this entry, including input interface
lt       Match only packets with a lower port number
neq      Match only packets not on a given port number
precedence Match packets with given precedence value
psh      Match on the PSH bit
range    Match only packets in the range of port numbers
rst      Match on the RST bit
syn      Match on the SYN bit
time-range Specify a time-range
tos      Match packets with given TOS value
urg      Match on the URG bit
<cr>
```

We will select the destination which is IP address 2.2.2.2. I could have typed “2.2.2.2 0.0.0.0” but it’s easier to use the host keyword. Besides the destination IP address we can select a destination port number with the **eq** keyword:

```
Robocop(config)#access-list 100 permit tcp 1.1.1.0 0.0.0.255 host 2.2.2.2 eq 80
```

This will be the end result. Before we apply it to the interface I will add one useful extra statement:

```
Robocop(config)#access-list 100 deny ip any any log
```

Using the statement above I can make that invisible “deny any” visible. The **log** keyword will output all denied packets to the console.

Now let’s apply it and give it a test run!

```
Robocop(config)#interface fastEthernet 0/0
Robocop(config-if)#ip access-group 100 in
```

We’ll apply it to the interface inbound. Don’t forget to enable the HTTP server:

```
Robocop(config)#ip http server
```

Now let’s generate some traffic:

```
ED209#telnet 2.2.2.2 80
Trying 2.2.2.2, 80 ...
% Destination unreachable; gateway or host down
```

I don’t need a web browser to test if the HTTP server is running. I can use telnet to connect to TCP port 80. The traffic above is denied as you will see on the console of router Robocop:

```
Robocop# %SEC-6-IPACCESSLOGP: list 100 denied tcp 192.168.12.1(55419) ->
2.2.2.2(80), 1 packet
```

Or we can take a look at the matches on the access-list:

```
Robocop#show access-lists
Extended IP access list 100
 10 permit tcp 1.1.1.0 0.0.0.255 host 2.2.2.2 eq www
 20 deny ip any any log (1 match)
```

The packet was denied because the source IP address was 192.168.12.1. Let’s connect from IP address 1.1.1.1:

```
ED209#telnet 2.2.2.2 80 /source-interface loopback 0
Trying 2.2.2.2, 80 ... Open
```

There we go! It now says open which means that it connected. When we use telnet we can select the **source interface**. The packet is now allowed because it matches the first statement of the access-list.

If I want to remove a single statement from my access-list I have two options:

- Copy your access-list to notepad, edit it and paste it back to your router and use a new access-list..
- Use the access-list editor.

The access-list editor sounds easier right? This is how it works:

```
Robocop(config)#ip access-list extended 100
```

Use the **ip access-list** command to create new access-list or modify current ones.

Your console will look like this:

```
Robocop(config-ext-nacl)#
```

Now we can add or remove statements:

```
Robocop(config-ext-nacl)#?
Ext Access List configuration commands:
<1-2147483647>  Sequence Number
default        Set a command to its defaults
deny           Specify packets to reject
dynamic        Specify a DYNAMIC list of PERMITs or DENYs
evaluate       Evaluate an access list
exit           Exit from access-list configuration mode
no             Negate a command or set its defaults
permit         Specify packets to forward
remark         Access list entry comment
```

Let's remove statement 20 from access-list 100:

```
Robocop(config-ext-nacl)#do show access-list 100
Extended IP access list 100
 10 permit tcp 1.1.1.0 0.0.0.255 host 2.2.2.2 eq www (21 matches)
 20 deny ip any any log (1 match)
```

This is what it looks like now...

```
Robocop(config-ext-nacl)#no 20
```

Type **no** in front of the sequence number and it will be gone:

```
Robocop(config-ext-nacl)#do show access-list 100
Extended IP access list 100
 10 permit tcp 1.1.1.0 0.0.0.255 host 2.2.2.2 eq www (21 matches)
```

Voila it's now gone.

Last but not least we can also create a named access-list. Let's create something that denies ICMP traffic from router Robocop to router ED209's loopback0 interface but allows everything else:

```
ED209(config)#ip access-list extended DROPICMP
ED209(config-ext-nacl)#deny icmp host 192.168.12.2 1.1.1.0 0.0.0.255
ED209(config-ext-nacl)#deny icmp host 2.2.2.2 1.1.1.0 0.0.0.255
ED209(config-ext-nacl)#permit ip any any
ED209(config-ext-nacl)#exit
```

This is what the access-list will look like. I'll call it "DROPICMP". The first statement will drop ICMP traffic from IP address 192.168.12.2 and the second line is for IP address 2.2.2.2. All other traffic is permitted. Let's apply it to the interface:

```
ED209(config)#interface fastEthernet 0/0
ED209(config-if)#ip access-group DROPICMP in
```

Now let's test it:

```
Robocop#ping 1.1.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1.1.1.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

```
ED209#show access-lists
Extended IP access list DROPICMP
  10 deny icmp host 192.168.12.2 1.1.1.0 0.0.0.255 (15 matches)
  20 deny icmp host 2.2.2.2 1.1.1.0 0.0.0.255
  30 permit ip any any
```

The first ping is failing as it should...

```
Robocop#ping 1.1.1.1 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 2.2.2.2
.....
Success rate is 0 percent (0/5)
```

```
ED209#show access-lists
Extended IP access list DROPICMP
  10 deny icmp host 192.168.12.2 1.1.1.0 0.0.0.255 (15 matches)
  20 deny icmp host 2.2.2.2 1.1.1.0 0.0.0.255 (15 matches)
  30 permit ip any any
```

And the second ping fails too...

Let's do something crazy to get a match on the last statement:

```
Robocop#telnet 1.1.1.1
Trying 1.1.1.1 ...
```

```
ED209#show access-lists
Extended IP access list DROPICMP
  10 deny icmp host 192.168.12.2 1.1.1.0 0.0.0.255 (27 matches)
```

```
20 deny icmp host 2.2.2.2 1.1.1.0 0.0.0.255 (18 matches)
30 permit ip any any (12 matches)
```

I didn't configure telnet on router ED209 but my packets will hit the last statement anyway.

Want some labs to practice access-lists? I have some very nice labs to do!

<http://gns3vault.com/Security/standard-access-list.html>

This will teach you how to configure the standard access-list.

<http://gns3vault.com/Security/extended-access-list.html>

Same idea as the previous lab but now you have to use extended access-lists to create more advanced access-lists.

<http://gns3vault.com/Security/named-access-list.html>

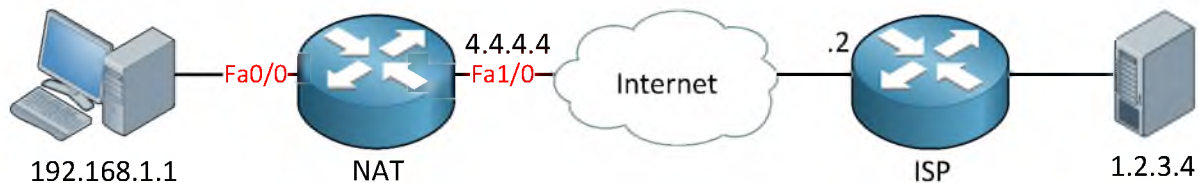
Named access-lists can be standard or extended but configuration is a bit different, good idea to check them out.

<http://gns3vault.com/Security/time-based-access-list.html>

Just a little extra, access-lists can be time-based which is what you will learn with this lab.

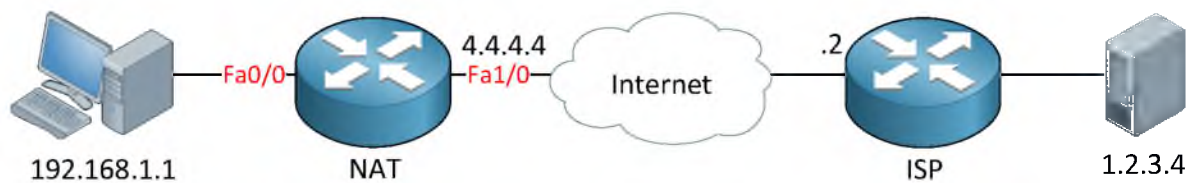
## 19. Network and Port address Translation (NAT & PAT)

Without network address translation (NAT) or port address translation (PAT) you probably wouldn't be able to access the internet from your computer or at least you'll be the only one in the house having internet access...



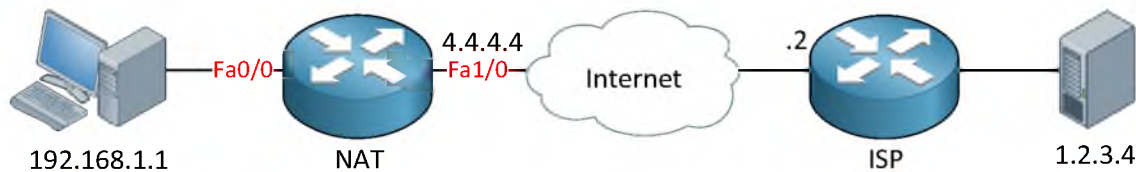
On the left side we have a computer on our LAN with the IP address 192.168.1.1 connected to a router. From our ISP we got the IP address 4.4.4.4 and there's a server on the Internet using IP address 1.2.3.4

If our computer send something to the server what would be the source and destination IP address of the IP packet it will send?



| IP Packet | Source IP   | Destination IP |
|-----------|-------------|----------------|
|           | 192.168.1.1 | 1.2.3.4        |

The source IP address will be our computer and the destination IP address will be the server as you can see in the IP packet in the picture above.



| IP Packet | Source IP   | Destination IP |
|-----------|-------------|----------------|
|           | 192.168.1.1 | 1.2.3.4        |

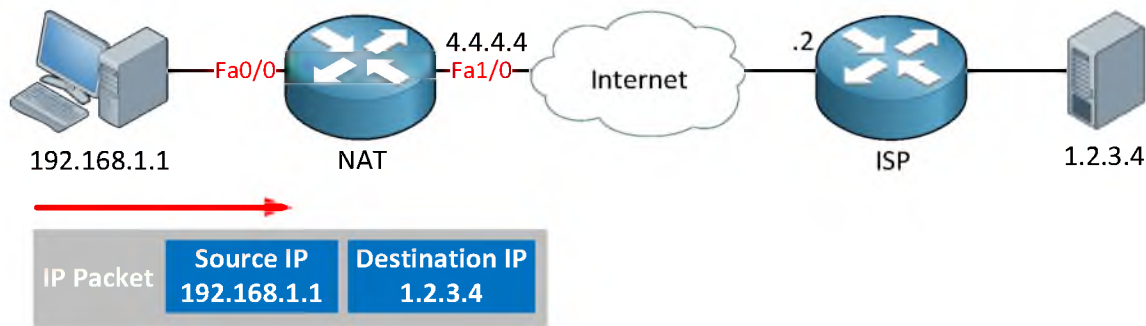
| IP Packet | Source IP | Destination IP |
|-----------|-----------|----------------|
|           | 1.2.3.4   | 192.168.1.1    |

Once our server responds it will create an IP packet specifying the computer's IP address as the destination and the source IP address will be its own IP address.

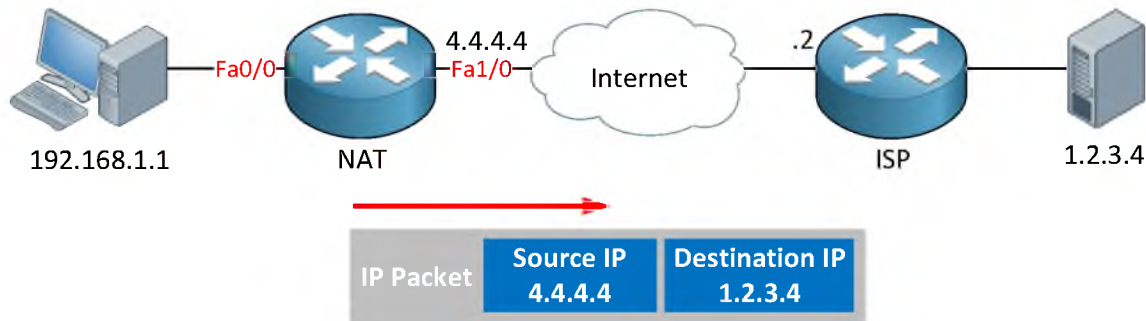


Is there anything wrong with this example? No, it's perfectly fine except for one detail...the IP address of the computer is a private IP address. As you might recall from one of the first chapters in this book, private IP addresses are meant for our LANs and public IP addresses are for the Internet. Your ISP will drop traffic from private IP addresses.

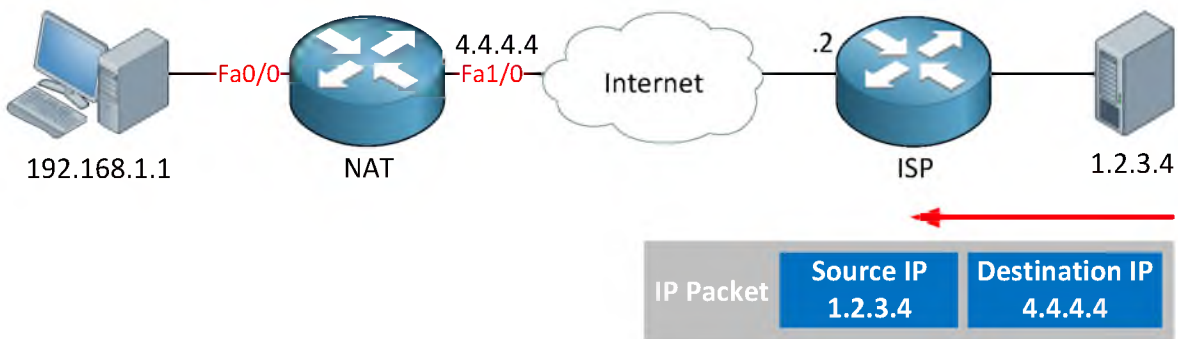
This time we are going to configure NAT (Network Address Translation) and see what the difference is.



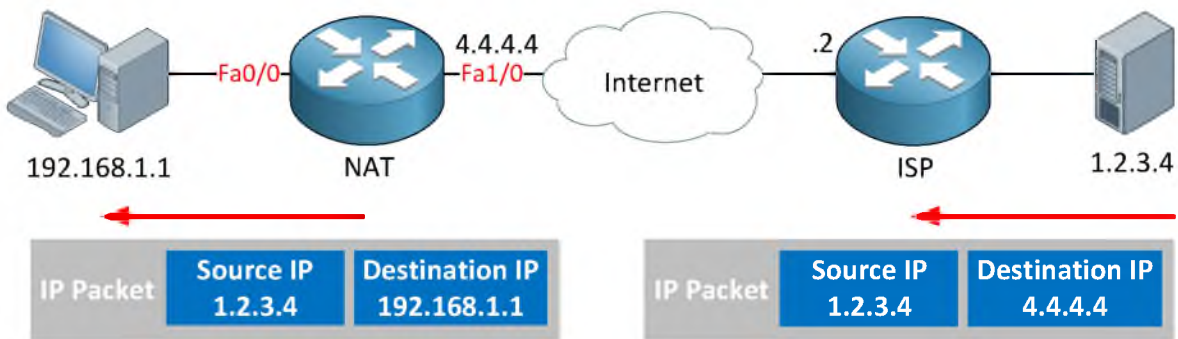
Same story, our computer is going to send something to the server but now our router has been configured for NAT. The NAT router has been configured so IP address 192.168.1.1 has to be translated to IP address 4.4.4.4.



Here's what happens. Our NAT router will rewrite the source IP address from 192.168.1.1 to 4.4.4.4 as you can see in the IP packet above.



The server thinks it's talking to IP address 4.4.4.4 which is why you see this IP address as the destination in the IP packet it's sending.



Once this IP packet reaches the router it will look again at its NAT table and translate the IP address 4.4.4.4 back into 192.168.1.1 and send it towards the computer.

The example I just showed you is called **static NAT**. There is a 1:1 relationship between the IP address of our computer on the LAN and the IP address we got from our ISP. So what are we going to do if we have more computers on our LAN? We can use something called **dynamic NAT**.

Dynamic NAT is different compared to static NAT because:

- You can use a pool of IP addresses to translate into.
- You can use an access-list to match the hosts on your LAN which should be translated.

To give you an example, in our static NAT picture we used the 10.1.1.1 IP address from the ISP to translate. Our ISP is very generous and instead of giving us a single IP address we get a range of IP addresses, in fact we got the whole 10.1.1.0/24 subnet.

Besides our computer 192.168.1.1 there are 10 other computers that need Internet access. What's going to happen now? We now have a pool of IP addresses from the ISP we can use to translate into.

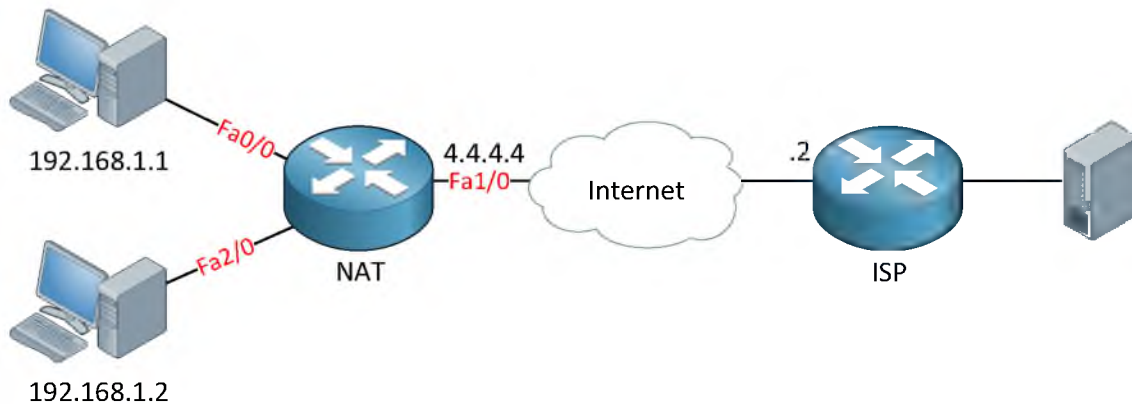
Let's discuss an example:

1. The computer with 192.168.1.1 is visiting a server on the Internet, our NAT router will translate this IP address to the first IP address from the pool, 10.1.1.1.
2. The next computer with 192.168.1.2 is now visiting a server on the Internet, our NAT router will translate this IP address to the second IP address from the pool, 10.1.1.2.
3. The third computer with 192.168.1.3 is also visiting something on the Internet, the NAT router will translate this IP address to the third IP address from the pool, 10.1.1.3.
4. Etc.

This is what we call dynamic NAT.

Now maybe I got you puzzled...you probably have more than one device at your LAN accessing the Internet but you only got a single IP address from your ISP. How can this work?

This is where we introduce **PAT** or **Port Address Translation**. NAT only gives us a 1:1 relationship between two IP addresses. If we have multiple computers on our LAN and only a single IP address from our ISP we need to translate port numbers as well. This way we can have multiple computers behind a single public IP address from the ISP.



Look at the network above, we have two computers on our LAN with IP address 192.168.1.1 and 192.168.1.2. Our router is configured for NAT:

The following situation is happening:

1. Computer with IP address 192.168.1.1 is going to connect to the server.
2. Our NAT router will translate 192.168.1.1 to 4.4.4.4.
3. Our other computer with IP address 192.168.1.2 is also connecting to the server.
4. Our NAT router now has a problem since 192.168.1.1 is already translated to 4.4.4.4. You can't have two IP addresses translated.

This is where PAT kicks in; with PAT this is what will happen:

1. Computer with IP address 192.168.1.1 is going to connect to the server.
2. Our NAT router will translate 192.168.1.1 to 4.4.4.4 but will also keep track of the source and destination port!

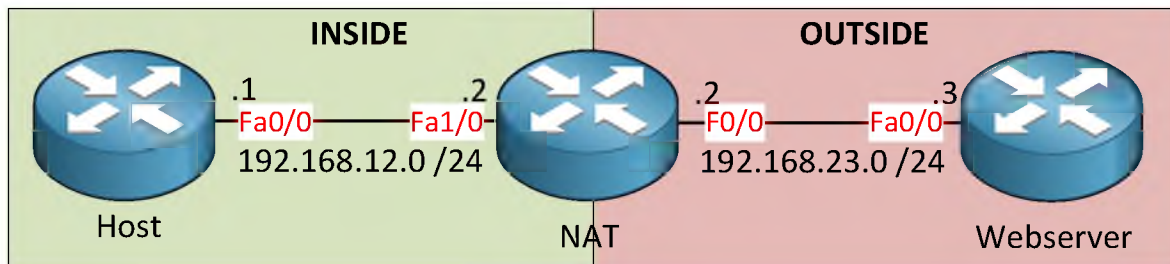
3. Our other computer with IP address 192.168.1.2 is also connecting to the server.
4. Since our NAT router also does PAT it will translate 192.168.1.2 to 4.4.4.4 as well and use another source port number.

That's how you can have multiple computers on your LAN and make all of them access the Internet behind a single public IP address from your ISP.

The server thinks it's only talking to 4.4.4.4 so it has no idea there is a computer with IP address 192.168.1.1 or 192.168.1.2. Does this mean NAT or PAT is a security protocol? This is a big debate but in my opinion it's no security mechanism. Not seeing the true hosts at your LAN doesn't mean you are unable to connect. As soon as your router is doing network and/or port address translation those hosts are reachable. Security is something you implement by using access-lists, firewalls, intrusion prevention systems and security policies.

Since NAT and/or PAT are changing the IP packet there are some applications that don't work too well with this translation of IP addresses and ports, IPSEC is an example. FTP is also troublesome behind a NAT router.

Enough theory? Let's take a look at NAT and PAT in action! I'll be using the following topology to demonstrate static NAT to you:



Above you see 3 routers called Host, NAT and Webserver. Imagine our host is on our LAN and the webserver is somewhere on the Internet. Our NAT router in the middle is our connection to the Internet.

There's a cool trick on our routers that we can use. It's possible to disable "routing" on a router which turns it into a normal host that requires a default gateway. This is very convenient because it will save you the hassle of connecting real computers/laptops to GNS3.

```
Host(config)#no ip routing
```

```
Webserver(config)#no ip routing
```

Use **no ip routing** to disable the routing capabilities.

The routing table is now gone, let me show you:

```
Host#show ip route
Default gateway is not set

Host                Gateway                Last Use      Total Uses   Interface
ICMP redirect cache is empty
```

```
Webserver#show ip route
Default gateway is not set

Host                Gateway                Last Use      Total Uses   Interface
ICMP redirect cache is empty
```

As you can see the routing table is gone. We'll have to configure a default gateway on router Host and Webserver or they won't be able to reach each other:

```
Host(config)#ip default-gateway 192.168.12.2
```

```
Webserver(config)#ip default-gateway 192.168.23.2
```

Both routers can use router NAT as their default gateway. Let's see if they can reach each other:

```
Host#ping 192.168.23.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.23.3, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/8/12 ms
```

Reachability is no issue as you can see. Now let me show you a neat trick:

```
Webserver#debug ip packet
IP packet debugging is on
```

I can use **debug ip packet** to see the IP packets that I receive. DON'T do this on a production network or you'll be overburdened with traffic! Now let's send that ping again...

```
Webserver#
IP: s=192.168.12.1 (FastEthernet0/0), d=192.168.23.3, len 100, rcvd 1
```

Above you see that our router has received an IP packet with source IP address 192.168.12.1 and destination IP address 192.168.23.3.

```
IP: tableid=0, s=192.168.23.3 (local), d=192.168.12.1 (FastEthernet0/0),
routed via RIB
```

And it will reply with an IP packet that has source address 192.168.23.3 and destination address 192.168.12.1.

Now let's configure NAT so you can see the difference:

```
NAT(config)#interface fastEthernet 1/0
NAT(config-if)#ip nat inside

NAT(config)#interface fastEthernet 0/0
NAT(config-if)#ip nat outside
```

First we'll have to configure the inside and outside interfaces. Our host is the "LAN" side so it's the inside. Our webserver is "on the Internet" so it's the outside of our network. Now we can configure our static NAT rule:

```
NAT(config)#ip nat inside source static 192.168.12.1 192.168.23.2
```

We use the **ip nat inside** command to translate an inside IP address (192.168.12.1) to an outside IP address (192.168.23.2).

```
NAT#show ip nat translations
Pro Inside global      Inside local      Outside local      Outside global
--- 192.168.23.2       192.168.12.1     ---               ---
```

You can use the **show ip nat translations** command to verify our configuration. Now let's send another ping and see if this configuration does anything...

```
Host#ping 192.168.23.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.23.3, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/8/12 ms
```

```
Webserver#
IP: s=192.168.23.2 (FastEthernet0/0), d=192.168.23.3, len 100, rcvd 1
```

See the difference? The packet that the webserver receives from the host has source IP address 192.168.23.2.

```
Webserver#
IP: tableid=0, s=192.168.23.3 (local), d=192.168.23.2 (FastEthernet0/0),
routed via RIB
```

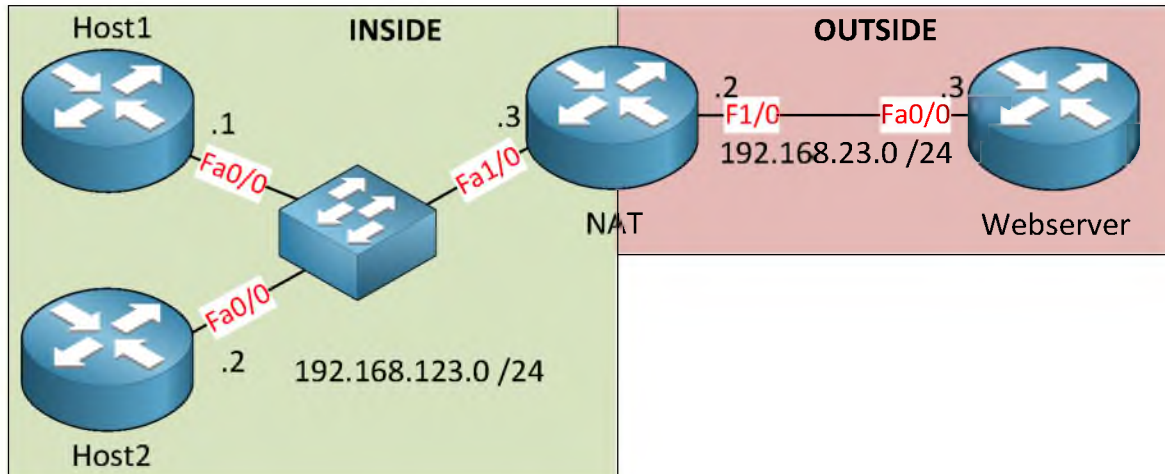
And when it responds the destination IP address is 192.168.23.2.

Now we know that static NAT is working.



*In the example I just showed you our Webserver doesn't require a default gateway anymore. The packets are translated to 192.168.23.2 and this network is directly connected for the Webserver.*

Next step is dynamic NAT. I'll use a different topology this time:



This time we have 2 host routers on the left side and I'm using another subnet. Let's prepare the host routers:

```
Host1(config)#no ip routing
Host1(config)#default gateway 192.168.123.3
```

```
Host2(config)#no ip routing
Host2(config)#ip default-gateway 192.168.123.3
```

Next step is to configure NAT:

```
NAT(config)#interface fastEthernet 0/0
NAT(config-if)#ip nat inside
```

```
NAT(config)#interface fastEthernet 1/0
NAT(config-if)#ip nat outside
```

First we'll configure the correct inside and outside interfaces. Now I will create a pool with IP addresses that we can use for the translation:

```
NAT(config)#ip nat pool MYPOOL 192.168.23.10 192.168.23.20 prefix-length 24
```

The **ip nat pool** command lets us create a pool. I'm calling mine "MYPOOL" and I'm using IP address 192.168.23.10 up to 192.168.23.20. We can now select the hosts that we want to translate:

```
NAT(config)#access-list 1 permit 192.168.123.0 0.0.0.255
```

The access-list above matches network 192.168.123.0 /24. That's where host1 and host2 are located. The last step is to put the access-list and pool together:

```
NAT(config)#ip nat inside source list 1 pool MYPOOL
```



The command above selects access-list 1 as the source and we will translate it to the pool called "MYPOOL". This ensures that host1 and host2 are translated to an IP address from our pool. Now let's verify our configuration!

```
Host1#ping 192.168.23.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.23.3, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/12/28 ms
```

Host1 is able to ping the webserver, now let's take a look at our NAT router:

```
NAT#show ip nat translations
Pro Inside global      Inside local           Outside local          Outside global
icmp 192.168.23.10:3    192.168.123.1:3       192.168.23.3:3       192.168.23.3:3
--- 192.168.23.10      192.168.123.1        ---                  ---
```

As you can see above host1 has been translated to IP address 192.168.23.10. Now let's send some traffic from host2 to see the difference in our NAT table...

```
Host2#ping 192.168.23.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.23.3, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/9/16 ms
```

```
NAT#show ip nat translations
Pro Inside global      Inside local           Outside local          Outside global
icmp 192.168.23.10:4    192.168.123.1:4       192.168.23.3:4       192.168.23.3:4
--- 192.168.23.10      192.168.123.1        ---                  ---
icmp 192.168.23.11:2    192.168.123.2:2       192.168.23.3:2       192.168.23.3:2
--- 192.168.23.11      192.168.123.2        ---                  ---
```

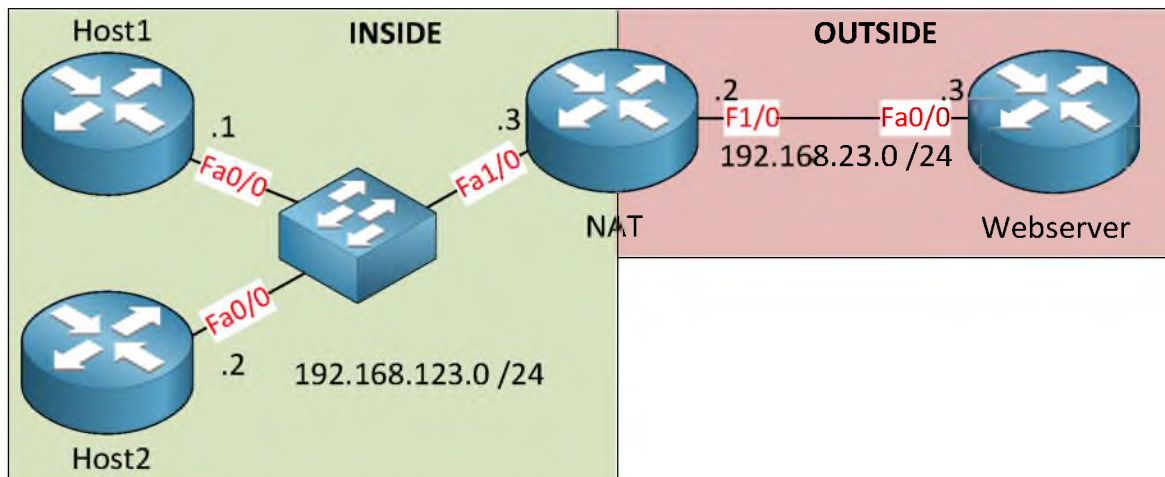
And as you can see host2 has been translated to IP address 192.168.23.11. Excellent our dynamic NAT is working! In case you are wondering...what do the inside global, inside local, outside local and outside global addresses mean? Let me explain to you:

- Inside global is the IP address on the outside interface of your router performing NAT.
- Inside local is the IP address of one of your inside hosts that is translated with NAT.
- Outside local is the IP address of the device you are trying to reach, in our example the webserver.
- Outside global is also the IP address of the device you are trying to reach, in our example the webserver.

Why are the outside local and outside global IP address the same? Well this is way outside the scope of the CCNA exam but with NAT it's possible to translate more than just from "inside" to "outside". It's possible to create an entry in our NAT router that whenever one of the hosts sends a ping to an IP address (let's say 5.5.5.5) that it will be forwarded to the webserver. In this example the "outside webserver" is "locally" seen by our hosts as 5.5.5.5, not 192.168.23.3.



Anyway let's continue with PAT!



I'll

use the same topology as previously. Let's prepare the hosts:

```
Host1(config)#no ip routing
Host1(config)#default gateway 192.168.123.3
```

```
Host2(config)#no ip routing
Host2(config)#ip default-gateway 192.168.123.3
```

Next step is to configure NAT:

```
NAT(config)#interface fastEthernet 0/0
NAT(config-if)#ip nat inside
```

```
NAT(config)#interface fastEthernet 1/0
NAT(config-if)#ip nat outside
```

So far so good, let's create an access-list that matches both hosts:

```
NAT(config)#access-list 1 permit 192.168.123.0 0.0.0.255
```

And finally we'll configure PAT:

```
NAT(config)#ip nat inside source list 1 interface fastEthernet 1/0 overload
```

I select access-list 1 as my inside source and I will translate them to the IP address on FastEthernet 1/0. The big magic keyword here is **overload**. If you add this we will enable PAT!

Let's give it a test run shall we?

To take a closer look at the port number I won't use ping but we'll connect to TCP port 80 of the webserver:

```
Webserver(config)#ip http server
```

First we'll enable the webserver.

We can use telnet to connect to port 80:

```
Host1#telnet 192.168.23.3 80
Trying 192.168.23.3, 80 ... Open
```

```
Host2#telnet 192.168.23.3 80
Trying 192.168.23.3, 80 ... Open
```

As you see it says "open" which means that we successfully connected to port 80.

Let's see what the NAT/PAT table looks like now:

```
NAT#show ip nat translations
Pro Inside global      Inside local      Outside local      Outside global
tcp 192.168.23.2:46369 192.168.123.1:46369 192.168.23.3:80   192.168.23.3:80
tcp 192.168.23.2:50669 192.168.123.2:50669 192.168.23.3:80   192.168.23.3:80
```

Above you see that it keeps track of the port number and that both hosts are translated to IP address 192.168.23.2. Mission accomplished!



*Telnet is a great command to connect to different TCP ports. You can use it to test access-lists or connectivity...or in my example to play with NAT/PAT.*

That's the end of the NAT/PAT chapter. What do you think so far? If you want some exercise see if you can configure the things I just showed you yourself, or try these labs:

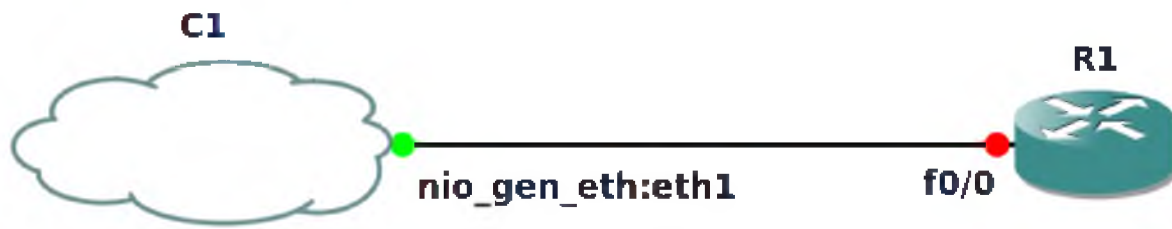
<http://gns3vault.com/Network-Services/nat-static.html>

And if you think that one was easy see if you can solve one of my troubleshooting labs on NAT and PAT:

<http://gns3vault.com/Troubleshooting/nat-dynamic-troubleshooting.html>

The troubleshooting labs are a bit different. You will have to solve a network that has configuration errors. If you manage to work your way through you'll be good on NAT / PAT for CCNA!

A fun way to test NAT by using GNS3 is to connect your emulated router to the Internet by using the "cloud" icon. This allows you to connect the interface of your emulator router to a real physical network card in your computer:



Give it a try and see if your router in GNS3 can access the Internet by using NAT/PAT!

## 20. Wide area networks

It's time to explore the world and get out of our LAN! In the Ethernet chapter we talked a bit about local area networks and what the definition LAN exactly means. So when do we talk about our LAN and when about WAN?

Our LAN is our local network. We own it and it's mostly a single building or a small geographic area like a campus with some buildings. If we want to connect to other networks and get out of our LAN that's when we use our **WAN** (Wide Area Network). To get WAN access we'll call a service provider and they might offer our Internet access or perhaps a leased line or other WAN connectivity method.

Remember the OSI-model?



If we talk about wide area networks what layers of the OSI-Model are we talking about?

Does anything change with IP or the upper layers? Nope we will still run IPv4 (or IPv6). The only layers that might change when we talk about WANs are the data link (layer 2) and the physical layer (layer 1).

When we talk about the physical layer there are two key terms you need to understand. Normally the customer side is called the **DTE** or **Data Terminal Equipment**. The ISP side is called the **DCE** or **Data Circuit-Terminating Equipment**. I'm not going into much detail here but for the CCNA we are going to play with WAN protocols like **PPP**, **HDLC** and **Frame-Relay**. Since we don't use Ethernet and UTP cables we use serial cables for these protocols.



Courtesy of Cisco Systems, Inc. Unauthorized use not permitted

Why am I telling you all this? If you use serial cables there's a DTE and DCE side. The DCE side (which is normally your ISP) will have to provide a **clock rate**. If you don't provide a clock rate your physical link won't even work!

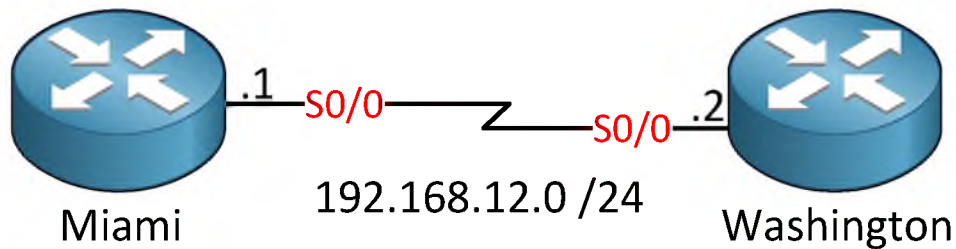
There are three WAN protocols I'm going to cover:

- HDLC
- PPP
- Frame-Relay

HDLC and PPP are point-to-point protocols. If you have two sites, let's say one in New York and the other one in Washington you might get a leased line. A leased line is an example of a point-to-point link and this is where you would run HDLC or PPP.

There is not much to tell you about **HDLC**. It's a point-to-point protocol and it's the default for Cisco routers. HDLC is a standard but running HDLC between routers from different vendors **is not going to work**. Keep this in mind. Every vendor has a proprietary field in their HDLC implementation which is what makes it incompatible between vendors.

Let me show you a quick example of HDLC:



I'm using two routers connected to each other with a serial link. First we will configure the clock rate on the router that has the DCE side of the cable:

```
Washington(config)#interface serial 0/0
Washington(config-if)#clock rate 64000
```

The clock rate will set the speed. For my example it doesn't matter much what clock speed we use. We can use a command to verify that the DTE router has received the clock rate:

```
Miami#show controllers serial 0/0
Hardware is PowerQUICC MPC860
DTE V.35 TX and RX clocks detected
Idb at 0x81081AC4, driver data structure at 0x 81084AC0
```

In the example above router Miami is the DTE side and it has received a clock rate. **Show controllers** is useful when you don't have physical access to your hardware so you don't know which side of the cable is DTE or DCE.

Let's configure the IP addresses:

```
Miami(config)#interface serial 0/0
Miami(config-if)#no shutdown
Miami(config-if)#ip address 192.168.12.1 255.255.255.0
```

```
Washington(config)#interface serial 0/0
Washington(config-if)#no shutdown
Washington(config-if)#ip address 192.168.12.2 255.255.255.0
```

Nothing special so far, just two serial interfaces in the same subnet.

We can verify that it's using HDLC:

```
Miami#show interfaces serial 0/0
Serial0/0 is up, line protocol is up
  Hardware is M4T
  Internet address is 192.168.12.1/24
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
    reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation HDLC, crc 16, loopback not set
```

```
Washington#show interfaces serial 0/0
Serial0/0 is up, line protocol is up
  Hardware is M4T
  Internet address is 192.168.12.2/24
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
    reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation HDLC, crc 16, loopback not set
```

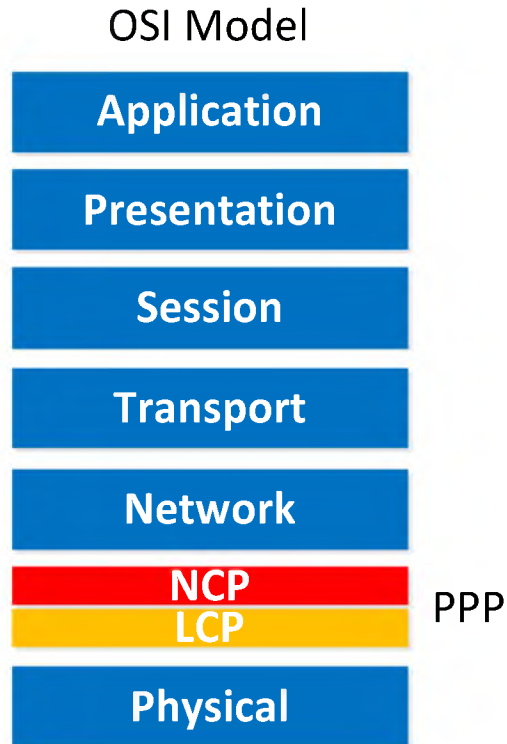
Use the **show interfaces** command to check the encapsulation type. You can see it says HDLC on both sides. Let's test connectivity with a quick ping:

```
Washington#ping 192.168.12.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.12.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/8/8 ms
```

And you see above that it works. There's nothing special to using HDLC.

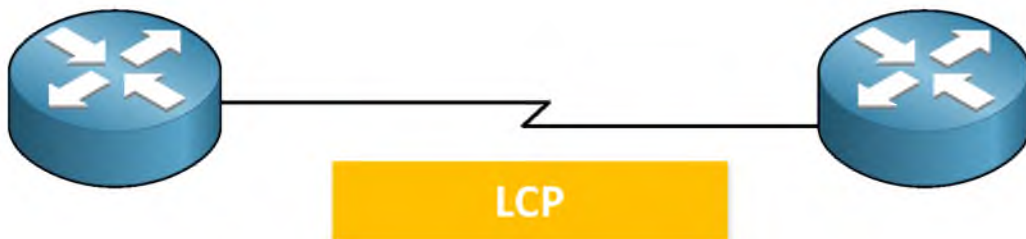
**PPP (Point-to-Point Protocol)** is a protocol that I'm going to tell you more about. Still have the OSI-model fresh in mind? Good! I told you WAN protocols operate on the physical and data link layer. I just told you about the physical layer and its serial cables, now it's time for the data link layer.



PPP operates on the data link layer (layer 2) but as you can see the data link layer has been split into two pieces:

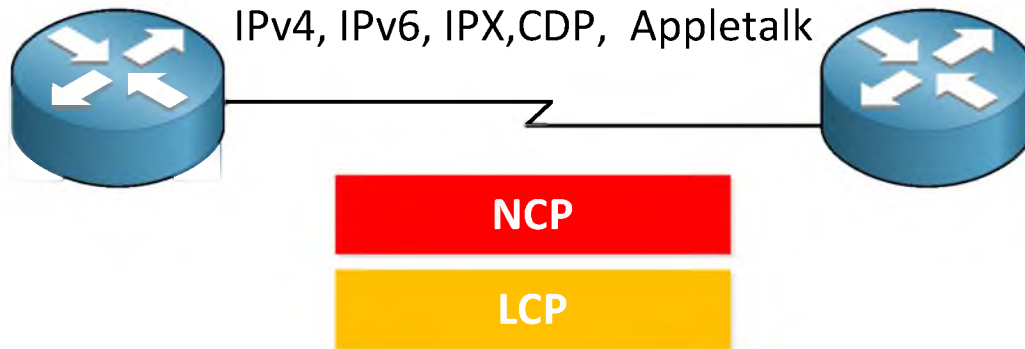
- **NCP: Network Control Protocol**
- **LCP: Link Control Protocol**

And you thought the OSI-model finally made sense? ;)



LCP takes care of setting up the link. If you enable authentication for PPP it will take care of authentication. Once the link has been setup we use NCP.





NCP will make sure you can run different protocols over our PPP link like IP, IPv6 but also CDP (Cisco Discovery Protocol) and older protocols like IPX or AppleTalk.

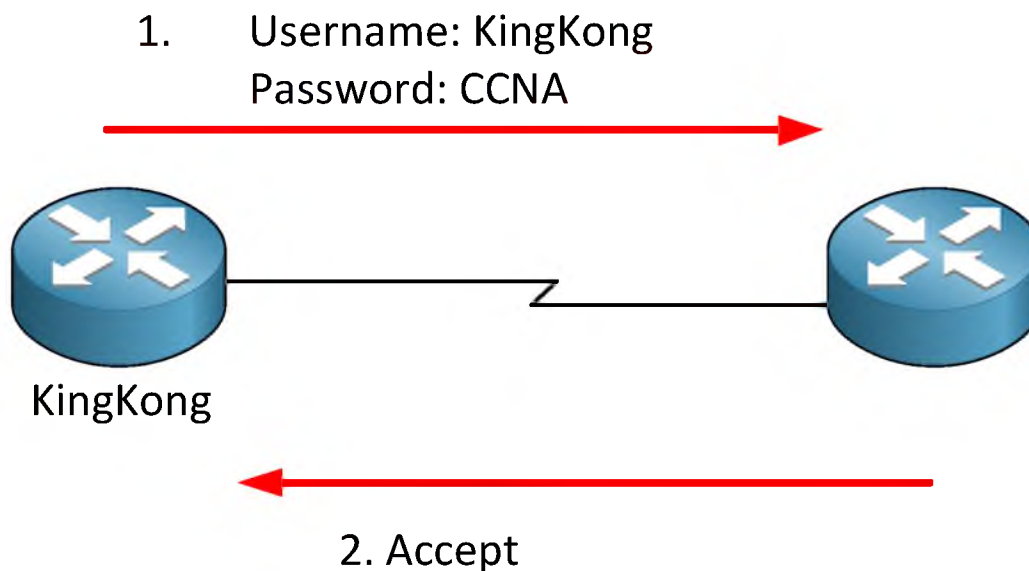
So in short if you enable PPP on both routers this is what happens:

1. LCP: Takes care of setting up the link.
2. (Optional): Authentication.
3. NCP: Makes sure we can send IP and other protocols across our PPP link.

If you want to use authentication for PPP you have two options:

- **PAP (Password Authentication Protocol):** This is plaintext! It will send the username and password over the PPP link and the router on the other side will check it.
- **CHAP (Challenge Authentication Protocol):** Instead of sending the password in plaintext we are going to send a "challenge" which is a hash of the password. This is far more secure.

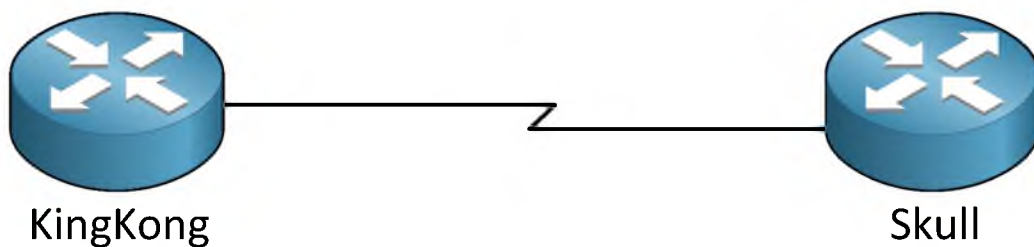
Let me show you an example of PAP:



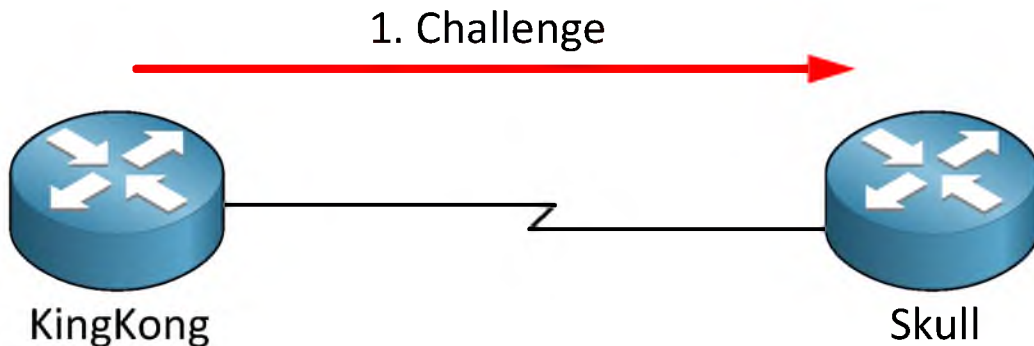
In the example above we are using PAP:

- On the left side we have a router called KingKong.
- PPP will start setting up the link by using LCP.
- Since authentication is enabled our router on the right side will authenticate router KingKong:
  - Router KingKong will send its hostname and the password in **plaintext**.
  - The router on the right side will accept or deny the credentials.
- If everything is ok PPP will work and we are authenticated.

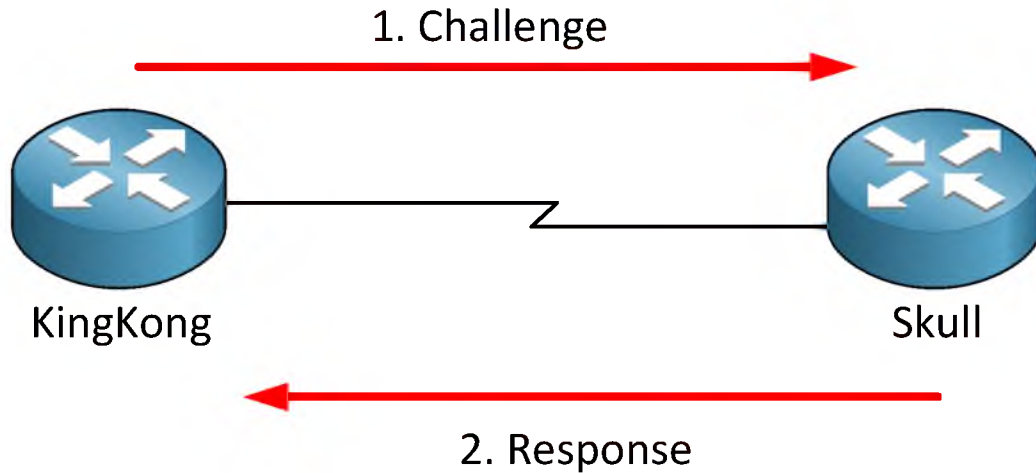
PAP is very simple but sending a username and password in plaintext is not a very secure method. Let's see how CHAP works:



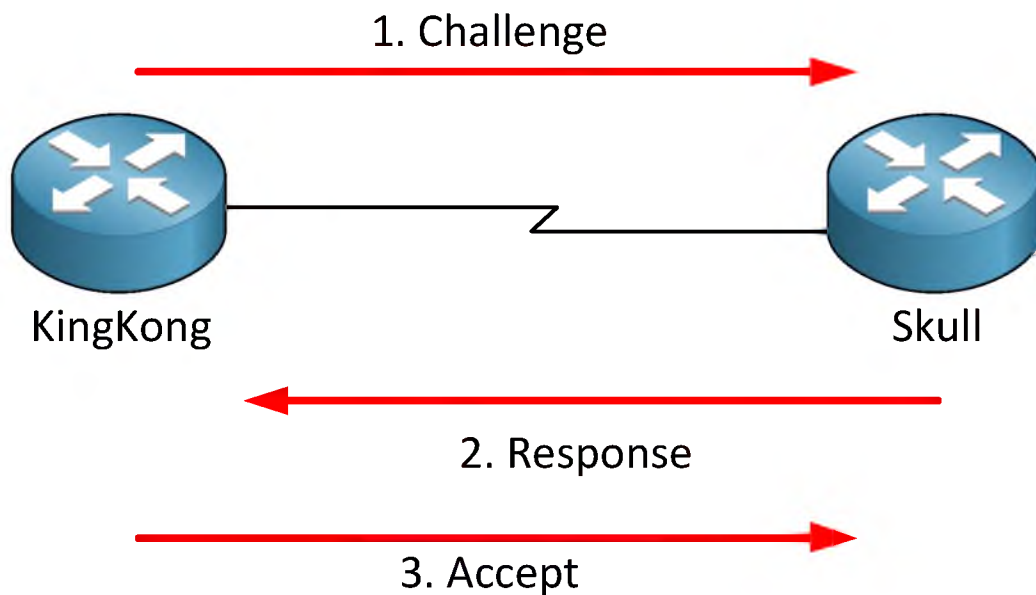
- On the left side we have router KingKong, on the right side is router Skull.
- PPP will start setting up the link by using LCP.
- Router Kingkong will try to authenticate router Skull using CHAP.



I'm going to show you how router Skull is going to authenticate router KingKong. Here you see a challenge is sent from router KingKong towards router Skull.



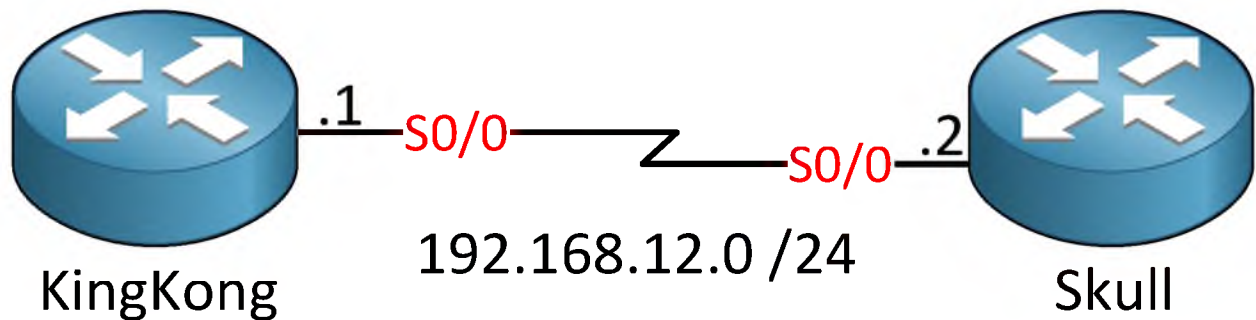
Instead of just sending a password in plaintext router Skull will send a hash. This is far more secure.



If the hash is ok router KingKong will let router Skull know everything is as it should be and our PPP link is working.

Keep in mind authentication is ONLY about checking the username and password, it doesn't have anything to do with encryption. All traffic sent on this PPP connection is in cleartext, if you want encryption you need to use a VPN.

Let's see what PPP looks like on some real routers. I'll be using the same two routers I used for HDLC:



First we'll configure the serial interfaces so that they use PPP:

```
KingKong(config)#interface serial 0/0
KingKong(config-if)#encapsulation ppp
```

```
Skull(config)#interface serial 0/0
Skull(config-if)#encapsulation ppp
```

Use the **encapsulation ppp** command to change the serial interface to PPP (remember that HDLC is the default). Let's configure an IP address on the interfaces:

```
KingKong(config)#interface serial 0/0
KingKong(config-if)#ip address 192.168.12.1 255.255.255.0
```

```
Skull(config)#interface serial 0/0
Skull(config-if)#ip address 192.168.12.2 255.255.255.0
```

Now before I type "no shutdown" I will enable a debug so you can see the LCP building the link:

```
Skull#debug ppp negotiation
PPP protocol negotiation debugging is on
```

The **debug ppp negotiation** command will show you how the PPP link is established. Let's activate the interfaces:

```
KingKong(config)#interface serial 0/0
KingKong(config-if)#no shutdown
```

```
Skull(config)#interface serial 0/0
Skull(config-if)#no shutdown
```

No shutdown will activate our interfaces, and this is what you will see if you enabled the debug:

```
Skull#
%LINK-3-UPDOWN: Interface Serial0/0, changed state to up
Se0/0 PPP: Using default call direction
Se0/0 PPP: Treating connection as a dedicated line
Se0/0 PPP: Session handle[9C000002] Session id[2]
Se0/0 PPP: Phase is ESTABLISHING, Active Open
Se0/0 LCP: O CONFREQ [Closed] id 4 len 10
Se0/0 LCP:   MagicNumber 0x0045BF0E (0x05060045BF0E)
Se0/0 LCP: I CONFREQ [REQsent] id 136 len 10
Se0/0 LCP:   MagicNumber 0x004598FC (0x0506004598FC)
Se0/0 LCP: O CONFACK [REQsent] id 136 len 10
Se0/0 LCP:   MagicNumber 0x004598FC (0x0506004598FC)
Se0/0 LCP: I CONFACK [ACKsent] id 4 len 10
Se0/0 LCP:   MagicNumber 0x0045BF0E (0x05060045BF0E)
Se0/0 LCP: State is Open
Se0/0 PPP: Phase is FORWARDING, Attempting Forward
Se0/0 PPP: Phase is ESTABLISHING, Finish LCP
Se0/0 PPP: Phase is UP
Se0/0 IPCP: O CONFREQ [Closed] id 1 len 10
Se0/0 IPCP:   Address 192.168.12.2 (0x0306C0A80C02)
Se0/0 CDPCP: O CONFREQ [Closed] id 1 len 4
Se0/0 PPP: Process pending ncp packets
Se0/0 CDPCP: I CONF
Skull#REQ [REQsent] id 1 len 4
Se0/0 CDPCP: O CONFACK [REQsent] id 1 len 4
Se0/0 IPCP: I CONFREQ [REQsent] id 1 len 10
Se0/0 IPCP:   Address 192.168.12.1 (0x0306C0A80C01)
Se0/0 IPCP: O CONFACK [REQsent] id 1 len 10
Se0/0 IPCP:   Address 192.168.12.1 (0x0306C0A80C01)
Se0/0 IPCP: I CONFACK [ACKsent] id 1 len 10
Se0/0 IPCP:   Address 192.168.12.2 (0x0306C0A80C02)
Se0/0 IPCP: State is Open
Se0/0 CDPCP: I CONFACK [ACKsent] id 1 len 4
Se0/0 CDPCP: State is Open
Se0/0 IPCP: Install route to 192.168.12.1
Skull#
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to up
```

Above you first see first the "LCP" messages that are setting up the link. Once LCP is done you see "IPCP" and "CDPCP" messages. This is NCP making sure that we can send IP and CDP traffic over our PPP link. **CDP (Cisco Discovery Protocol)** is used to detect other Cisco devices on our network.

Let's see if there is connectivity between the routers:

```
KingKong#ping 192.168.12.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.12.2, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/12 ms
```

A quick ping proves that PPP is working. Let's disable debug before we continue:

```
Skull#no debug all  
All possible debugging has been turned off
```

Let's see if we can enable authentication for PPP, I'll show you how to use CHAP:

```
KingKong(config)#username Skull password MYSECRET
```

```
Skull(config)#username KingKong password MYSECRET
```

First we'll configure the username and password. The **username is the hostname of the router on the other side**. Of course the password has to be the same.

```
KingKong(config)#interface serial 0/0  
KingKong(config-if)#ppp authentication chap
```

```
Skull(config)#interface serial 0/0  
Skull(config-if)#ppp authentication chap
```

Use **ppp authentication chap** command to enable CHAP authentication. If I enable it on both routers then they will authenticate each other. It's also possible to configure this only on one of the routers. Let me show you what I mean:

```
KingKong#debug ppp authentication  
PPP authentication debugging is on
```

You can enable debugging for PPP authentication, this way we can see what is going on:

```
KingKong(config)#interface serial 0/0  
KingKong(config-if)#shutdown  
KingKong(config-if)#no shutdown
```

A quick shutdown and no shutdown will make sure that our routers have to reconnect and reauthenticate.

Here's what you will see:

```
KingKong#
Se0/0 PPP: Using default call direction
Se0/0 PPP: Treating connection as a dedicated line
Se0/0 PPP: Session handle[C2000009] Session id[33]
Se0/0 PPP: Authorization required
%LINK-3-UPDOWN: Interface Serial0/0, changed state to up
Se0/0 CHAP: O CHALLENGE id 32 len 29 from "KingKong"
Se0/0 CHAP: I CHALLENGE id 3 len 26 from "Skull"
Se0/0 CHAP: I RESPONSE id 32 len 26 from "Skull"
Se0/0 PPP: Sent CHAP LOGIN Request
Se0/0 CHAP: Using hostname from unknown source
Se0/0 CHAP: Using password from AAA
Se0/0 CHAP: O RESPONSE id 3 len 29 from "KingKong"
Se0/0 PPP: Received LOGIN Response PASS
Se0/0 PPP: Sent LCP AUTHOR Request
Se0/0 PPP: Sent IPCP AUTHOR Request
Se0/0 LCP: Received AAA AUTHOR Response PASS
Se0/0 IPCP: Received AAA AUTHOR Response PASS
Se0/0 CHAP: O SUCCESS id 32 len 4
Se0/0 CHAP: I SUCCESS id 3 len 4
Se0/0 PPP: Sent CDPCP AUTHOR Request
Se0/0 CDPCP: Received AAA AUTHOR Response PASS
```

Above you see an "O" that stands for outgoing and the "I" for incoming. Our router sends a challenge, receives a response and sends a success message. It also receives a challenge from router Skull, sends a response and receives a success message.

PPP is also often used by ISPs for DSL connections. The most important reason why they use this is because PPP offers (CHAP) authentication, this can be used to authenticate customers when they try to make a connection with their DSL modem.

DSL however does not create a point-to-point link and couldn't support PPP and CHAP so something had to be done. Since the link between the DSL modem and the router or computer were mostly Ethernet links, a new protocol was created...PPPoE (PPP over Ethernet).

PPPoE creates a tunnel through the DSL connection so that PPP packets can be sent between the customer router and the ISP router. PPPoE is a minor topic on the CCNA blueprint and you might face some questions about the bits that are required to configure it. The configuration has a number of items:

- Configuration of a "dialer" interface where we configure the PPP settings.
- Configuration of CHAP authentication.
- Attaching the dialer interface to the interface facing the DSL modem.

First we will configure a dialer interface that has all PPP settings:

```
Customer(config)#interface dialer 1
Customer(config-if)#encapsulation ppp
Customer(config-if)#ip address negotiated
```

```
Customer(config-if)#dialer pool 1
Customer(config-if)#mtu 1492

Customer(config-if)#ppp authentication chap callin
Customer(config-if)#ppp chap hostname MYUSERNAME
Customer(config-if)#ppp chap password SECRET
```

We will tell the dialer interface to use PPP encapsulation and that we want to receive an IP address from the ISP through PPP by using the **ip address negotiated** command. The dialer pool command tells the dialer interface that when it wants to setup a connection, it has to use a member interface of the **"dialer pool"**. We'll configure the FastEthernet interface as a member of this pool in a bit. The **MTU** has to be set to 1492 instead of the default (1500) to make sure there is enough room for the PPPoE headers.

We also configured the dialer interface to use CHAP authentication with username 'MYUSERNAME' and 'MYSECRET' as password. Not all ISPs will require authentication but if required, this is how to do it. Now we can attach the dialer to an interface:

```
BO1(config)#interface Fa0/0
BO1(config-if-atm-vc)#pppoe-client dial-pool-number 1
```

Above is the interface that is facing the DSL modem. With the **pppoe-client** command we can attach this FastEthernet0/0 interface to the dialer.

I wouldn't worry too much about PPPoE for the exam, just make sure you understand the parts that are required and the commands that I just showed you...

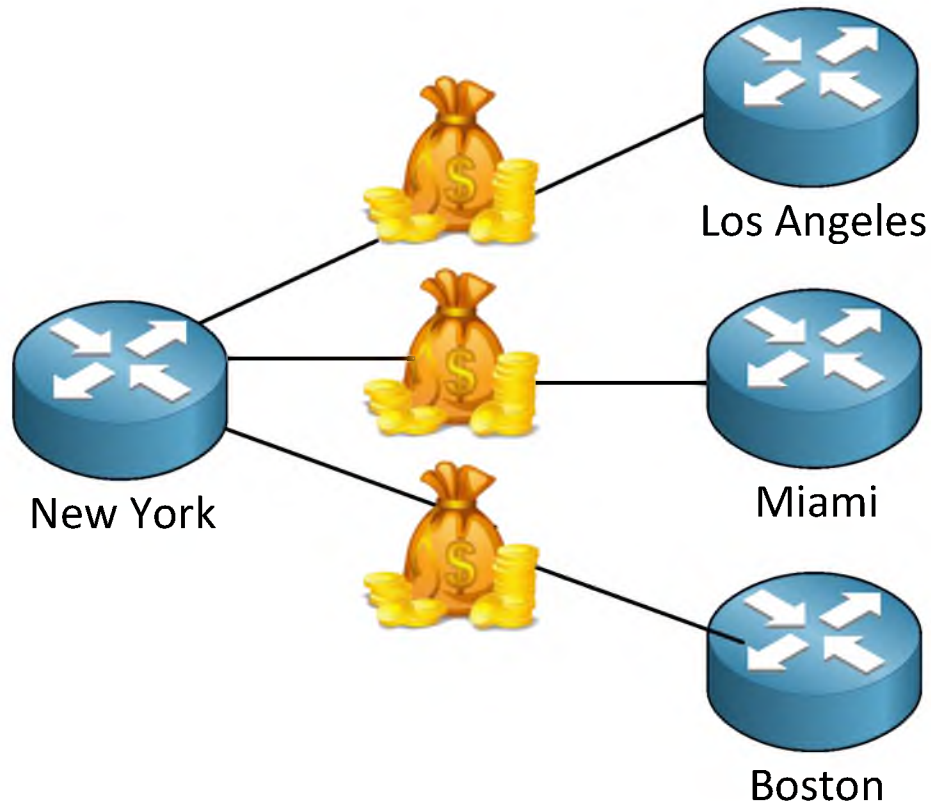
That's all I wanted to share about PPP...only one WAN protocol left and that's Frame-Relay. Frame-Relay is a funky ride!

If you want to learn more about PPP check out this lab, it covers the configuration of PPP and both authentication types:

<http://gns3vault.com/HDLC-PPP/ppp-authentication.html>

Next stop...Frame-Relay!





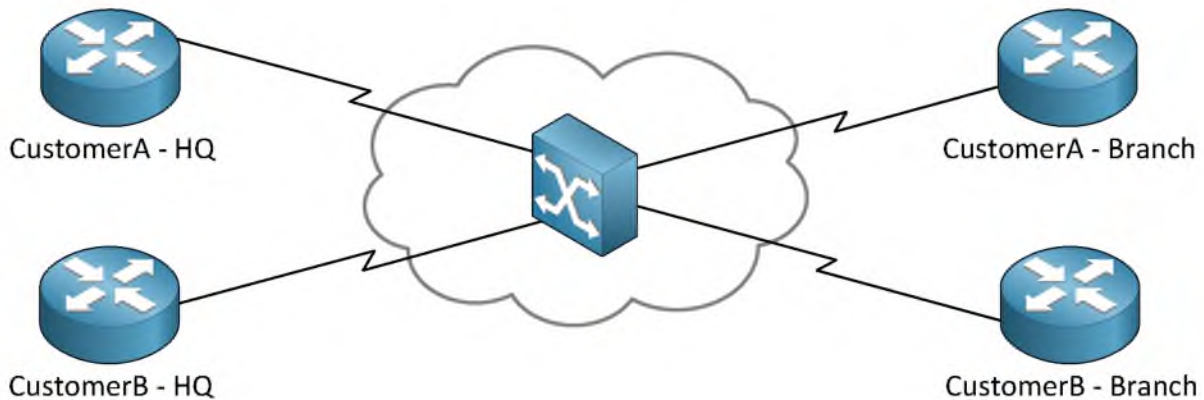
Before we start looking at frame relay let me tell you a little story. We have a network with four sites: there's New York, Los Angeles, Miami and Boston. Since we want connectivity between the sites we have an ISP who sold us three leased lines:

- Between New York and Los Angeles.
- Between New York and Miami.
- Between New York and Boston.

Using leased lines is awesome! You are the only person using these lines since you are paying for them. This means high quality and probably a low chance of congestion (if you have fast links). It's also pretty secure since it's only your traffic that's flowing through these lines.

There are also a few downsides to using leased lines however:

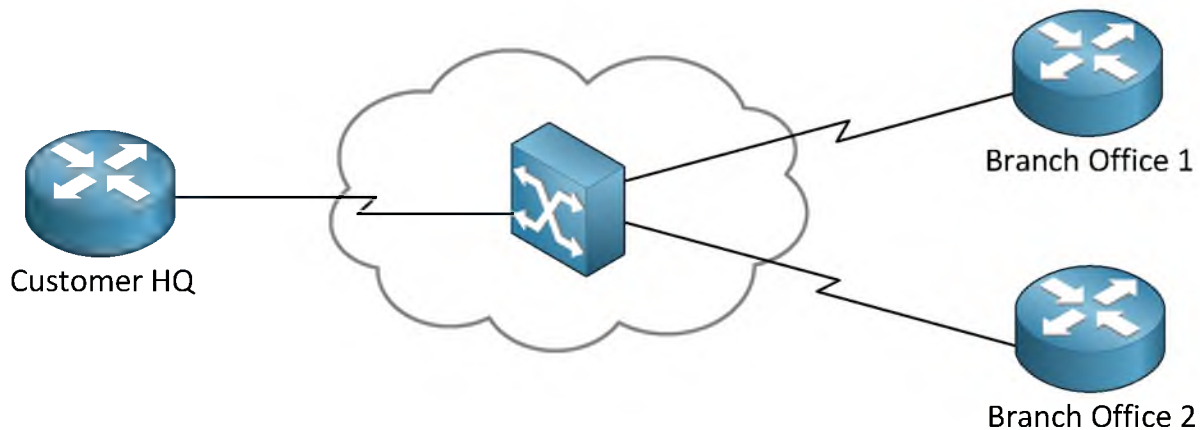
- You are the only one using these lines so you gotta pay for them, being exclusive is fun but expensive.
- On router New York you'll need three interfaces for each leased line, more interfaces means more money.
- What happens if you are going to move the site in New York? It's not always possible to move the leased lines with you.
- What if the New York router crashes? Boston, Miami and Los Angeles will be isolated as well.



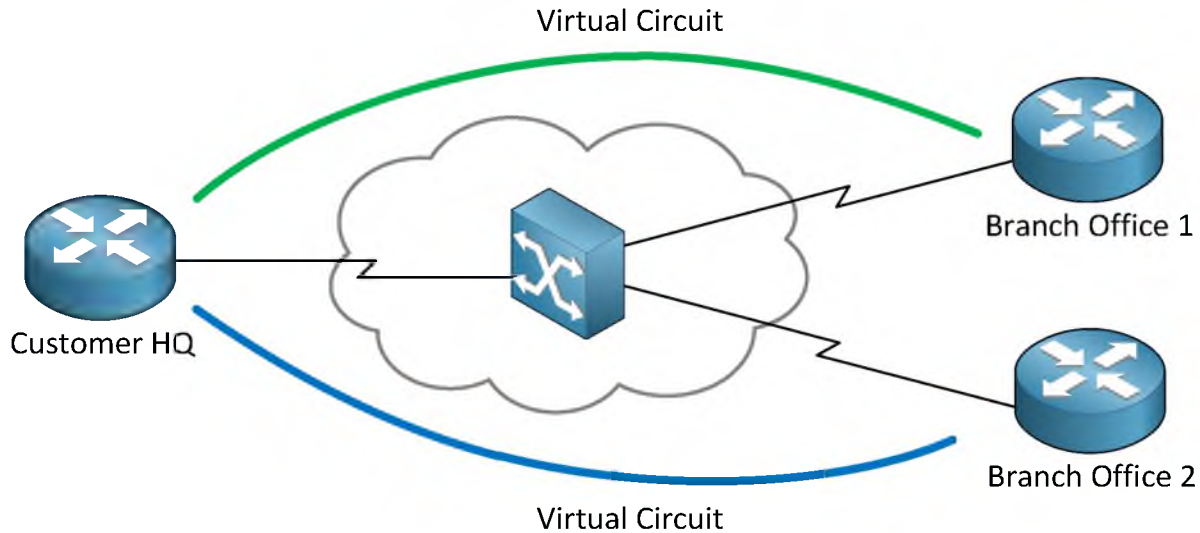
This is a picture of frame relay and it works a bit different. The idea behind frame relay is that you have a single infrastructure from the service provider and multiple customers are connected to it, effectively sharing everything.

In the middle you see a cloud with an icon you probably haven't seen before. This icon is the frame relay switch. The cloud is called the **frame relay cloud** and the reason it has this name is because for us as customer it's unknown what happens in the frame relay cloud. This is the service provider's infrastructure and we really don't care what happens there...we are the customer and all we want is connectivity!

What else do you see? There are two customers (A and B) and each of them has a HQ (Headquarters) and a branch office.



One more picture, here's a frame relay network with three routers from one company. There's a router at the headquarters and we have two branch offices. All of them are connected to the frame relay cloud.



We call our service provider since we want connectivity and the first question they'll ask us is which sites should be connected?. In the example above you can see two **virtual circuits**, the green and blue one. With frame relay there's a difference between the physical and logical connections. The physical connection is just the serial cable which is connected to the provider. Our logical links are virtual circuits. As you can see there is a virtual circuit from branch Office 1 to the HQ router and another one from branch office 2 to the HQ routers.

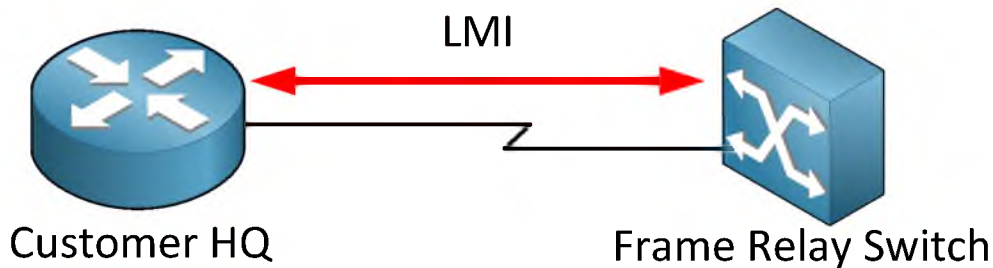
This means that we can send traffic through our virtual circuits between:

- Branch office 1 and HQ.
- Branch office 2 and HQ.

There is no virtual circuit between branch office 1 and branch office 2. Does this mean there is no connectivity between them? No you can still have connectivity between them by sending data to the HQ router! Of course you can get another virtual circuit between branch office 1 and branch office 2 but you'll have to pay for it. Virtual circuits are also called **PVC (Permanent Virtual Circuit)**.

You also pay for a certain speed called the **CIR (Committed Information Rate)**. The cool thing about frame relay is that when no other customers are using the frame relay network it's possible you get a higher speed than what you paid for...the CIR however is a speed that is guaranteed.

How do we know if a PVC is working or not?



Frame-relay uses something called **LMI** which stands for **Local Management Interface**.

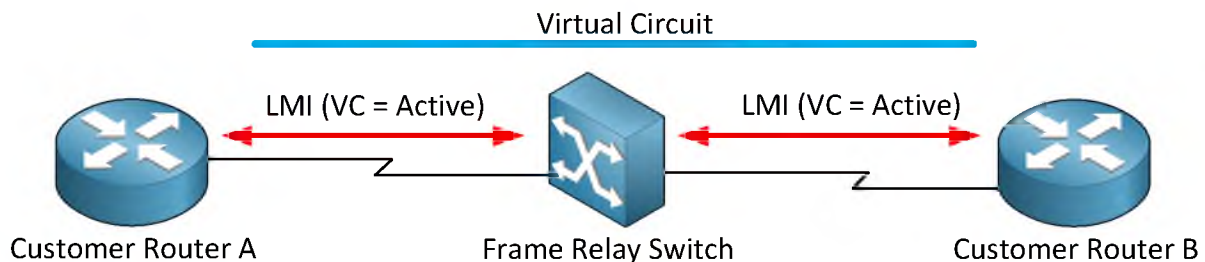
LMI has two functions:

- It's a **keepalive** mechanism.
- It tells us if the PVC is active or inactive.
- It also gives us a **DLCI** (Data Link Connection Identifier). I'll get back to this in a bit

There are 3 types of LMI. They all do the same thing but there are three standards which are not compatible with each other. Whatever you choose make sure it's the same between two devices:

- Cisco
- ANSI T1.617 Annex D
- ITU-T Q.933 Annex A

So if you pick Cisco on one side, use Cisco on the other side as well.

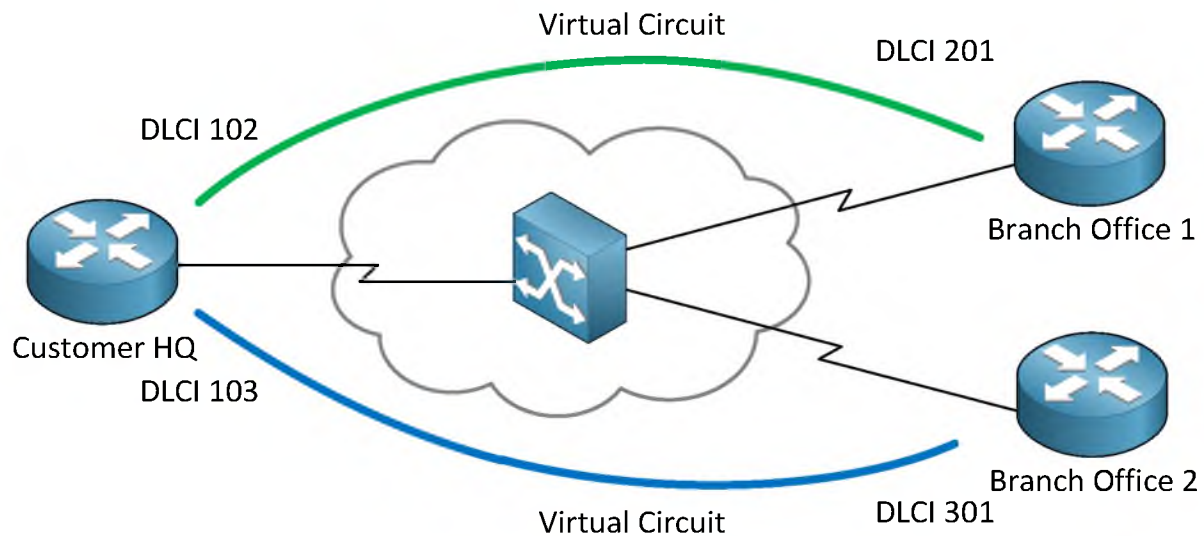


Here's an example of LMI in action. In the middle we have the frame relay switch. LMI packets are sent between Router A and the frame relay switch and router B and the frame relay switch. The frame relay switch tells our routers that the PVC is active.

What else do you need to know about frame relay? Once again I'm going to throw the OSI-model at you:



I explained before that WAN protocols describe the physical (layer 1) and data link (layer 2) layer. What does frame relay use on the data link layer? We don't use MAC addresses since that's Ethernet but we do have something else called a **DLCI (Data Link Connection Identifier)**.





For each PVC you will get a DLCI per router. In our example above you can see that for the PVC between router HQ and branch office 1 we have DLCI 102 on the HQ router and DLCI 201 on the branch office 1 router.

Between router HQ and router branch office 2 we have DLCI 103 on HQ and DLCI 301 on branch office 2. Our DLCI is nothing more but a unique identifier for the data link layer per PVC.

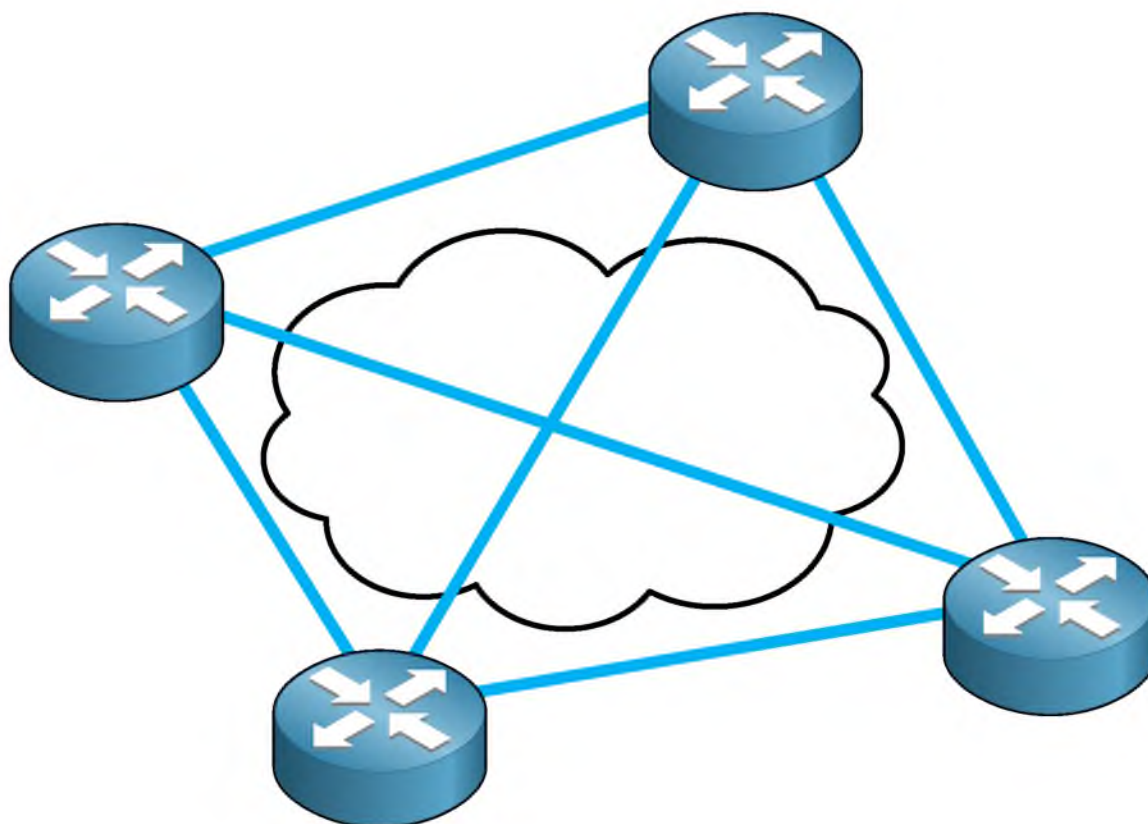


Now there is an important concept to grasp and remember about DLCI. DLCI's are only **locally known** to the router! Your router does **not know** the DLCI of the router on the other side. This is different if you compare it to Ethernet. In our Ethernet world you need to know the MAC address of the computer on the other side in order to send something to it.

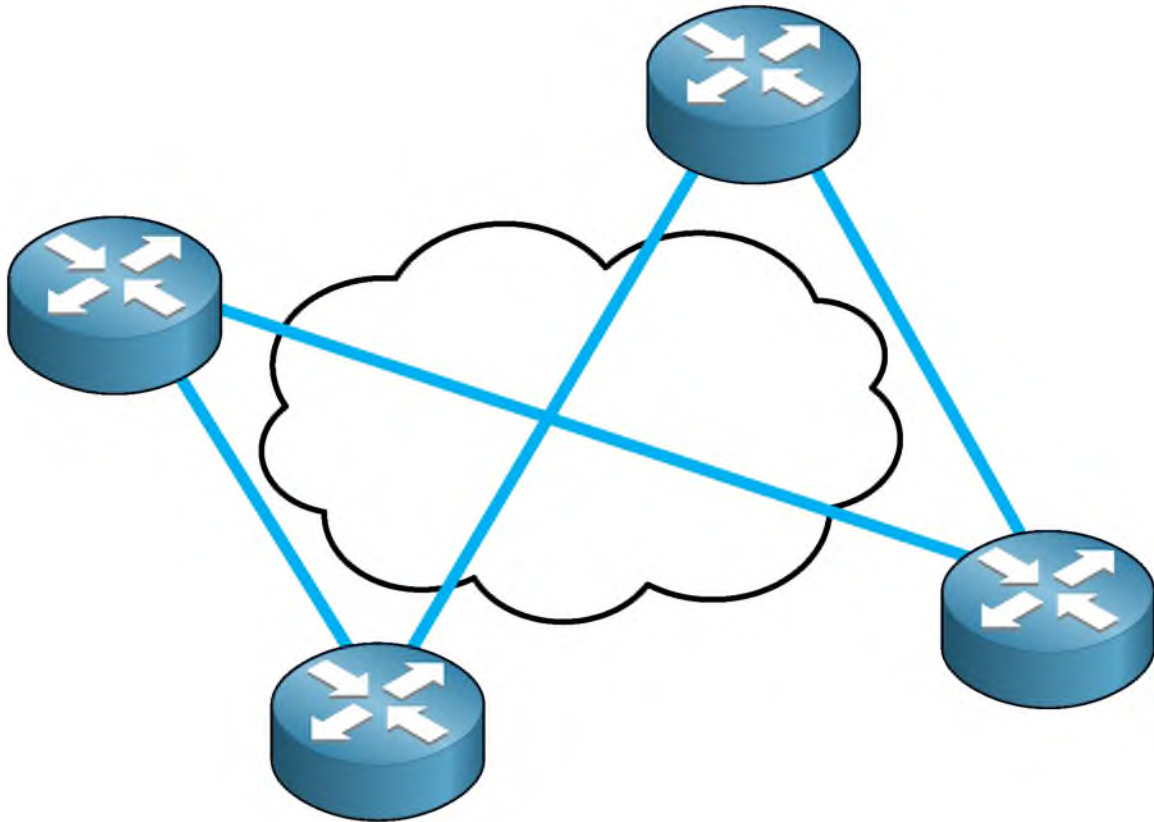
This is just like taking a train. If you are at the train station you walk to the correct train platform and take the train. You have no idea on which train platform you will arrive and you don't care.

Frame-relay supports multiple topologies:

- **Full-mesh**
- **Partial-mesh**
- **Hub and Spoke**

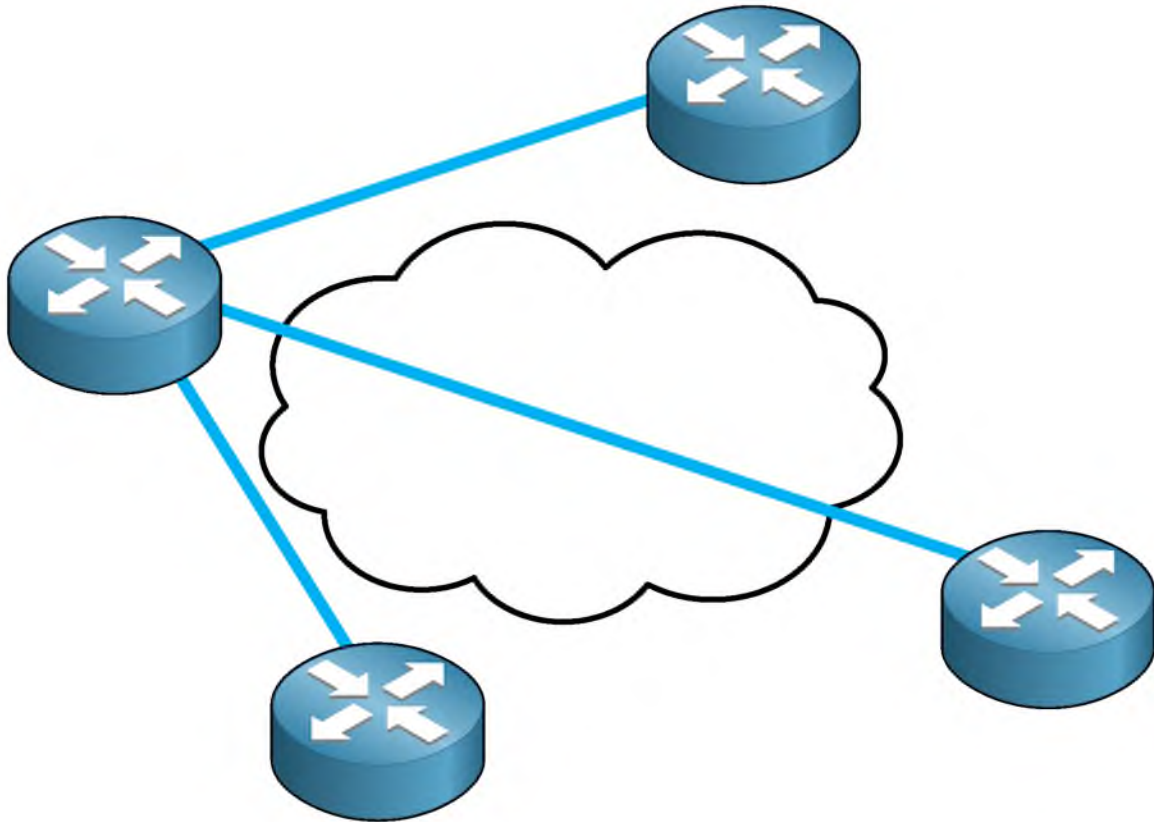


This is our full-mesh topology. As you can see there is a PVC between every router.



This is partial-mesh. The more important routers will have multiple connections to others.



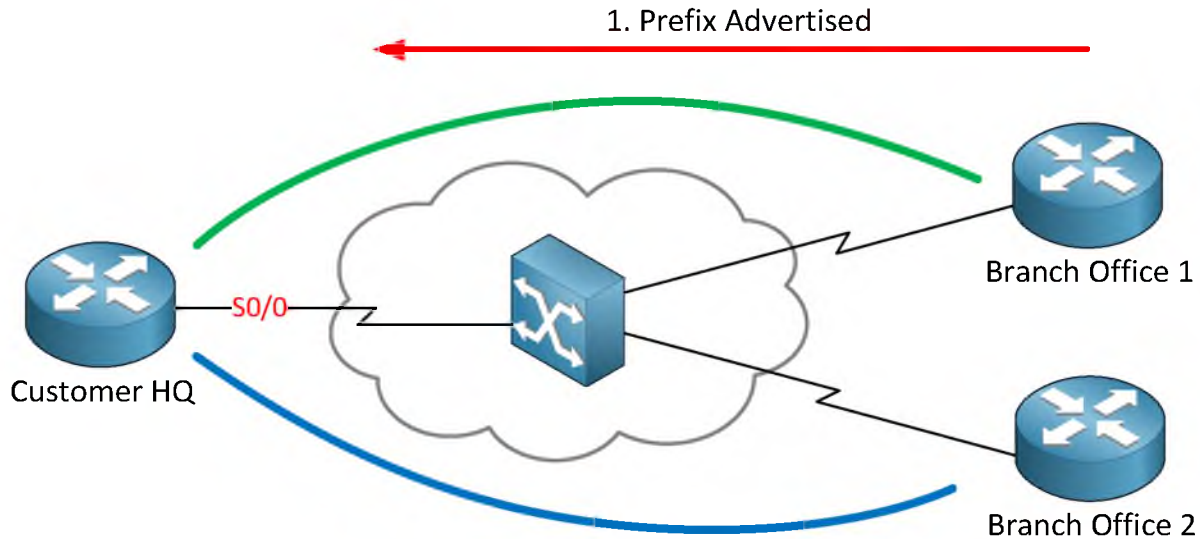


This is the hub and spoke model. The router on the left is our hub and the other routers are spokes. If the spokes want to communicate with each other they'll have to send traffic towards the hub router.

## Frame-relay is **NBMA (non-broadcast multi-access)**

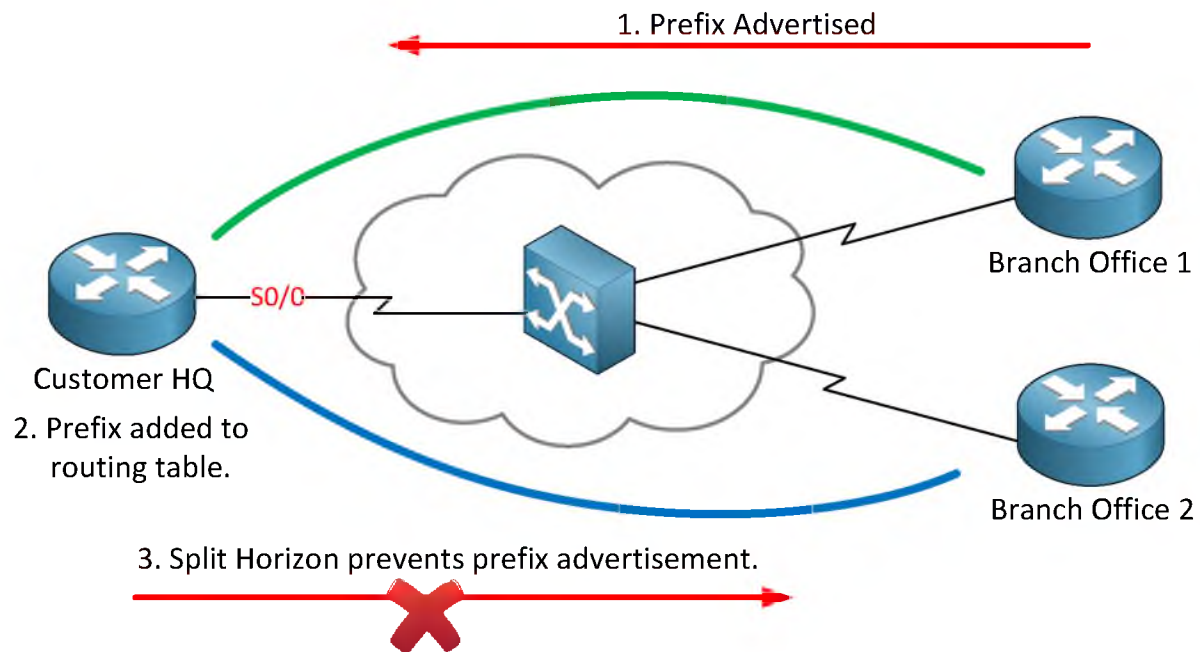
Keep this in mind. What it means is that frame relay is multi-access since all routers can access the network but you are unable to send broadcasts over the frame relay network. No broadcast also means you are unable to send multicast traffic. No multicast means you'll be in trouble with routing protocols. Rip version 2, OSPF and EIGRP all use multicast. Does this mean you can't use routing protocols with frame relay? Well no but it's a bit tricky:

- OSPF and EIGRP can also use unicast instead of multicast.
- There is a method to "emulate" broadcasts over your frame relay network.



What other problems might we encounter with frame relay and routing? Do you remember the characteristics of distance vector routing protocols?

In the picture above I have configured EIGRP on all the routers. Router branch office 1 is sending routing information towards router Customer HQ who will store the prefix in its routing table.

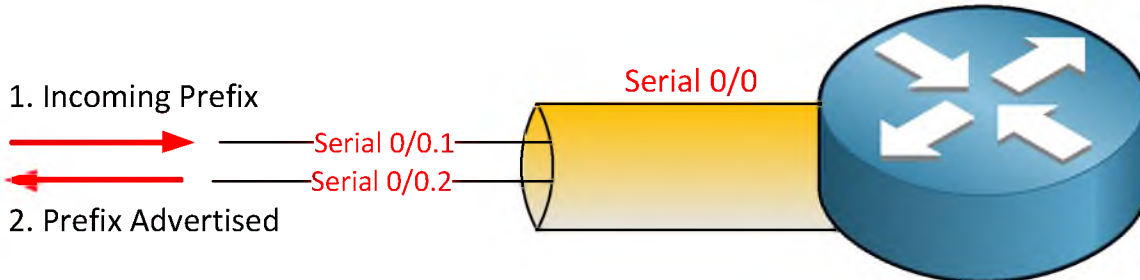


Do you remember the split-horizon rule? Whatever you learn from your neighbor you don't advertise back to them. To be more specific: **whatever you learn on an interface you don't advertise it back out on the same interface.**

We are using two PVC's but on router HQ there is still only one physical interface. Split-horizon will prevent the advertisement of routing information towards router branch office 2.

How can we solve this problem?

- You can disable split horizon (the default on physical interfaces).
- You can use sub-interfaces.

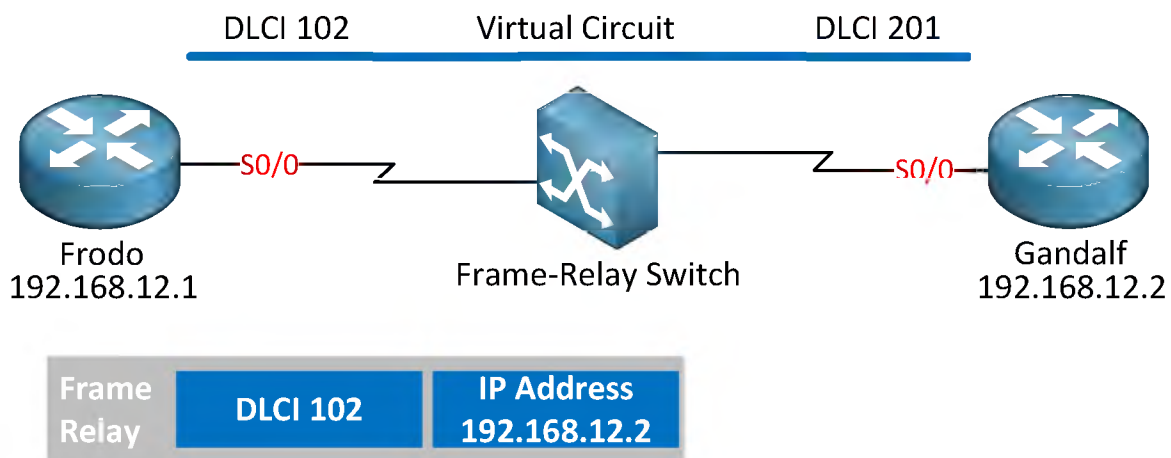


If you use a sub-interface you don't have the split-horizon problem since you are learning routing information on serial0/0.1 and advertising it out of serial0/0.2

Frame-relay can use **point-to-point** sub-interfaces or **point-to-multipoint** sub-interfaces. If you use point-to-point it will solve your split-horizon problem but you'll need to use a different IP subnet per PVC. Point-to-multipoint means you have the split-horizon problem but you can use a single IP subnet for all PVCs.

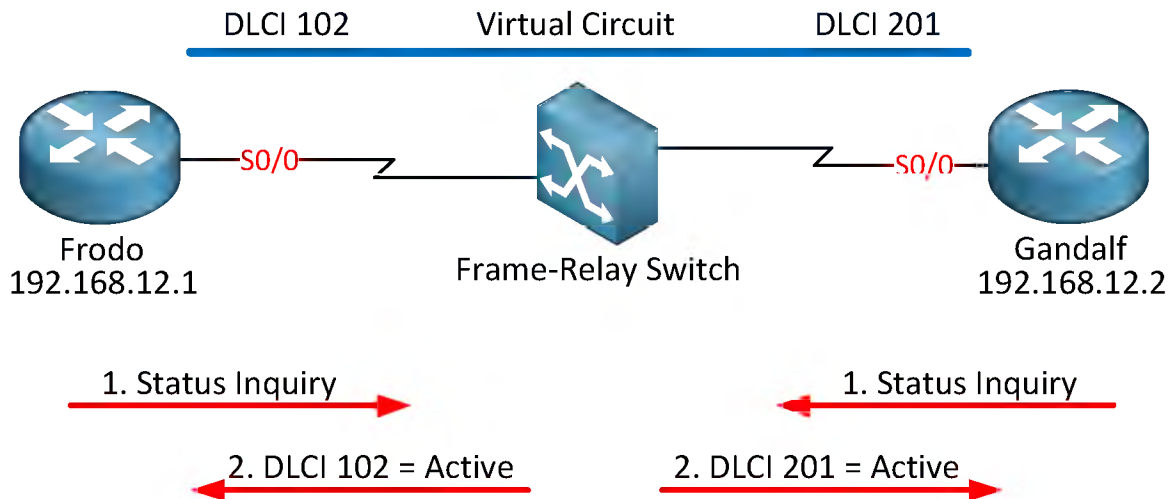
Remember ARP (address resolution protocol)? When we use ARP for Ethernet we need to learn the MAC address of the computer we want to send something to. ARP effectively maps the destination IP address to the destination MAC address.

Frame-relay uses **inverse ARP** and is a bit different. Remember my story about the train platform and how your router only knows it's local DLCI? You don't know the DLCI of the other side. Inverse ARP is going to map your local DLCI to the IP address of the other side:



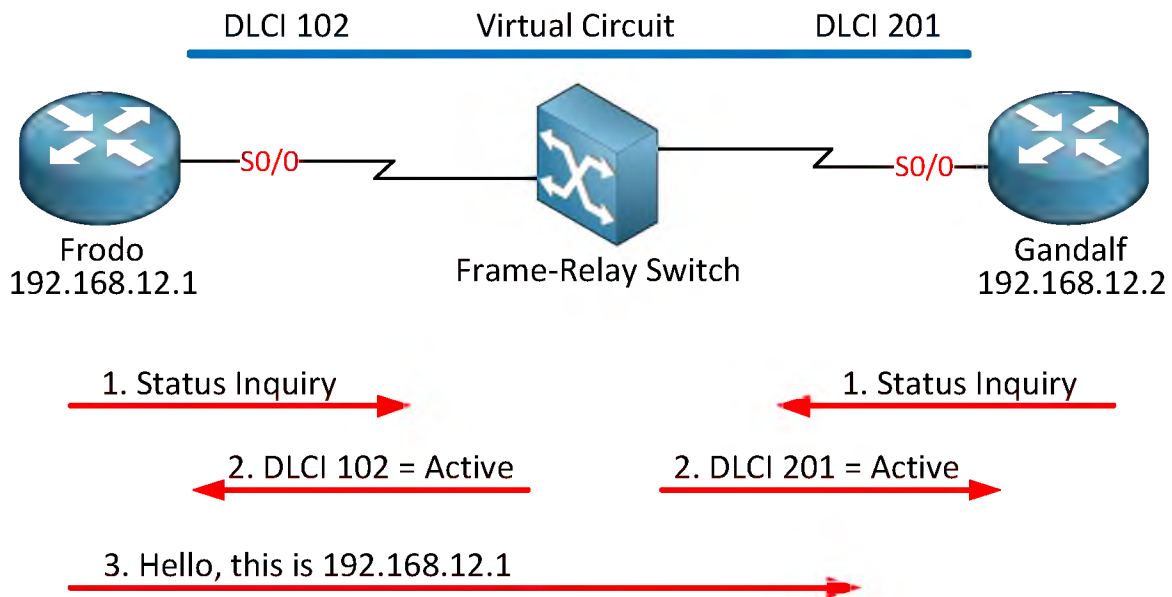
Router Frodo in my example above has mapped the IP address of router Gandalf (192.16.12.2) to its local DLCI 102. That's inverse ARP.

Let's see it in more detail:

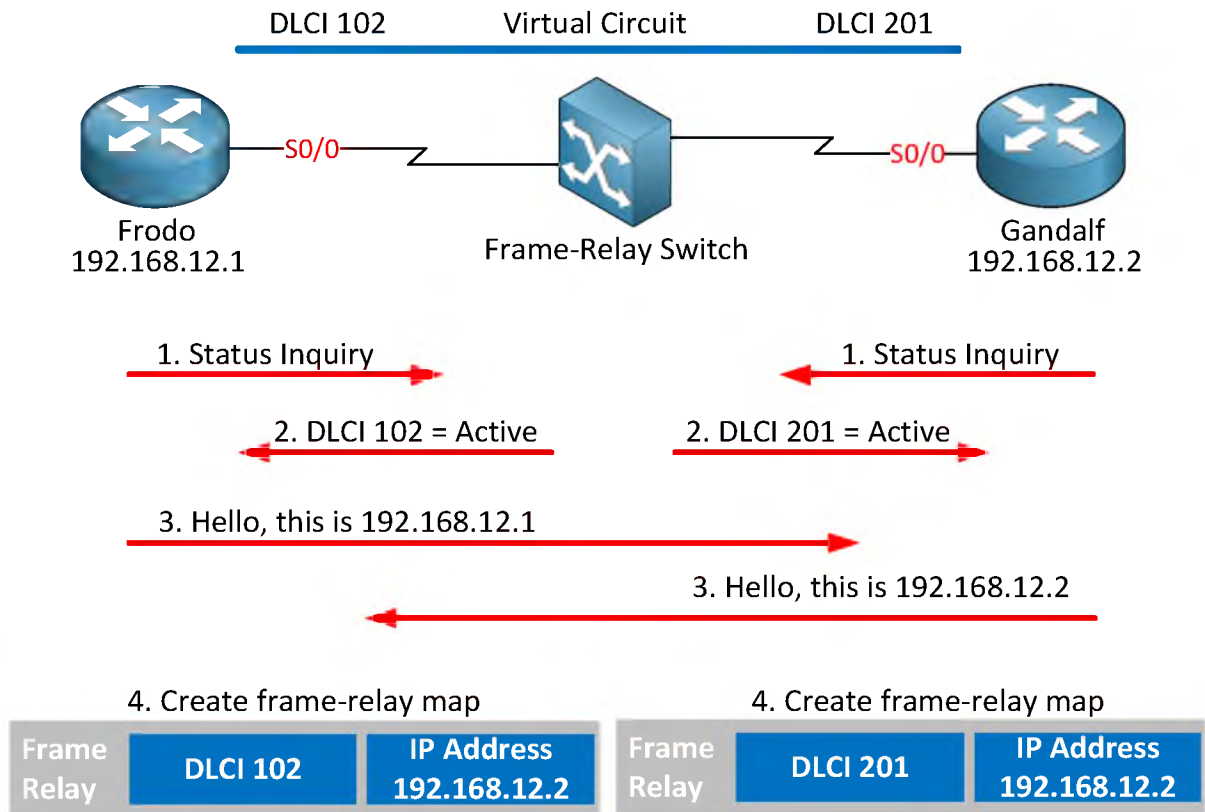


When we configure frame relay here's what happens:

1. Our router will do a status enquiry using LMI.
2. The frame relay switch will give us our DLCI number (or you can configure it yourself).

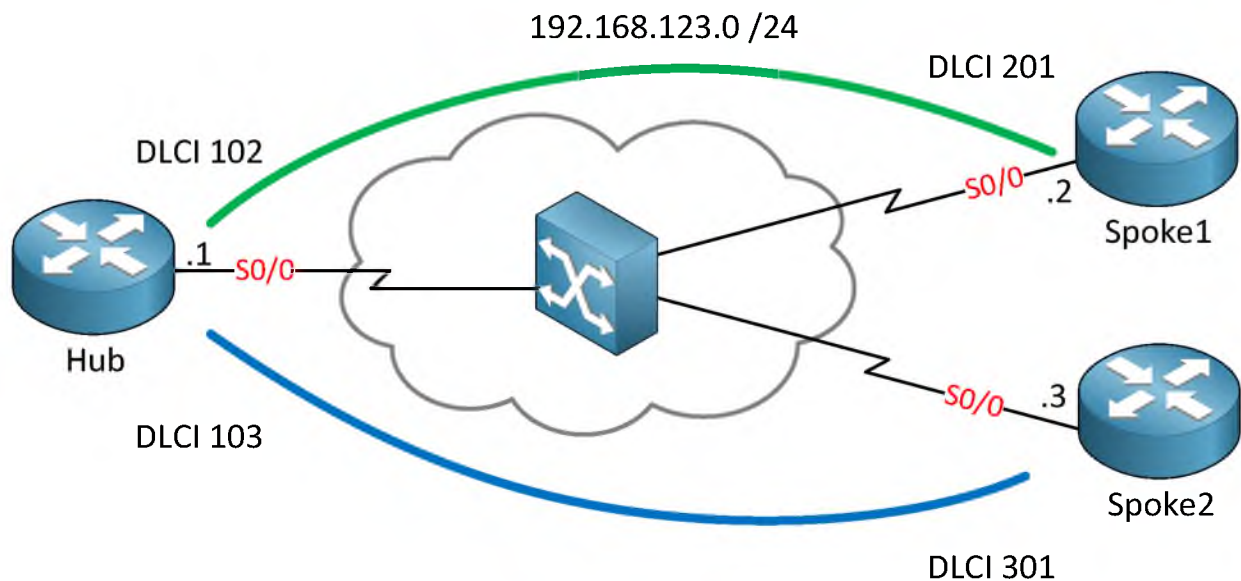


Once our routers know the PVC is active they will send a hello message with their IP address. In my example you only see router Frodo sending this but of course router Gandalf will also send its IP address.



Router Frodo will now know it can reach IP address 192.168.12.2 by sending traffic through the PVC with DLCI 102. Router Gandalf will know that it can reach IP address 192.168.12.1 through the PVC with DLCI 201.

Now you know how frame-relay works. Let's take a look at the actual configuration on our routers!



Above is the topology that we'll use. 3 routers in a hub and spoke model. There are two PVCs and you can see the DLCI numbers in the picture. I'm using a single subnet (192.168.123.0 /24) so we will start with frame-relay point-to-multipoint.



*Configuring a frame-relay switch is outside the scope of the CCNA, CCNP and even the CCIE exam. If you use GNS3 you can use the built-in frame-relay switch emulator.*

Let's prepare the interfaces:

```
Hub(config)#interface s0/0  
Hub(config-if)#encapsulation frame-relay
```

```
Spoke1(config)#interface serial 0/0  
Spoke1(config-if)#encapsulation frame-relay
```

```
Spoke2(config)#interface serial 0/0  
Spoke2(config-if)#encapsulation frame-relay
```

We'll change the encapsulation type to frame-relay for all interfaces.

Let's verify if our PVCs are working first:

```
Hub#show frame-relay pvc

PVC Statistics for interface Serial0/0 (Frame Relay DTE)

      Active      Inactive      Deleted      Static
Local          2             0             0             0
Switched       0             0             0             0
Unused         0             0             0             0

DLCI = 102, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial0/0

input pkts 12          output pkts 11          in bytes 1108
out bytes 1074         dropped pkts 0          in pkts dropped 0
out pkts dropped 0     out bytes dropped 0
in FECN pkts 0        in BECN pkts 0         out FECN pkts 0
out BECN pkts 0       in DE pkts 0           out DE pkts 0
out bcast pkts 1      out bcast bytes 34
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
pvc create time 00:15:37, last time pvc status changed 00:15:37

DLCI = 103, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial0/0

input pkts 12          output pkts 11          in bytes 1108
out bytes 1074         dropped pkts 0          in pkts dropped 0
out pkts dropped 0     out bytes dropped 0
in FECN pkts 0        in BECN pkts 0         out FECN pkts 0
out BECN pkts 0       in DE pkts 0           out DE pkts 0
out bcast pkts 1      out bcast bytes 34
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
pvc create time 00:15:41, last time pvc status changed 00:15:41
```

The **show frame-relay pvc** command tells us that the PVCs are active. You can also see the DLCI numbers this way. This tells us that layer 2 of our frame-relay is working.

In case of trouble it might be a good idea to verify LMI:

```
Hub#show frame-relay lmi

LMI Statistics for interface Serial0/0 (Frame Relay DTE) LMI TYPE = ANSI
Invalid Unnumbered info 0      Invalid Prot Disc 0
Invalid dummy Call Ref 0      Invalid Msg Type 0
Invalid Status Message 0      Invalid Lock Shift 0
Invalid Information ID 0      Invalid Report IE Len 0
Invalid Report Request 0      Invalid Keep IE Len 0
Num Status Enq. Sent 147      Num Status msgs Rcvd 148
Num Update Status Rcvd 0      Num Status Timeouts 0
Last Full Status Req 00:00:35      Last Full Status Rcvd 00:00:35
```

Use **show frame-relay lmi** to see the LMI information. It tells us that we are currently using the ANSI type. It doesn't matter which one you use as long as it's the same on all routers.



Since layer 2 is working we'll configure some IP addresses and see if we can get layer 3 working:

```
Hub(config)#interface s0/0
Hub(config-if)#ip address 192.168.123.1 255.255.255.0
```

```
Spoke1(config)#interface serial 0/0
Spoke1(config-if)#ip address 192.168.123.2 255.255.255.0
```

```
Spoke2(config)#interface serial 0/0
Spoke2(config-if)#ip address 192.168.123.3 255.255.255.0
```

Let's see if we can reach the other side:

```
Hub#ping 192.168.123.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.123.2, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/8/24 ms
```

```
Hub#ping 192.168.123.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.123.3, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/8 ms
```

As you can see the hub router can reach both spoke routers. This is because Inverse ARP is enabled by default.

We can check the frame-relay maps to confirm this:

```
Hub#show frame-relay map
Serial0/0 (up): ip 192.168.123.2 dlci 102(0x66,0x1860), dynamic,
broadcast,, status defined, active
Serial0/0 (up): ip 192.168.123.3 dlci 103(0x67,0x1870), dynamic,
broadcast,, status defined, active
```

```
Spoke1#show frame-relay map
Serial0/0 (up): ip 192.168.123.1 dlci 201(0xC9,0x3090), dynamic,
broadcast,, status defined, active
```

```
Spoke2#show frame-relay map
Serial0/0 (up): ip 192.168.123.1 dlci 301(0x12D,0x48D0), dynamic,
broadcast,, status defined, active
```

Above you see the mappings between the IP address and the DLCI number. There are two other interesting things to see here. The keyword **dynamic** means that the entry was learned because of inverse ARP. The keyword **broadcast** means that we can send broadcast or multicast through our PVC. Let's disable Inverse ARP and create some mappings ourselves:



```
Hub(config)#interface serial 0/0
Hub(config-if)#no frame-relay inverse-arp
```

```
Spoke1(config)#interface serial 0/0
Spoke1(config-if)#no frame-relay inverse-arp
```

```
Spoke2(config)#interface serial 0/0
Spoke2(config-if)#no frame-relay inverse-arp
```

Use **no frame-relay inverse-arp** to disable it...

```
Hub#clear frame-relay inarp
```

```
Spoke1#clear frame-relay inarp
```

```
Spoke2#clear frame-relay inarp
```

And we'll use **clear frame-relay inarp** to get rid of the current frame-relay maps that were created using inverse ARP.

You'll see that connectivity is now impossible:

```
Hub#ping 192.168.123.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.123.2, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

```
Hub#ping 192.168.123.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.123.3, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

The router has no idea where to send its IP packets to since we don't have any frame-relay maps.

Let's create them ourselves:

```
Hub(config)#interface serial 0/0
Hub(config-if)#frame-relay map ip 192.168.123.2 102 broadcast
Hub(config-if)#frame-relay map ip 192.168.123.3 103 broadcast
```

```
Spoke1(config)#interface serial 0/0
Spoke1(config-if)#frame-relay map ip 192.168.123.1 201
```

```
Spoke2(config)#interface serial 0/0
Spoke2(config-if)#frame-relay map ip 192.168.123.1 301
```

Use the **frame-relay map** command to map the IP address of your neighbor to your own DLCI number. The keyword **broadcast** is optional. We require it on the hub router if you want to run a routing protocol like OSPF or EIGRP over your frame-relay network.



*Routing protocols like OSPF or EIGRP use multicast so that's why you require the "broadcast" keyword. It's also possible to configure these routing protocols to use unicast traffic using the "neighbor" command in your OSPF or EIGRP configuration.*

Let's see if we have any frame-relay maps now:

```
Hub#show frame-relay map
Serial0/0 (up): ip 192.168.123.2 dlci 102(0x66,0x1860), static,
                broadcast,
                CISCO, status defined, active
Serial0/0 (up): ip 192.168.123.3 dlci 103(0x67,0x1870), static,
                broadcast,
                CISCO, status defined, active
```

Above you can see that they are still there. The word **static** tells us that this is a frame-relay map that we configured ourselves.

So can we ping again now?

```
Hub#ping 192.168.123.2

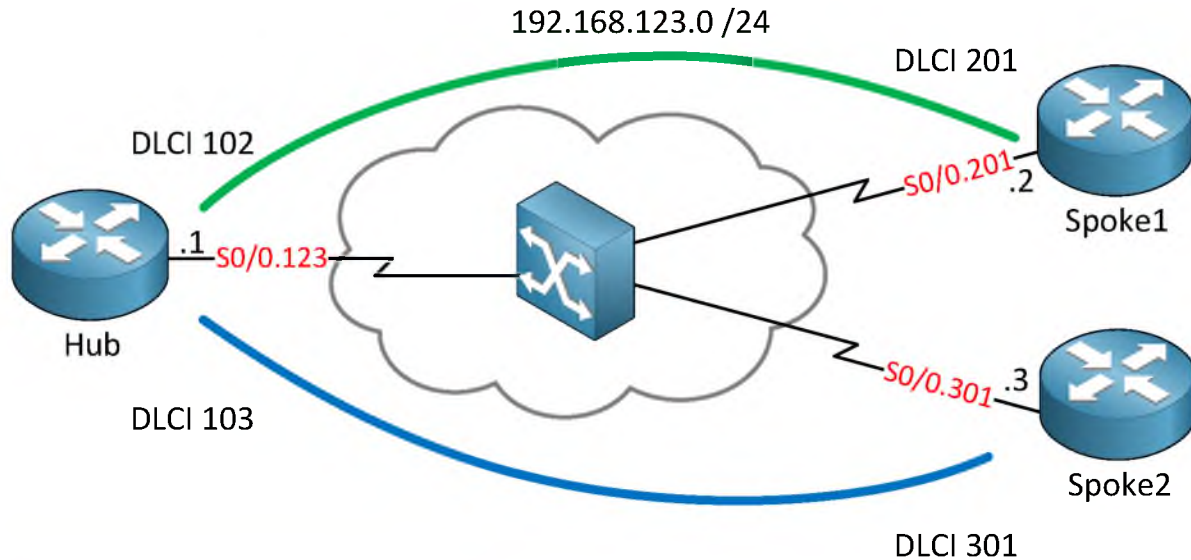
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.123.2, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/8/24 ms
```

```
Hub#ping 192.168.123.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.123.3, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/8 ms
```

No issues here...great! So now you have seen how to configure frame-relay point-to-multipoint, how to check PVCs, LMI and the frame-relay maps with or without Inverse ARP.

I showed you this using the physical interfaces but we can also use sub-interfaces, let me show you how this works:



```
Hub(config)#default interface serial 0/0
Building configuration...
```

Interface Serial0/0 set to default configuration

```
Spoke1(config)#default interface serial 0/0
Building configuration...
```

Interface Serial0/0 set to default configuration

```
Spoke2(config)#default interface serial 0/0
Building configuration...
```

Interface Serial0/0 set to default configuration

The **default interface** command is nice to reset your interface configuration. Now let's configure the interfaces:

```
Hub(config)#interface s0/0
Hub(config-if)#encapsulation frame-relay
```

```
Spoke1(config)#interface serial 0/0
Spoke1(config-if)#encapsulation frame-relay
```

```
Spoke2(config)#interface serial 0/0
Spoke2(config-if)#encapsulation frame-relay
```

On the physical interfaces I still have to configure that we are using frame-relay.

Now I can create sub-interfaces:

```
Hub(config)#interface serial 0/0.123 multipoint
Hub(config-subif)#ip address 192.168.123.1 255.255.255.0
Hub(config-subif)#frame-relay map ip 192.168.123.2 102 broadcast
Hub(config-subif)#frame-relay map ip 192.168.123.3 103 broadcast
```

```
Spoke1(config)#interface serial 0/0.201 multipoint
Spoke1(config-subif)#ip address 192.168.123.2 255.255.255.0
Spoke1(config-subif)#frame-relay map ip 192.168.123.1 201 broadcast
```

```
Spoke2(config)#interface serial 0/0.301 multipoint
Spoke2(config-subif)#ip address 192.168.123.3 255.255.255.0
Spoke2(config-subif)#frame-relay map ip 192.168.123.1 301 broadcast
```

You can use any sub-interface number you want. I like to use the DLCI number for the sub-interface number but I can't do this on the hub router since it has two DLCI numbers. There's no way for your router to tell which DLCI number belongs to which sub-interface so we need to tell it using the frame-relay map command.

So does our configuration work? Let's send a ping...

```
Hub#ping 192.168.123.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.123.2, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/8/24 ms
```

```
Hub#ping 192.168.123.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.123.3, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/8 ms
```

Excellent this is working! Are you following me so far?

Next step is to configure a distance vector routing protocol so I can demonstrate how to deal with split-horizon. I'll use EIGRP:

```
Hub(config)#router eigrp 1
Hub(config-router)#no auto-summary
Hub(config-router)#network 192.168.123.0
```

```
Spoke1(config)#interface loopback 0
Spoke1(config-if)#ip address 2.2.2.2 255.255.255.0
```

```
Spoke1(config)#router eigrp 1
Spoke1(config-router)#no auto-summary
Spoke1(config-router)#network 192.168.123.0
Spoke1(config-router)#network 2.0.0.0
```

```
Spoke2(config)#router eigrp 1
Spoke2(config-router)#no auto-summary
Spoke2(config-router)#network 192.168.123.0
```

As you can see I have enabled EIGRP and created a loopback interface on router Spoke1 so that we have something to advertise. Let's take a look at our routing tables:

```
Hub#show ip route eigrp
      2.0.0.0/24 is subnetted, 1 subnets
D       2.2.2.0 [90/2297856] via 192.168.123.2, 00:00:30, Serial0/0.123
```

The hub router has learned network 2.2.2.0 /24 from Spoke1. What about Spoke2?

```
Spoke2#show ip route eigrp
```

There's nothing there. This is because split horizon on the Hub router is blocking the advertisement.

Let's solve it:

```
Hub(config)#interface serial 0/0.123
Hub(config-subif)#no ip split-horizon ?
    eigrp   Enhanced Interior Gateway Routing Protocol (EIGRP)
    <cr>

Hub(config-subif)#no ip split-horizon eigrp 1
```

When I'm using EIGRP you can use **no ip split-horizon eigrp** to disable it Let's check spoke2 again:

```
Spoke2#show ip route eigrp
      2.0.0.0/24 is subnetted, 1 subnets
D       2.2.2.0 [90/2809856] via 192.168.123.1, 00:00:21, Serial0/0.301
```

Problem solved! We can now see the entry in the routing table. So is it reachable?

```
Spoke2#ping 2.2.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

Even though it's in the routing table I'm unable to ping it. This is happening because the next hop IP address (192.168.123.2) is unreachable for Spoke2. I only created a frame-relay map to reach the Hub router, not spoke1.

Let's create two additional mappings so that spoke1 and spoke2 can reach each other:

```
Spoke1(config)#interface serial 0/0.201
Spoke1(config-subif)#frame-relay map ip 192.168.123.3 201 broadcast
```

```
Spoke2(config)#interface serial 0/0.301
Spoke2(config-subif)#frame-relay map ip 192.168.123.2 301 broadcast
```

The frame-relay maps above will ensure that the spoke routers can reach each other. Let's try that ping again:

```
Spoke2#ping 2.2.2.2
```

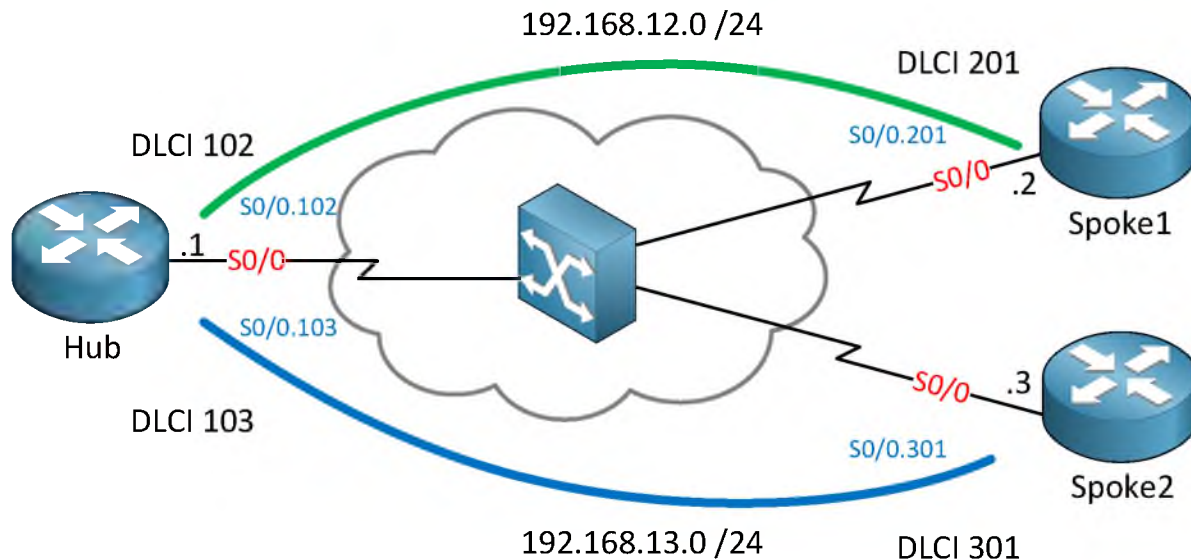
Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 4/9/20 ms

Now you know how to configure frame-relay point-to-multipoint, Let's take a look at the point-to-point topology:



We are using the same topology with one difference. Point-to-point links require an **IP Subnet per PVC**:

- Hub and Spoke1: 192.168.12.0 /24
- Hub and Spoke2: 192.168.13.0 /24



```
Hub(config)#interface serial 0/0
Hub(config-if)#encapsulation frame-relay

Hub(config)#interface serial 0/0.102 point-to-point
Hub(config-subif)#ip address 192.168.12.1 255.255.255.0
Hub(config-subif)#frame-relay interface-dlci 102

Hub(config)#interface serial 0/0.103 point-to-point
Hub(config-subif)#ip address 192.168.13.1 255.255.255.0
Hub(config-subif)#frame-relay interface-dlci 103
```

```
Spoke1(config)#interface s0/0
Spoke1(config-if)#encapsulation frame-relay
Spoke1(config-if)#exit
Spoke1(config)#interface serial 0/0.201 point-to-point
Spoke1(config-subif)#ip address 192.168.12.2 255.255.255.0
Spoke1(config-subif)#frame-relay interface-dlci 201
```

```
Spoke2(config)#interface serial 0/0
Spoke2(config-if)#encapsulation frame-relay
Spoke2(config-if)#exit
Spoke2(config)#interface serial 0/0.301 point-to-point
Spoke2(config-subif)#ip address 192.168.13.3 255.255.255.0
Spoke2(config-subif)#frame-relay interface-dlci 301
```

Here is the configuration for the hub and spoke routers. You only have to specify encapsulation frame-relay on the physical interface. The rest of the commands are on the sub-interfaces. Your router can't read your mind and find out on which sub-interfaces which DLCI's should be so you have to configure it yourself. We don't use the frame-relay map command for point-to-point sub-interfaces but you have to use the **frame-relay interface-dlci** command here.

Let's configure EIGRP:

```
Hub(config)#router eigrp 1
Hub(config-router)#no auto-summary
Hub(config-router)#network 192.168.12.0
Hub(config-router)#network 192.168.13.0
```

```
Spoke1(config)#interface loopback 0
Spoke1(config-if)#ip address 2.2.2.2 255.255.255.0
```

```
Spoke1(config)#router eigrp 1
Spoke1(config-router)#no auto-summary
Spoke1(config-router)#network 192.168.12.0
Spoke1(config-router)#network 2.0.0.0
```

```
Spoke2(config)#router eigrp 1
Spoke2(config-router)#no auto-summary
Spoke2(config-router)#network 192.168.13.0
```

And let's see what we have in the routing tables:

```
Hub#show ip route eigrp
  2.0.0.0/24 is subnetted, 1 subnets
D    2.2.2.0 [90/2297856] via 192.168.12.2, 00:00:23, Serial0/0.102
```

Our Hub router has learned network 2.2.2.0 /24.

```
Spoke1#show ip route eigrp
D    192.168.13.0/24 [90/2681856] via 192.168.12.1, 00:00:56, Serial0/0.201
```

Spoke1 has learned about network 192.168.13.0 /24.

```
Spoke2#show ip route eigrp
D    192.168.12.0/24 [90/2681856] via 192.168.13.1, 00:01:08, Serial0/0.301
  2.0.0.0/24 is subnetted, 1 subnets
D    2.2.2.0 [90/2809856] via 192.168.13.1, 00:01:08, Serial0/0.301
```

And spoke2 knows about network 192.168.12.0 /24 and 2.2.2.0 /24. If you look closely you can see that the next hop IP address is now 192.168.13.1 (the Hub router).

That's all I have for you about frame relay!

Want to try and see if you can configure frame-relay?

<http://gns3vault.com/Frame-Relay/frame-relay-basics.html>

This lab will teach you the basics; see if you can configure EIGRP on top of it!



## 21. Introduction to IPv6

In this chapter we'll take a look at IPv6, the reason why we need it and some of the differences with IPv4. As long as the Internet exists, IPv4 has been the protocol that is used for addressing and routing. The problem with IPv4 however is that we are running out of addresses.

So what happened to IPv4? What went wrong? We have 32-bits which gives us 4,294,967,296 IP addresses. Remember our Class A,B and C story at the beginning of this book? When the Internet was born you would get a Class A,B or C network. Class C gives you a block of 256 IP addresses, a class B is 65,536 IP addresses and a class A even 16,777,216 IP addresses. Large companies like Apple, Microsoft, IBM etc. got one or more Class A networks...but do they really need 16 million IP addresses? Many IP addresses were just wasted.

So we started using VLSM (Variable Length Subnet Mask) so we could use any subnet mask we like and create smaller subnets, no longer just the class A,B or C networks. We also have NAT and PAT so we can have many private IP addresses behind a single public IP addresses.

Nevertheless the Internet has grown in a way nobody expected 20 years ago. Despite all our cool tricks like VLSM and NAT/PAT we needed more IP addresses and so IPv6 was born.

What happened to IPv5? Good question...IP version 5 was used for an experimental project called "Internet Stream Protocol". It's defined in a RFC if you are interested for historical reasons: <http://www.faqs.org/rfcs/rfc1819.html>

IPv6 has 128-bit addresses compared to our 32-bit IPv4 addresses. Keep in mind every additional bit doubles the number of IP addresses...so we go from 4 billion to 8 billion, 16,32,64, etc. Keep doubling until you reach 128-bit. Just for fun I looked up how many IPv6 addresses this will give us:

- 340,282,366,920,938,463,374,607,431,768,211,456

Can we even pronounce this? Let's try this:

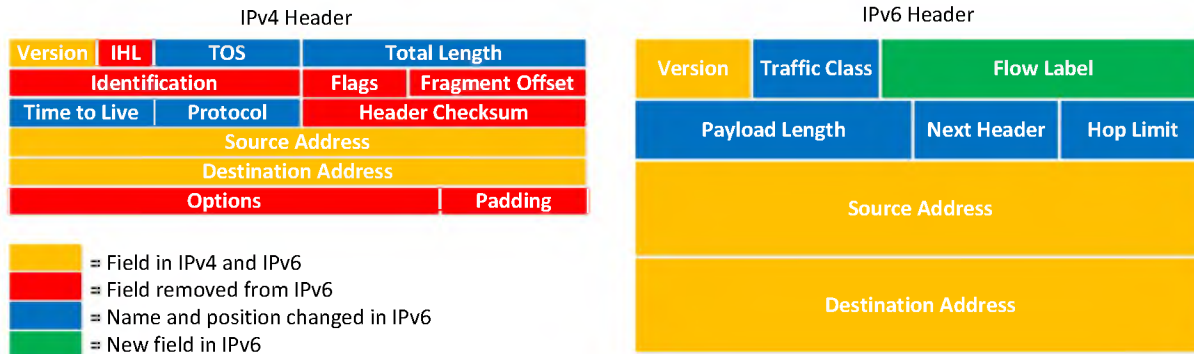
- 340- undecillion
- 282- decillion
- 366- nonillion
- 920- octillion
- 938- septillion
- 463- sextillion
- 463- quintillion
- 374- quadrillion
- 607- trillion
- 431- billion
- 768- million
- 211- thousand
- 456

That's mind boggling... This gives us enough IP addresses for networks on earth, the moon, mars and the rest of the universe. IPv6 addresses are written down in hexadecimal.

IPv4 and IPv6 are **not compatible with each other** so many protocols have been updated or replaced to work with IPv6, here are some examples:

- OSPF has been upgraded from version 2 (IPv4) to version 3 (IPv6).
- ICMP has been upgraded to ICMP version 6.
- ARP has been replaced with **NDP (Neighbor Discovery Protocol)**.

The header of an IPv6 packet contains the source and destination addresses but compared to IPv4 it has become much simpler:



Instead of adding all the fields to the header already, the IPv6 header uses a “next header” that refers to optional headers. Since the header is much simpler routers will have less work to do.

What about routing? Is there a difference between IPv4 and IPv6? Let’s look at the routing options:

- Static Routing
- RIPng
- OSPFv3
- MP-BGP4
- EIGRP

You can still use static routes just like in IPv4, nothing new here. RIP has been upgraded and is now called **RIPng** or **RIP Next Generation** (from the Star Trek TV series).

OSPF for IPv4 is actually version 2 and for IPv6 we have version 3. This is a separate protocol, it only runs IPv6. There are only minor changes made to OSPFv3.

BGP (Border Gateway Protocol) is the routing protocol that glues the Internet together. It’s not a CCNA topic but it’s good to know that it supports IPv6. MP-BGP stands for Multi-Protocol BGP and it can route IPv6.

EIGRP supports IPv6 as well.

Just keep in mind that OSPF and EIGRP support IPv6 but those are separate protocols. If you have a network with IPv4 and IPv6 you will run a routing protocol for IPv4 and another one for IPv6. Running IPv4 and IPv6 at the same time is called **dual stack**.

Since the two protocols are not compatible the future will be a migration from IPv4 to IPv6. This means you will run both protocols on your network and perhaps one day you can pull the plug on IPv4 since the whole Internet has been configured for IPv6.

Let's take a look at the format of an IPv6 address:

2041:0000:140F:0000:0000:0000:875B:131B

First of all it's hexadecimal and it's a much longer than an IPv4 address. There are eight pieces that consist of 4 hex digits each, so a 128-bit address can be represented with 32-bit hex characters. If you are unsure how hexadecimal works, take a look at the table below:

| Hexadecimal | Binary | Hexadecimal | Binary |
|-------------|--------|-------------|--------|
| 0           | 0000   | 8           | 1000   |
| 1           | 0001   | 9           | 1001   |
| 2           | 0010   | A           | 1010   |
| 3           | 0011   | B           | 1011   |
| 4           | 0100   | C           | 1100   |
| 5           | 0101   | D           | 1101   |
| 6           | 0110   | E           | 1110   |
| 7           | 0111   | F           | 1111   |

In hexadecimal we count from 0 to F just like we would count from 0 to 15 in decimal:

- A = 10
- B = 11
- C = 12
- D = 13
- E = 14
- F = 15

Using hexadecimal helps to make our addresses shorter but typing in an IPv6 address is still a lot of work. Imagine calling a friend and asking him if he can ping IPv6 address 2041:0000:140F:0000:0000:0000:875B:131B to see if he can reach his default gateway.

To help us out, it's possible to make IPv6 addresses **shorter**. Here's an example:

- Original: 2041:0000:140F:0000:0000:0000:875B:131B
- Short: 2041:0000:140F::875B:131B

If there is a string of zeroes you can remove them by replacing them with a double colon (::). In the IPv6 address above I removed the zeroes making the address a bit shorter. You can only do this **once**.

We can make this IPv6 address even shorter using another trick:

- Short: 2041:0000:140F::875B:131B
- Shorter: 2041:0:140F::875B:131B

If you have a block with 4 zeroes you can remove them and leave only a single zero there.

We can also remove any leading zeroes:

- Original: 2001:0001:0002:0003:0004:0005:0006:0007
- Short: 2001:1:2:3:4:5:6:7

Let me summarize the rules:

- A string of zeroes can be removed leaving only a colon (::). You can only do this once.
- 4 zeroes can be removed leaving only a single zero.
- Leading zeroes can be removed within a block.

You can't remove all zeroes otherwise your IPv6 device has no idea where to fill in the zeroes to make it 128-bit again.

IPv4 addresses have a subnet mask but instead of typing something like 255.255.255.0 we use a **prefix length** for IPv6. Here is an example of an IPv6 prefix:

2001:1111:2222:3333::/64

This is pretty much the same as using 192.168.1.1 /24. The number behind the / are the number of bits that we use for the prefix. In the example above it means that 2001:1111:2222:3333 is the prefix (64 bits) and everything behind it can be used for hosts.

When calculating subnets for IPv4 we can use the subnet mask to determine the network address and for IPv6 we can do something alike. For any given IPv6 address we can calculate what the prefix is but it works a bit different.

Let me show you what I'm talking about, here's an IPv6 address that could be assigned to a host:

**2001:1234:5678:1234:5678:ABCD:EF12:1234/64**

What part from this IPv6 address is the prefix and what part identifies the host?

| Prefix              | Host                |
|---------------------|---------------------|
| 2001:1234:5678:1234 | 2001:1234:5678:1234 |

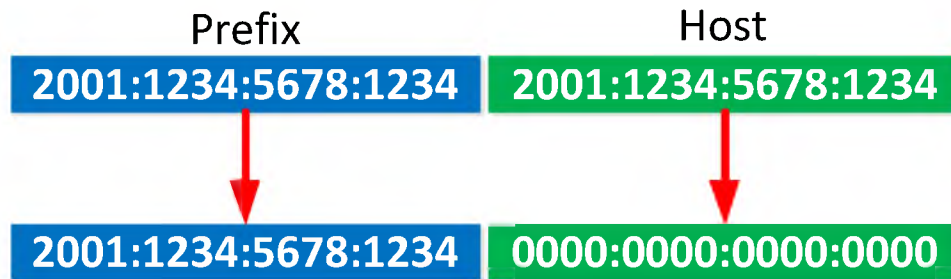
Since we use a /64 it means that the first 64 bits are the prefix. Each hexadecimal character represents 4 binary bits so that means that this part is the prefix:

**2001:1234:5678:1234**

This part has 16 hexadecimal characters. 16 x 4 means 64 bits. So that's the prefix right there. The rest of the IPv6 address identifies the host:

**5678:ABCD:EF12:1234**

So we figured out that "2001:1234:5678:1234" is the prefix part but writing it down like this is not correct. To write down the prefix correctly we need to add 0s at the end of this prefix so that it is a 128 bit address again and add the prefix length:



2001:1234:5678:1234:0000:0000:0000:0000/64 is a valid prefix but we can make it shorter. This string of 0s can be removed and replaced by a single ::

**2001:1234:5678:1234::/64**

That's the shortest way to write down the prefix. Let's look at another example:

**3211::1234:ABCD:5678:1010:CAFE/64**

Before we can see what the prefix is, we should write down the complete address as this one has been shortened (see the :: ?). Just add the 0s until we have a full 128 bit address again:

**3211:0000:0000:1234:ABCD:5678:1010:CAFE/64**

We still have a prefix length of 64 bits. A single hexadecimal character represents 4 binary bits, so the first 16 hexadecimal characters are the prefix:

**3211:0000:0000:1234**

Now we can add 0s at the end to make it a 128 bit address again and add the prefix length:

**3211:0000:0000:1234::/64**

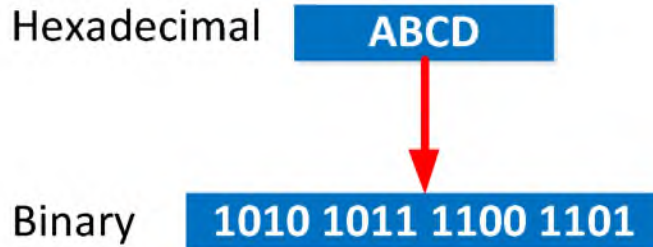
That's a good looking prefix but we can make it a little shorter:

**3211:0:0:1234::/64**

4 zeroes in a row can be replaced by a single one, so "3211:0:0:1234::/64" is the shortest we can make this prefix.

Depending on the prefix length it makes the calculations very easy or (very) difficult. In the examples I just showed you both prefixes had a length of 64. What if I had a prefix length of /53 or something?

Each hexadecimal character represents 4 binary bits. When your prefix length is a multiple of 16 then it's easy to calculate because 16 binary bits represent 4 hexadecimal characters. Here's an illustration:



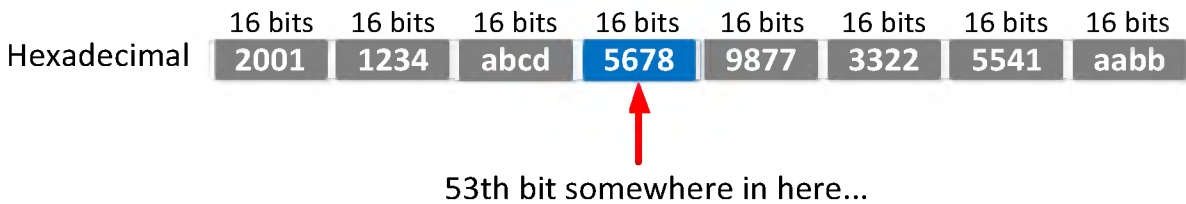
So with a prefix length of 64 we have 4 "blocks" with 4 hexadecimal characters each which makes it easy to calculate. When the prefix length is a multiple of 4 then it's still not too bad because the boundary will be a single hexadecimal character.

When the prefix length is not a multiple of 16 or 4 it means we have to do some binary calculations. Let me give you an example!

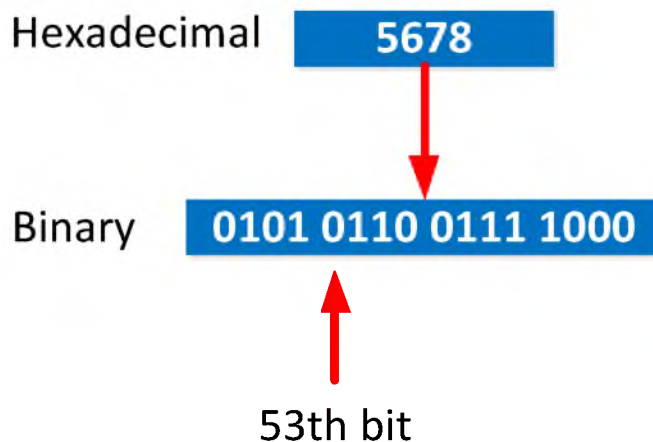
**2001:1234:abcd:5678:9877:3322:5541:aabb/53**

This is our IPv6 address and I would like to know the prefix for this address. Where do I start?

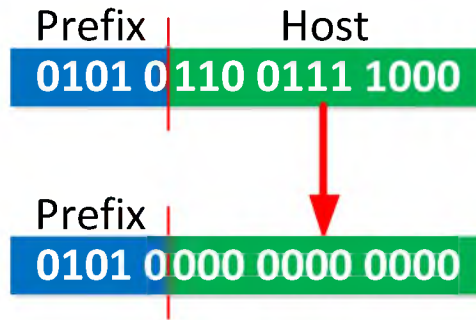
First I have to determine in what "block" my 53th bit is located:



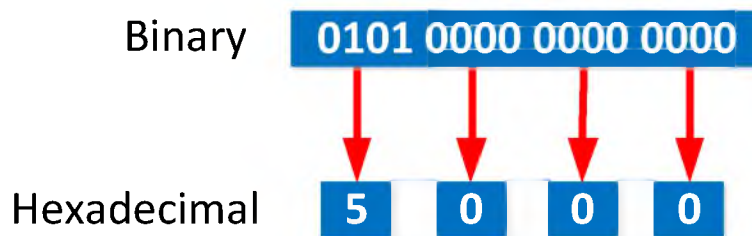
Somewhere in the blue block we will find the 53th bit. To know what the prefix is we will have to calculate those hexadecimal characters to binary:



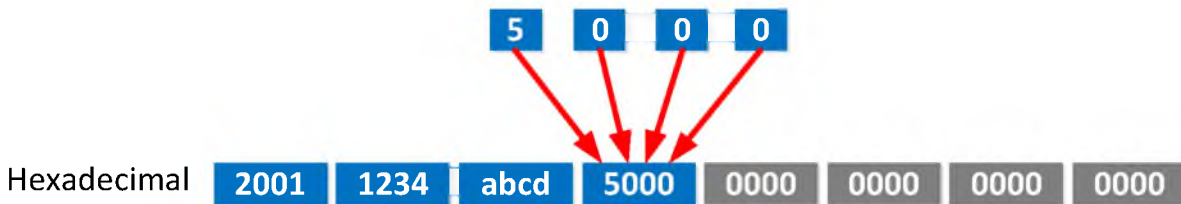
We now have the block that contains the 53th, this is where the boundary is between "prefix" and "host":



Now we will set the host bits to 0 so that only the prefix remains. Finally we calculate from binary back to hexadecimal:



Put this block back into place and set all the other host bits to 0 as well:



We have now found our prefix! 2001:1234:abcd:5000::/53 is the answer. It's not that bad to calculate but you do have to get your hands dirty with binary...

It will take some time to get used to IPv6 addressing and finding the prefixes, but the more you do it the easier it will become.

In the remainder of this chapter we'll talk some more about the different IPv6 addressing types.

IPv4 addresses are organized with a "class system" where Class A,B & C are for unicast IP addresses and class D are for multicast. Most of the IP addresses in these classes are public IP addresses and some are private IP addresses meant for our internal networks.

There is no such thing as classes for IPv6 but IANA did reserve certain IPv6 ranges for specific purposes. We also have private and public IPv6 addresses.

Originally the idea behind IPv4 was that each and every host that is connected to the Internet would have a public IP address. Each company would get a class A,B or C network

and the network engineers at the company would further subnet it so that each host and network device would have a public IP address.

The problem however is that the IPv4 address space was too small and giving out complete A, B or C networks wasn't very wise. Even if you only required a small number of IP addresses you would still receive a class C network which gives you 254 usable IP addresses. A company that required 2,000 IP addresses would get a class B that gives you over 65,000 IP addresses.

Since we were running out of IP addresses we started using things like VLSM (getting rid of the class A, B, C idea) and configured private IP addresses on our LANs and used NAT/PAT instead.

IPv6 offers two options for unicast addresses:

- **Global Unicast**
- **Unique Local**

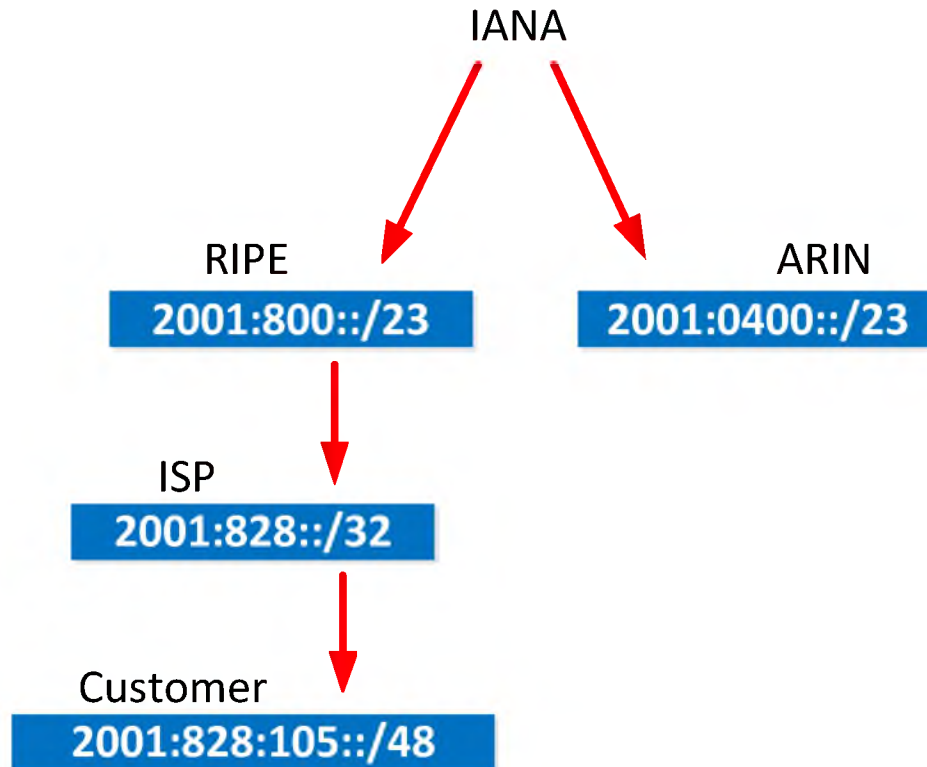


*There used to be a third range of addresses called "site local" that start with FEC0::/10. This range was originally intended to be used on internal networks but has been removed from the IPv6 standard.*

The global unicast IPv6 addresses are similar to IPv4 public addresses. Each company that wants connectivity to the Internet with IPv6 will receive a block of IPv6 addresses that they can further subnet into smaller prefixes so that all their devices have a unique IPv6 address. The reserved block is called a **global routing prefix**.

Since the IPv6 address space is so huge, everyone can get a global routing prefix. Let's take a look at how the IPv6 addresses prefixes are assigned. Let's say a company receives prefix 2001:828:105:45::/64. How did they get it?





We'll walk through this picture from top to bottom:

- IANA is responsible for the allocation of all IPv6 prefixes. They will assign different blocks to the registries. ARIN is for North America, RIPE is for Europe, Middle East and central Asia. In total there are 5 of these registries. IANA assigns 2001:800::/23 to RIPE and 2001:0400::/23 to ARIN (and many other prefixes).
- An ISP that falls under the registry of RIPE requests a block of IPv6 space. They receive 2001:828::/32 from them that they can use for customers.
- The ISP will further subnet their 2001:828::/32 address space for their customers. In this example the customer receives the 2001:828:105::/48 prefix.

IANA has reserved certain IPv6 address ranges for different purposes, just like they did for IPv4. Originally they reserved IPv6 addresses that start with hexadecimal 2 or 3 as global unicast addresses. This can be written as **2000::/3**. Nowadays they use everything for global unicast that is not reserved for other purposes.

Some of the prefixes that are reserved are:

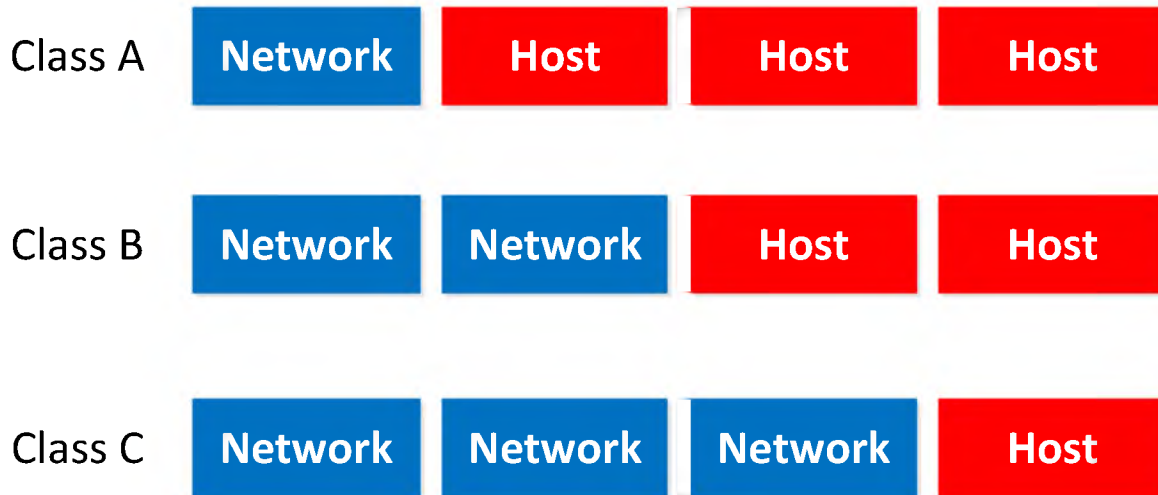
- **FD: Unique Local**
- **FF: Multicast**
- **FE80: Link-Local**

We will discuss the unique local and link-local prefixes later in this chapter.

In my example the customer received 2001:828:105::/48 from the ISP but before I can do anything with this prefix I will have to subnet it for the different VLANs and point-to-point links that I might have. Subnetting for IPv6 is kinda the same as for IPv4 but the math is easier most of the times. Since the address space is so huge almost everyone **uses the**

**/64 prefix for subnets.** It doesn't make sense to use smaller subnets like a /88 or /90 because we have plenty of room when we use IPv6.

Using IPv4 we had a "network" and "host" part and class A, B or C determines how many bits we use for the network part:



When we subnet in IPv4 we take some extra bits from the host part so that we can create more subnets:



And of course as a result we will have fewer hosts per subnet. Subnetting for IPv6 uses a similar structure that looks like this:



The global routing prefix was assigned to you by the ISP and in my example the customer received 2001:828:105::/48. The last 64 bits are called the **interface ID** and this is the equivalent of the host part in IPv4.

This leaves us with 16 bits in the middle that I can use to create subnets. If I want I can steal some more bits from the Interface ID to create even more subnets but there's no need for this.

Using 16 bits we can create 65,536 subnets ...more than enough for most of us. And with 64 bits for the interface ID per subnet, we can have eighteen quintillion, four hundred forty-six quadrillion, seven hundred forty-four trillion, seventy-four billion, seven hundred nine million, five hundred fifty-one thousand, six hundred and something hosts per subnet. That should be more than enough!

Using a 64 bit Interface ID is also very convenient because it cuts your IPv6 address exactly in half!

Let's say our customer with prefix 2001:828:105::/48 wants to create some subnets for his internal network. What addresses can we use?

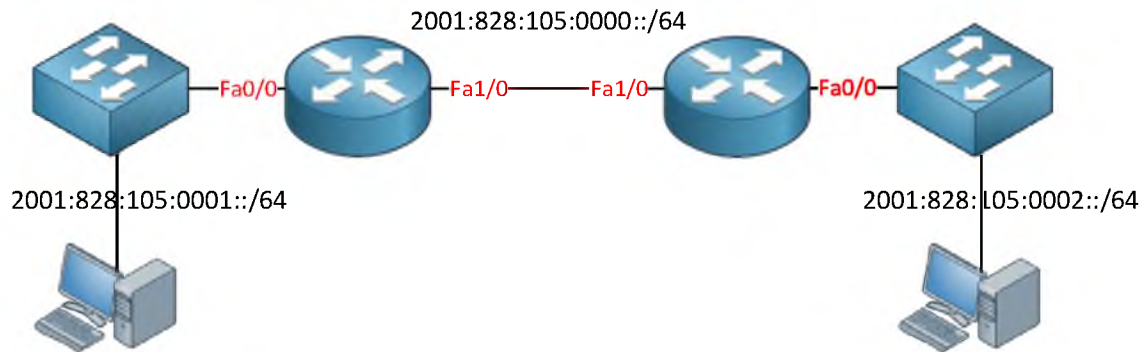


16 bit gives us 4 hexadecimal characters. So all the possible combinations we can make with those 4 characters are our possible subnets. Everything between 0000 and FFFF are valid subnets:

- 2001:828:105:**0000**::/64
- 2001:828:105:**0001**::/64
- 2001:828:105:**0002**::/64
- 2001:828:105:**0003**::/64
- 2001:828:105:**0004**::/64
- 2001:828:105:**0005**::/64
- 2001:828:105:**0006**::/64
- 2001:828:105:**0007**::/64
- 2001:828:105:**0008**::/64
- 2001:828:105:**0009**::/64
- 2001:828:105:**000A**::/64
- 2001:828:105:**000B**::/64
- 2001:828:105:**000C**::/64
- 2001:828:105:**000D**::/64
- 2001:828:105:**000E**::/64
- 2001:828:105:**000F**::/64
- 2001:828:105:**0010**::/64
- 2001:828:105:**0011**::/64
- 2001:828:105:**0012**::/64
- 2001:828:105:**0013**::/64
- 2001:828:105:**0014**::/64
- And so on...

In total there are 65,535 possible subnets so unfortunately I can't add all of them in the book...we can now assign these prefixes to different point-to-point links, VLANs, etc.

Here's an example of what it could look like:

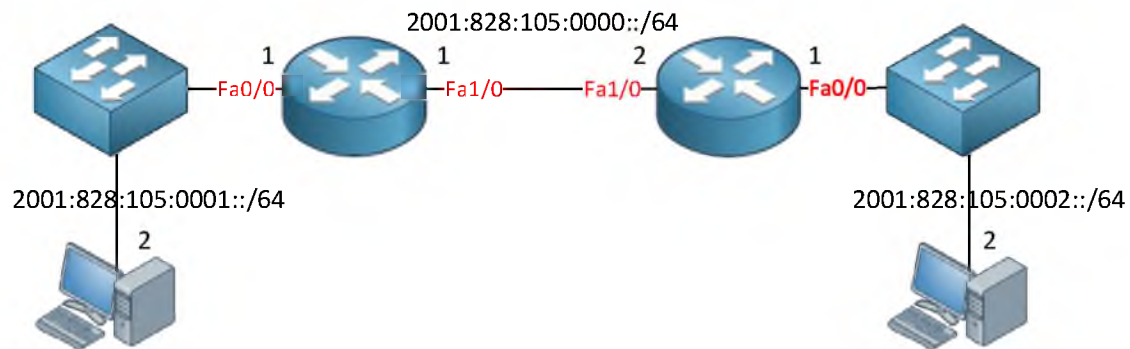


Now you know how the global prefixes work and how to subnet, what about the interface ID? We haven't talked about assigning IPv6 addresses to our hosts yet. Assigning addresses to hosts is almost the same as for IPv4:

- Addresses have to be unique for each host.
- You can't use the prefix address as a host address.

You can configure the IPv6 address **manually** along with a default gateway, DNS server and so on or your hosts can learn an IPv6 address automatically either through DHCP or something new called **SLAAC (Stateless Address Autoconfiguration)**.

Here's an example of IPv6 addresses you could pick for the topology I just showed you:



For the router interfaces I would suggest to use low numbers since they are easy to remember. This example shows a unique global unicast IPv6 address for each device.

This is all I have for you about global unicast addresses, we still have to cover the unique local unicast addresses.

Unique local addresses work like the IPv4 private addresses. You can use these addresses on your own network if you don't intend to connect to the Internet or if you plan to use IPv6 NAT. The advantage of unique local addresses is that you don't need to register at an authority to get some addresses.

You can recognize these addresses because they all start with **FD** in hexadecimal. There are still some rules you have to follow if you want to use unique local addresses:

- Make sure FD are **the first two hexadecimal** characters.
- You need to make up a **40 bit global ID**, you can pick whatever you like.
- Add the 40 bit global ID after the "FD" to **create a 48-bit prefix**.
- The next 16 bits have to be used for subnets.

This leaves you the last 64 bits to use for the interface ID. Here's what the unique local address looks like:



This gives us a unique local address that we can use on our own networks. Subnetting global unicast or unique local addresses is exactly the same with the exception that this time we make up the prefix ourselves instead of the ISP assigning us a global prefix. The global ID can be anything you like, with 40 bits you will have 10 hexadecimal characters to play with. You could pick something like 00 0000 0001, so when you put "FD" in front of it you will have FD00:0000:0001::/48 as your prefix. You can remove some zeroes and make this prefix shorter, it will look like this: FD00:0:1::/48

Now you can add different values behind the prefix to make unique subnets:

- FD00:0:1:**0000**::/64
- FD00:0:1:**0001**::/64
- FD00:0:1:**0002**::/64
- FD00:0:1:**0003**::/64
- FD00:0:1:**0004**::/64
- FD00:0:1:**0005**::/64
- FD00:0:1:**0006**::/64
- FD00:0:1:**0007**::/64
- FD00:0:1:**0008**::/64
- FD00:0:1:**0009**::/64
- FD00:0:1:**000A**::/64
- FD00:0:1:**000B**::/64
- FD00:0:1:**000C**::/64
- FD00:0:1:**000D**::/64
- FD00:0:1:**000E**::/64
- FD00:0:1:**000F**::/64
- FD00:0:1:**0010**::/64
- FD00:0:1:**0011**::/64
- FD00:0:1:**0012**::/64
- FD00:0:1:**0013**::/64
- FD00:0:1:**0014**::/64
- And so on...

When you are doing labs it's a good idea to pick a simple global ID so that you end up with a short and simple to remember prefix. For production networks, it's better to use a global ID that is truly unique. Perhaps one day you want to connect your network to another

network, or perhaps your company buys another company so you have to merge networks. When both networks have the same global ID you will have to renumber IPv6 address for one network, while if the global IDs are different you can just connect them to each other without any issues.

In the remaining of this chapter we'll take a look at how we can configure IPv6 on our routers. When you want to configure an IPv6 address on a router you have two options:

- Manually configure the full 128 bit IPv6 address.
- Use EUI-64.

I'll show you how to manually configure the IPv6 address first and then I'll explain what EUI-64 is all about. This is how you do it:

```
Router(config)#interface fastEthernet 0/0
Router(config-if)#ipv6 address 2001:1234:5678:abcd::1/64
```

You need to use the **ipv6 address** command and then you can type in the IPv6 address. The prefix that I am using is 2001:1234:5678:abcd and this router will have "1" as its "host" address. You can also type the full IPv6 address if you want:

```
Router(config)#interface fastEthernet 0/0
Router(config-if)#ipv6 address 2001:1234:5678:abcd:0000:0000:0000:0001/64
```

This command will have the exact same result. We can verify the subnet and IPv6 address like this:

```
Router#show ipv6 interface fa0/0
FastEthernet0/0 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::C000:18FF:FE5C:0
  No Virtual link-local address(es):
Global unicast address(es):
  2001:1234:5678:ABCD::1, subnet is 2001:1234:5678:ABCD::/64
```

This shows us the global unicast address and our subnet. There is one more important thing when we configure IPv6 on a router. By default, the router will not forward any IPv6 packets and it will not build a routing table. To enable the "processing" of IPv6 packets we need to enable it:

```
Router(config)#ipv6 unicast-routing
```

Most of the "ip" commands will work, just try "ipv6" instead and see what it does:

```
Router#show ipv6 interface brief
FastEthernet0/0          [up/up]
  FE80::C000:18FF:FE5C:0
  2001:1234:5678:ABCD::1
```

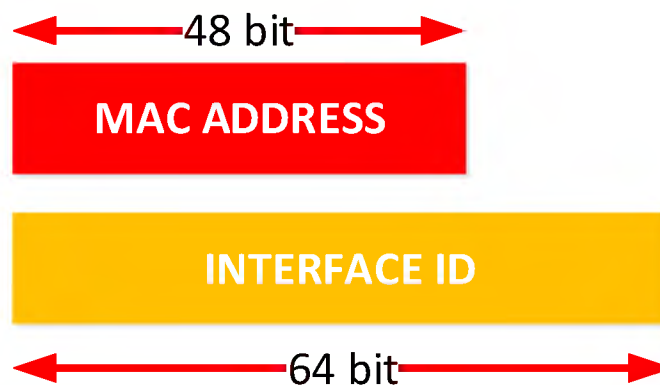
```

Router#show ipv6 route connected
IPv6 Routing Table - 3 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
       O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external
C    2001:1234:5678:ABCD::/64 [0/0]
    via ::, FastEthernet0/0

```

Now you know how to configure an IPv6 address yourself and how to verify it. The second method we can use to configure an address is called **EUI-64 (Extended Unique Identifier)**. This can be used to make the router **generate its own interface ID** instead of typing it in ourselves.

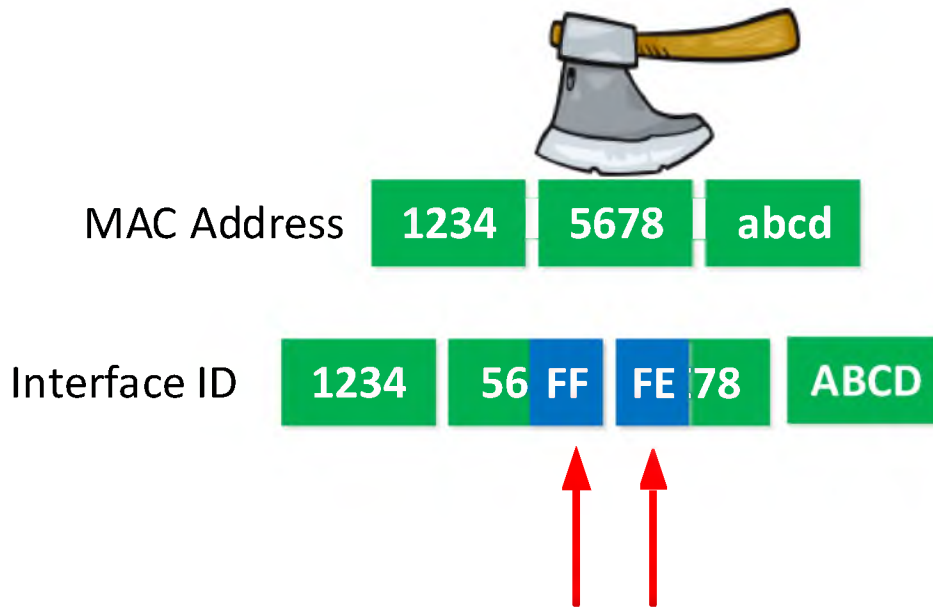
The router will take the MAC address of its interface and use it as the interface ID. However, a MAC address is 48 bit and the interface ID is 64 bit. What are we going to do with the missing bits?



Here's what we will do to fill the missing bits:

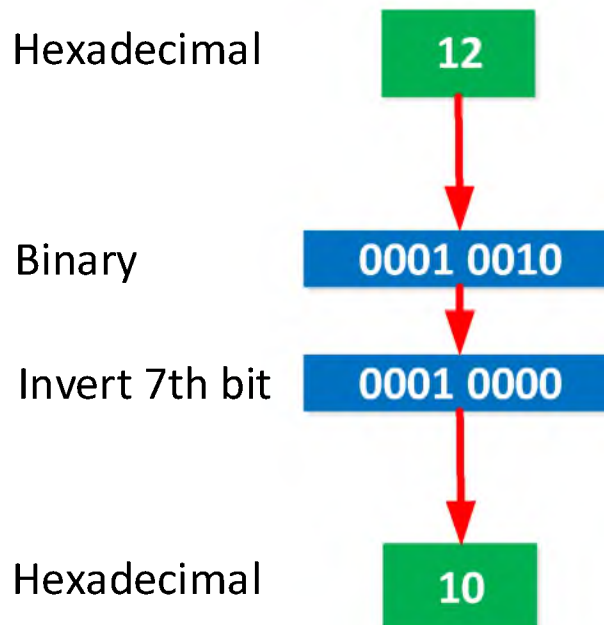
1. We take the MAC address and split it into two pieces.
2. We insert "FFFE" in between the two pieces so that we have a 64 bit value.
3. We invert the 7<sup>th</sup> bit of the interface ID.

So if my MAC address would be 1234.5678.ABCD then this is what the interface ID will become:



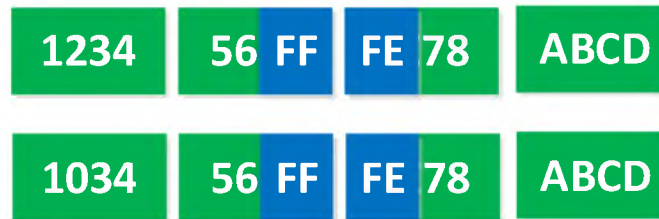
Above you see how we split the MAC address and put FFFE in the middle. It doesn't include the final step which is "inverting the 7<sup>th</sup>" bit. To do this you have to convert the first two hexadecimal characters of the first byte to binary, lookup the 7<sup>th</sup> bit and invert it. This means that if it's a 0 you need to make it a 1, and if it's a 1 it has to become a 0.

The 7<sup>th</sup> bit represents the "universal unique" bit. A burned in MAC address will always have this bit set to 0. When you change the MAC address this bit has to be set to 1. Normally people don't change the MAC address of this router which means that EUI-64 will change the 7<sup>th</sup> bit from 0 to 1 most of the time. Here's what it looks like:





We take the first two hexadecimal characters of the first byte which are "12" and convert those back to binary. Then we invert the 7<sup>th</sup> bit from 1 to 0 and make it hexadecimal again. So in fact my EUI-64 interface ID will look like this:



Now let's take a look at the configuration of EUI-64 on a router! I'll use 2001:1234:5678:abcd::/64 as the prefix:

```
Router(config)#interface fastEthernet 0/0
Router(config-if)#ipv6 address 2001:1234:5678:abcd::/64 eui-64
```

In this I configured the router with the IPv6 prefix and I used **EUI-64** at the end. This is how we can automatically generate the interface ID using the mac address. Now take a look at the IPv6 address that it created:

```
Router#show interfaces fastEthernet 0/0 | include Hardware
Hardware is Gt96k FE, address is c200.185c.0000 (bia c200.185c.0000)
```

```
Router#show ipv6 interface fa0/0
FastEthernet0/0 is up, line protocol is up
IPv6 is enabled, link-local address is FE80::C000:18FF:FE5C:0
No Virtual link-local address(es):
Global unicast address(es):
  2001:1234:5678:ABCD:C000:18FF:FE5C:0, subnet is 2001:1234:5678:ABCD::/64
[EUI]
```

See the C000:18FF:FE5C:0 part? That's the MAC address that is split in 2, FFFE in the middle and the "2" in "C200" of the MAC address has been inverted which is why it now shows up as "C000".

When you use EUI-64 on an interface that doesn't have a MAC address then it will select the MAC address of the lowest numbered interface on the router.



*When you use EUI-64 you should type a 64-bit prefix, not a full IPv6 128-bit address. When you do this, you won't get an error but Cisco IOS will only keep the 64-bit prefix of whatever you typed in and generates the interface ID anyway.*

Normally you probably won't use EUI-64 on a router to configure an interface but this is a very useful technique for normal hosts like windows, linux or mac computers. You will probably configure an IPv6 address manually on your router's interface or use an autoconfiguration technique like DHCP or SLAAC.

When you take a close look at the output of the show ipv6 interface you might noticed that there is one more IPv6 address in there:

```
Router#show ipv6 interface fa0/0
FastEthernet0/0 is up, line protocol is up
IPv6 is enabled, link-local address is FE80::C000:18FF:FE5C:0
```

This address is called a **link-local address** and it has some special purposes for IPv6.

Each device that has IPv6 enabled will **automatically** generate a link-local address. These addresses are unicast, **can't be routed** and are only used within the subnet, which is why they are called "link-local".

A number of protocols use the link-local addresses instead of the global unicast addresses, a good example is NDP (Neighbor Discovery Protocol) which is used to discover the MAC addresses of other IPv6 devices in the subnet (NDP replaces ARP for IPv4).

Routing protocols also use these link-local addresses to establish neighbor adjacencies and also as the next hop for routes. We'll see this when we talk about IPv6 routing.

The FE80::/10 address space has been reserved for link-local addresses which covers FE8, FE9, FEA and FEB. However the RFC that describes link-local addresses states that the next 54 bits have to be zeroes so the link-local addresses will always look like this:



The link-local address will always start with FE80:0000:0000:0000 and the interface ID can be configured using different methods. Cisco routers will use EUI-64 to create a interface ID while other operating systems like Microsoft use a random method to create the interface ID. In the example below you can see that EUI-64 was used to create the link-local address:

```
Router#show interfaces fastEthernet 0/0 | include Hardware
Hardware is Gt96k FE, address is c200.185c.0000 (bia c200.185c.0000)
```

```
Router#show ipv6 interface fa0/0
FastEthernet0/0 is up, line protocol is up
IPv6 is enabled, link-local address is FE80::C000:18FF:FE5C:0
```

The first part is FE80:: and the second part is the EUI-64 created interface ID:

C000:18FF:FE5C:0.

Cisco routers will create a link-local address when you configure an IPv6 address on the interface (global unicast or unique local) or when you enable IPv6 on the interface, you can do it like this:

```
Router(config)#interface fa0/0
Router(config-if)#ipv6 enable
```

Use the **ipv6 enable** command to make the router generate a link-local address.

```
Router#show ipv6 int fa0/0
FastEthernet0/0 is up, line protocol is up
IPv6 is enabled, link-local address is FE80::C000:15FF:FE94:0
```

By default Cisco IOS will use EUI-64 to create the link-local address but you can also configure one yourself. Just make sure the address starts with FE80::/10 (FE8,FE9,FEA or FEB). Here's how you can configure the link-local address:

```
Router(config-if)#ipv6 address FE90:1234:5678:ABCD::1 link-local
```

Just use the **link-local keyword** to tell the router this should be a link-local address. Let's verify it:

```
Router#show ipv6 int fa0/0
FastEthernet0/0 is up, line protocol is up
IPv6 is enabled, link-local address is FE90:1234:5678:ABCD::1
```

I can't think of any good reason why you might want to change the link-local address but it's something you might encounter on the exam.

Besides the link-local addresses there is one more addressing type we have to discuss and that's multicast.

Somewhere in the beginning of this book you learned about unicast and broadcast domains. When a host sends a broadcast, all other devices within the subnet will receive it whether they want it or not. Sending broadcasts is very inefficient and they have been **removed from IPv6**.

Multicast is also used to send something from one host to multiple receivers but the difference is that multicast traffic only ends up at hosts that want to receive it. Everyone that listens to a certain multicast address will receive the packets. It's just like a radiostation, if you want to listen...you have to *tune in* at the right frequency.

IPv6 uses multicast for many reasons. IPv6 hosts that want to send something to all hosts running IPv6 can use the **FF02::1** multicast address. Everyone that has IPv6 enabled listens to this address.

When an IPv6 router wants to send something to all other IPv6 routers (but not hosts!) it can send it to **FF02::2**.

Routing protocols also use multicast addresses. For example, EIGRP uses **FF02::A** and OSPF uses **FF02::5** and **FF02::6**.

Multicast traffic is routable but some traffic should stay within the subnet. When this is the case then these addresses will have a **link-local scope** and they won't be forwarded by routers from one subnet to another.

The **FF00::/8** range has been reserved for IPv6 multicast while the **FF02::/16** range is reserved for link-local scope multicast addresses. On a Cisco router you can see per interface to which multicast addresses the router is listening:

```
Router#show ipv6 int fa0/0
FastEthernet0/0 is up, line protocol is up
IPv6 is enabled, link-local address is FE90:1234:5678:ABCD::1
No Virtual link-local address(es):
No global unicast address is configured
Joined group address(es):
  FF02::1
  FF02::2
  FF02::1:FF00:1
```

This particular router is listening to the “all IPv6 hosts” and “all IPv6 routers” multicast addresses. Once you configure OSPF or EIGRP for IPv6 you will notice that the interface will join the corresponding multicast addresses.

The third address that we have (FF02::1:FF00:1) is called the **solicited-node multicast address**. It is used for *neighbor discovery* which we will discuss in the next chapter.

The solicited-node multicast address is based on the unicast IPv6 address of the host, and to be more precise...the **last six hexadecimal characters of the unicast address**. All hosts that have the same 6 hexadecimal characters in their unicast IPv6 address will end up with the same solicited-node address.

When you send something to this solicited-node address, all hosts with the same address will receive the packets. It’s kinda like the “all IPv6 hosts” multicast address but this time we have a private room where the only members are VIPs that share the same last 6 hexadecimal characters.

All solicited-node addresses start with FF02::1:FF so they look like this:



My router has a solicited node address of FF02::1:FF00:1 while the link-local address is FE90:1234:5678:ABCD::1.

When we write down the link-local address completely it looks like this:

FE90:1234:5678:ABCD:0000:0000:0000:0001

Take the last 6 hexadecimal characters from this address:

00:0001

And put those behind the solicited node address prefix to get the full solicited node address:

**FF02:0000:0000:0000:0001:FF00:0001**

We can remove some of the zeroes to make it shorter and it will look like this:

**FF02::1:FF00:1**

That's how to calculate the solicited-node address.

This is the end of the IPv6 introduction, you have seen some of the differences between IPv4 and IPv6 but also learned about the new addressing that is used. In the next chapter we'll take a look at some more differences between IPv4 and IPv6.

## 22. IPv6 NPD and Host Configuration

In this chapter we are going to look at some of the differences between IPv4 and IPv6 when it comes to hosts and their configuration. IPv6 and IPv4 are very similar in that we still have a unique address per host, we require a default gateway and DNS server but there are also some changes.

Something new to IPv6 is **autoconfiguration** which is almost a “mini-DHCP” server and some protocols have been removed or changed. Just like IPv4, hosts configured for IPv6 need to learn the MAC address of other devices but we don’t use ARP anymore, it has been replaced by a protocol called **NDP (Neighbor Discovery Protocol)**.

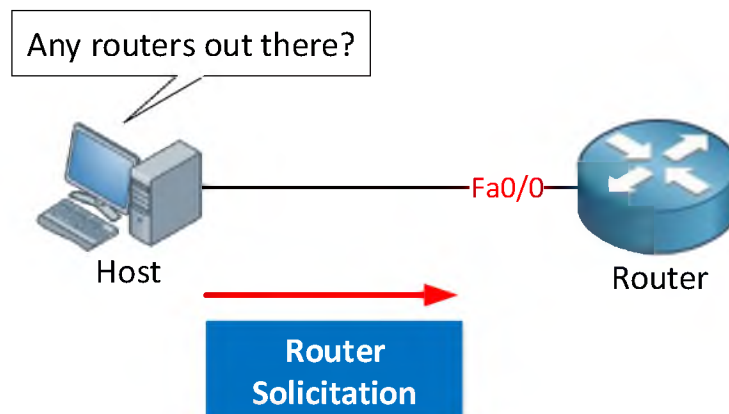
Besides learning MAC addresses, NDP is used for a number of tasks:

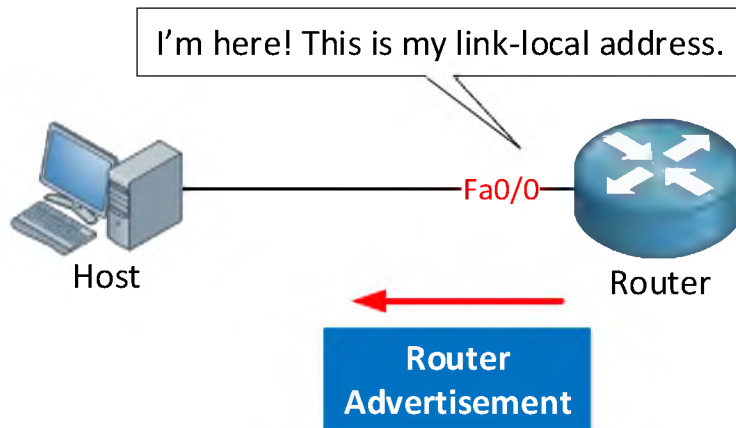
- **Router Discovery:** NDP is used to learn about all available IPv6 routers in the subnet.
- **MAC Address Discovery:** Once a host has done the DAD check and is using an IPv6 address it will have to discover the MAC addresses of hosts it wants to communicate with.
- **DAD (Duplicate Address Detection):** Each IPv6 host will wait to use its address unless it knows that no other device is using the same address. This process is called DAD and NDP does this for us.
- **SLAAC (Stateless Address Autoconfiguration):** We’ll cover SLAAC in a bit in more detail, but NDP is used to learn what address and prefix length the host should use.

We’ll walk through the different tasks to see how they work. We’ll start with the router discovery. When a host is configured for IPv6 it will automatically discover the routers on the subnet.

An IPv6 host can use NDP to detect all routers on the subnet that could be used as a default gateway. Basically the host will send a message asking if there are any routers out there, and the routers will respond. The two messages used are:

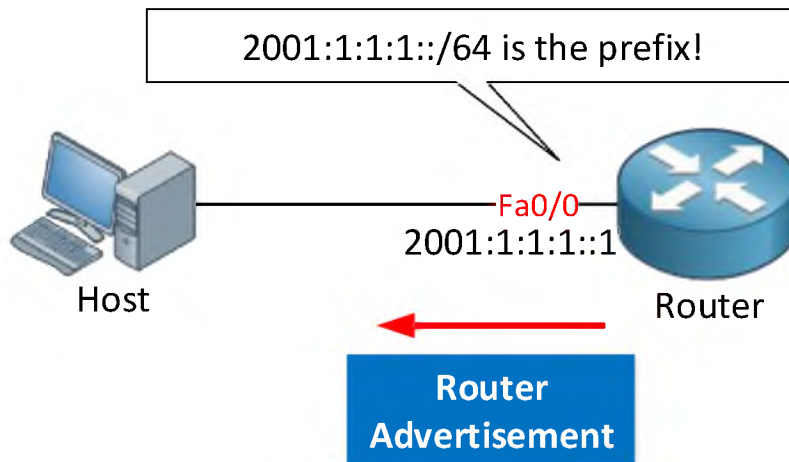
- **RS (Router Solicitation)** which is sent to the “all ipv6 routers” FF02::2 multicast address.
- **RA (Router Advertisement)** is sent by the router and includes its link-local IPv6 address.





When a host sends the router solicitation a router will respond to the unicast address of the host. Routers will also periodically send router advertisements for everyone that's interested, they will use the FF02::1 "all nodes" address for this.

Most routers will also have a global unicast address configured on the interface, when this is the case the hosts will not only learn about the link-local address but also the **prefix** that is used on the subnet.

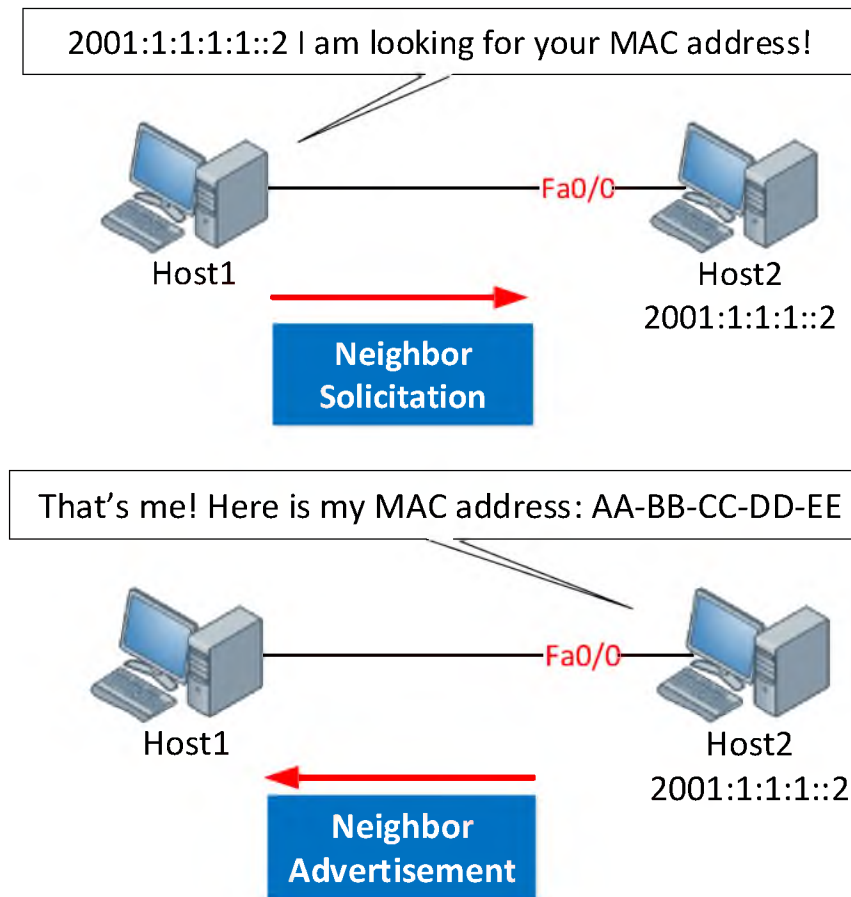


This prefix can be used for SLAAC (Stateless Address Autoconfiguration) that we'll look at later.

NPD is also used as a replacement for ARP. It uses two kind of messages for this:

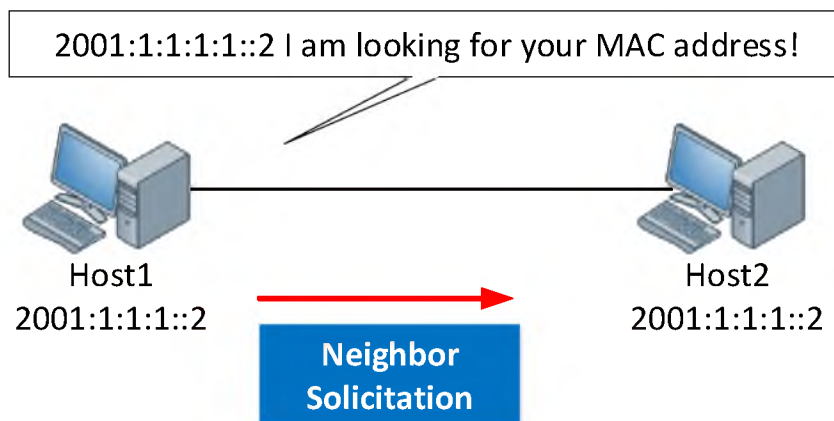
- **NS (Neighbor Solicitation)**
- **NA (Neighbor Advertisement)**

The neighbor solicitation works similar to the ARP request, it will ask a certain host for its MAC address and the neighbor advertisement is like the ARP reply as it is used to send the MAC address. Basically it looks like this:

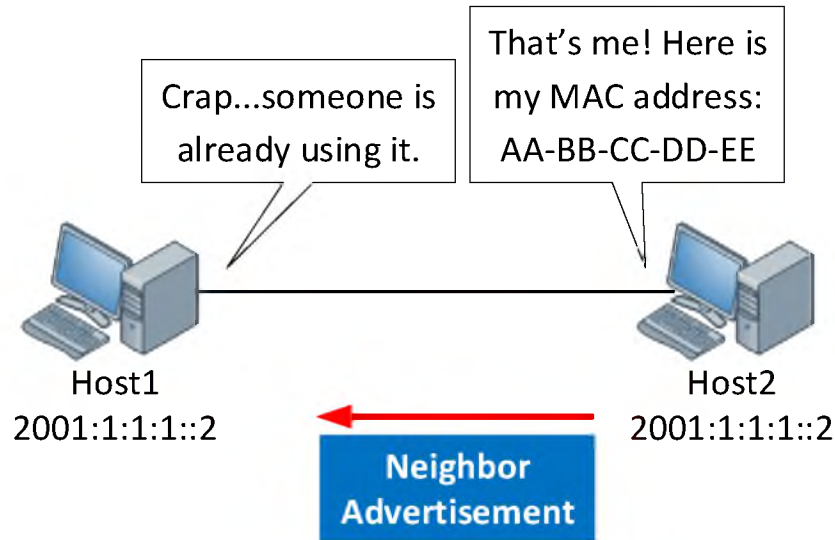


Whenever a host sends a neighbor solicitation it will first check its cache to see if it already knows the MAC address of the device it is looking for. If it's not there, it will send the neighbor solicitation. These neighbor solicitation messages use the **solicited-node multicast address**.

Besides discovering MAC addresses, the NS and NA messages are also used to **detect duplicate IPv6 addresses**. Before an IPv6 device uses a unicast address it will perform **DAD (Duplicate Address Detection)** to see if someone else is already using the same IPv6 address. If the address is in use, the host will not use it. Here's what it looks like:







Host1 has configured 2001:1:1:1::2 as its IPv6 address which is already in use by host2. It will send a neighbor solicitation but because host2 has the same IPv6 address it will reply with a neighbor advertisement. Host1 now knows that this is a duplicate IPv6 address. This check is performed for all of the unicast addresses including the link-local addresses. It occurs when you configure them and each time the interface is 'up'.

The last NPD that we will look at is SLAAC (Stateless Address Autoconfiguration), which lets hosts automatically configure their IPv6 address. For IPv4 we have always used DHCP to automatically assign an IP address, default gateway and DNS server to our hosts and this option is still available for IPv6 (we'll look at it in a minute).

DHCP works really well but the downside is that you need to install a DHCP server, configure the pool with address ranges, default gateways and DNS servers.

When we use SLAAC our hosts will not receive an IPv6 address from a central server but it will **learn the prefix** that is used on the subnet and then creates its own interface ID to generate a unique IPv6 address. This is how SLAAC works:

1. The host first learns about the prefix by using NDS RS & RA messages.
2. The host takes the prefix and creates an interface ID to make a unique IPv6 address.
3. The host performs a DAD to make sure the IPv6 address is not used by anyone else.

Cisco routers will use EUI-64 to create the interface ID but some operating systems will use a random value. Thanks to SLAAC the host will have an IPv6 address and a default router but one ingredient is still missing...**the DNS server**. SLAAC can't help us with finding a DNS server so for this step we still require DHCP.

DHCP for IPv6 is called DHCPv6 and comes in two forms:

- Stateful
- Stateless

We'll cover DHCPv6 in a minute but for SLAAC we need to understand what **stateless DHCPv6** is about. Normally a DHCP server will keep track of the IP addresses that is has

leased to clients, in other words it has to keep the “state” of what IP addresses have been leased and when they expire.

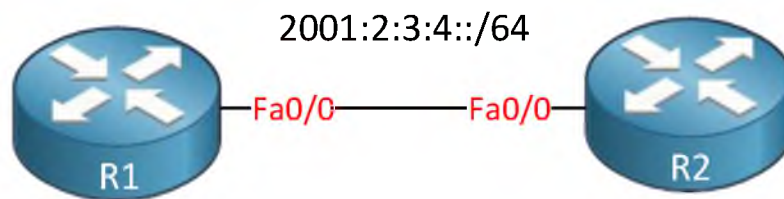
A stateless DHCPv6 server **doesn't keep track of anything** for clients. It has a simple configuration with the IPv6 addresses of a few DNS servers. When an IPv6 host asks the DHCPv6 server for the IPv6 address of the DNS server it will give this address and that's it.

So when you use SLAAC, you still need stateless DHCPv6 to learn about the DNS servers.

You have now seen all the tasks that NPD does for us:

- Router Discovery
- MAC Address Discovery
- Duplicate Address Detection
- Stateless Address Autoconfiguration

Let's take a look at NPD on some real routers so you can see how it works in action. I'll use the following topology to demonstrate this:



I will use R1 as a host that will autoconfigure itself using SLAAC and R2 as the router. 2001:2:3:4::/64 is the prefix that we will use. Let's configure R2 first:

```
R2(config)#ipv6 unicast-routing
```

Before R2 will act as a router we need to make sure IPv6 unicast routing is enabled. Now let's configure an IPv6 address on the interface:

```
R2(config)#interface fa0/0
R2(config-if)#no shutdown
R2(config-if)#ipv6 address 2001:2:3:4::1/64
```

Before we configure R1 we will enable NPD debugging on both routers so you can see the different messages:

```
R1#debug ipv6 nd
ICMP Neighbor Discovery events debugging is on
```

```
R2#debug ipv6 nd
ICMP Neighbor Discovery events debugging is on
```

The **debug ipv6 nd** command is very useful as it will show all the different messages that NPD uses.

Let's configure R1 now:

```
R1(config)#interface fa0/0
R1(config-if)#no shutdown
R1(config-if)#ipv6 address autoconfig
```

R1 will be configured to use SLAAC with the **ipv6 address autoconfig** command. With the debug enabled you will see the following items on your console:

```
R1#
ICMPv6-ND: Sending NS for FE80::C000:6FF:FE7C:0 on FastEthernet0/0
ICMPv6-ND: DAD: FE80::C000:6FF:FE7C:0 is unique.
```

It sends a NS for its own IPv6 address and when nobody responds, it understands that it is the only host using this address.

You can also see that R1 sends a neighbor advertisement towards R2:

```
R1#
ICMPv6-ND: Sending NA for FE80::C000:6FF:FE7C:0 on FastEthernet0/0
```

```
R2#
ICMPv6-ND: Received NA for FE80::C000:6FF:FE7C:0 on FastEthernet0/0 from
FE80::C000:6FF:FE7C:0
```

We can view the database with the L2 and L3 information like this:

```
R2#show ipv6 neighbors
IPv6 Address                               Age Link-layer Addr State Interface
FE80::C000:6FF:FE7C:0                    21 c200.067c.0000 STALE Fa0/0
```

**Show ipv6 neighbors** will show you the IPv6 addresses and MAC addresses.

R1 will also send a router solicitation and R2 will send a router advertisement in return:

```
R1#
ICMPv6-ND: Sending RS on FastEthernet0/0
```

```
R2#
ICMPv6-ND: Received RS on FastEthernet0/0 from FE80::C000:6FF:FE7C:0
ICMPv6-ND: Sending solicited RA on FastEthernet0/0
ICMPv6-ND: Sending RA from FE80::C001:6FF:FE7C:0 to FF02::1 on
FastEthernet0/0
ICMPv6-ND:      MTU = 1500
ICMPv6-ND:      prefix = 2001:2:3:4::/64 onlink autoconfig
ICMPv6-ND:      2592000/604800 (valid/preferred)
```

```
R1#
ICMPv6-ND: Received RA from FE80::C001:6FF:FE7C:0 on FastEthernet0/0
ICMPv6-ND: Selected new default router FE80::C001:6FF:FE7C:0 on
FastEthernet0/0
```

If you want to see all the routers that your host knows about you can use the following command:

```
R1#show ipv6 routers
Router FE80::C001:6FF:FE7C:0 on FastEthernet0/0, last update 0 min
  Hops 64, Lifetime 1800 sec, AddrFlag=0, OtherFlag=0, MTU=1500
  HomeAgentFlag=0, Preference=Medium
  Reachable time 0 msec, Retransmit time 0 msec
  Prefix 2001:2:3:4::/64 onlink autoconfig
  Valid lifetime 2592000, preferred lifetime 604800
```

Since R1 is configured for SLAAC it will use the prefix in the router advertisement to configure itself:

```
R1#
ICMPv6-ND: Prefix Information change for 2001:2:3:4::/64, 0x0 -> 0xE0
ICMPv6-ND: Adding prefix 2001:2:3:4::/64 to FastEthernet0/0
ICMPv6-ND: Sending NS for 2001:2:3:4:C000:6FF:FE7C:0 on FastEthernet0/0
ICMPv6-ND: Autoconfiguring 2001:2:3:4:C000:6FF:FE7C:0 on FastEthernet0/0
ICMPv6-ND: DAD: 2001:2:3:4:C000:6FF:FE7C:0 is unique.
```

It will use the prefix and configure an IPv6 address automatically. Before it uses the address it will use DAD to make sure the address is unique. Let's take a look what the IPv6 address is:

```
R1#show ipv6 int brief
FastEthernet0/0 [up/up]
  FE80::C000:6FF:FE7C:0
  2001:2:3:4:C000:6FF:FE7C:0
```

You can see that R1 used the 2001:2:3:4::/64 prefix to configure itself. If you want to configure this yourself and look at the debug, you might want to try this lab that I created for you:

<http://gns3vault.com/IPv6/ipv6-autoconfiguration.html>

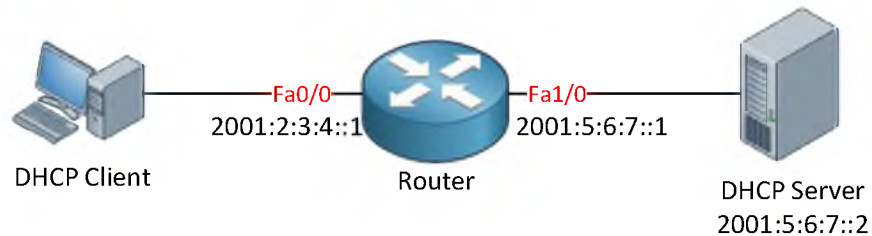
That's all I have about NPD for you now, let's continue by taking a closer look at DHCPv6! **Stateful DHCPv6** works similar to DHCP for IPv4. We still use it to give addresses, default gateways, DNS servers and some other options to clients but one of the key differences are the messages that we now use.

DHCP for IPv4 uses the Discover, Offer, Request and ACK messages. DHCPv6 uses a **Solicit, Advertise, Request and Reply message**.

The solicit message is similar to the discover message. A host will use this message when it's looking for the IPv6 address of the DHCPv6 server. The advertise message is used to give an IPv6 address, default gateway and DNS server to the host. The request message is used by the host to ask if it's ok to use this information and the ACK is sent by the server to confirm this.

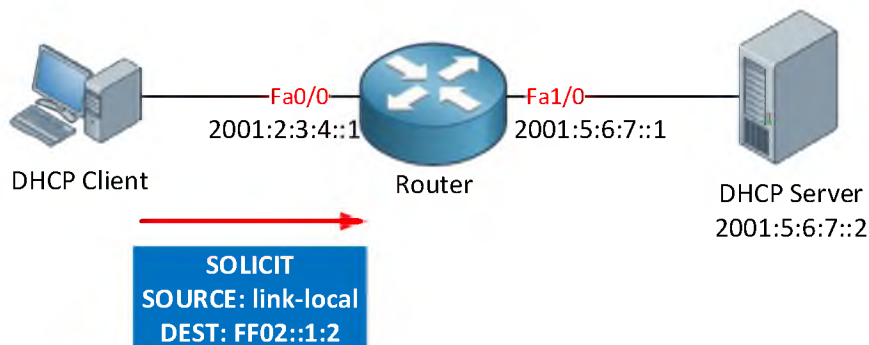
Just like DHCP for IPv4, when your DHCP server is not on the same subnet you will require **DHCP relay** to forward the DHCP messages to a central DHCP server.

You don't have to configure a DHCPv6 server for CCNA but you do have to understand how the different addresses are used. Let's look at an example:



In this example we have a DHCP client, a router and a DHCP server. Since the DHCP client and DHCP server are not on the same subnet we will have to configure the router to relay the DHCP messages.

When the client is looking for an IPv6 address it will start with the solicit message:

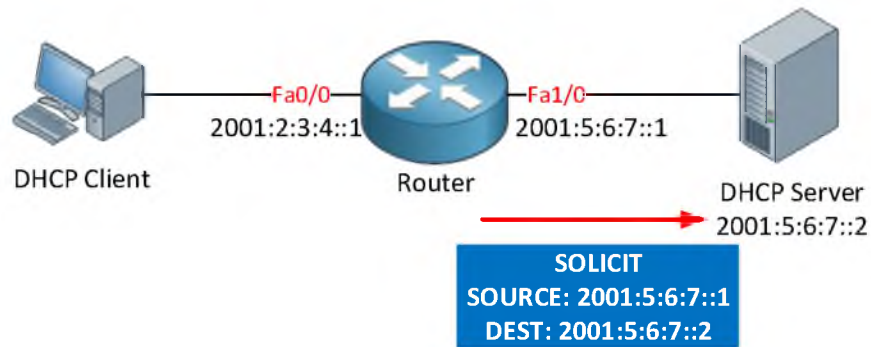


The client will use its link-local address as the source and the destination will be multicast address **FF02::1:2 (all-DHCP-agents)**. This is a link-local multicast address so it won't leave the subnet. As a result the DHCP server will never receive this solicit message.

On the router we will configure DHCP relay so that the solicit message will be forwarded to the DHCP server:

```
Router(config)#interface fa0/0
Router(config-if)#ipv6 dhcp relay destination 2001:5:6:7::2
```

This will ensure that the router forwards DHCP messages between the client and DHCP server. This is what it looks like:



The router will forward the solicit message and the addresses will change. The source will be the IPv6 address on the Fa1/0 interface of our router and the destination will be the IPv6 address of the DHCP server. This is a big difference compared to DHCP relay for IPv4 where the IP address on the Fa0/0 would have been used.

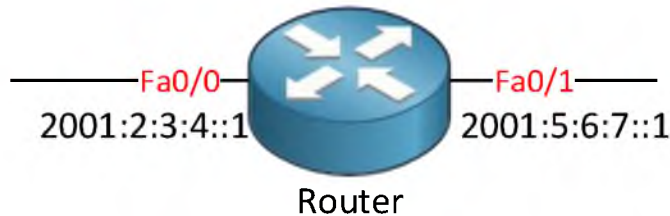
The other DHCP messages will use the same addresses. Between the router and DHCP server we will use the 2001:5:6:7::1 and 2001:5:6:7::2 addresses. The router will forward traffic to the DHCP client by using its link-local address as the destination.

That's all we have for this chapter, the most important part is understanding NPD and how it replaced some of the things from the IPv4 world. In the next chapter we'll take a look at IPv6 routing!

## 23. IPv6 Routing

In the previous chapters you have learned what IPv6 addresses look like, the differences with IPv4 and so on but we haven't talked about routing yet. In this chapter we'll take a look how we can use static routes, OSPF and EIGRP to fill our routing tables.

Just like IPv4, a router that is configured for IPv6 will build a routing table and adds the prefixes to it. Let's see how it works:



This router has two interfaces with a global unicast address. We'll configure this router and look at the routing table:

```

Router(config)#ipv6 unicast-routing
Router(config)#interface fa0/0
Router(config-if)#ipv6 address 2001:2:3:4::1/64

Router(config)#interface fa0/1
Router(config-if)#ipv6 address 2001:5:6:7::1/64
  
```

Let's check the routing table:

```

R2#show ipv6 route
IPv6 Routing Table - 5 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
       O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external
C   2001:2:3:4::/64 [0/0]
    via ::, FastEthernet0/0
L   2001:2:3:4::1/128 [0/0]
    via ::, FastEthernet0/0
C   2001:5:6:7::/64 [0/0]
    via ::, FastEthernet0/1
L   2001:5:6:7::1/128 [0/0]
    via ::, FastEthernet0/1
L   FF00::/8 [0/0]
    via ::, Null0
  
```

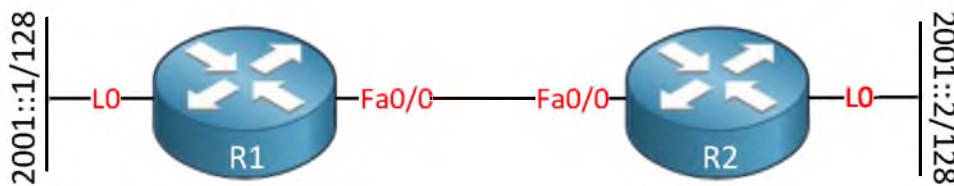
The routing table looks similar to an IPv4 routing table but there are some differences. First of all we can see the directly connected networks, here we find the 2001:2:3:4::/64 and 2001:5:6:7::/64 prefixes that we configured.

You can also see some **local** entries. These are so called **host routes**. The router will create an entry for each IPv6 address that has been configured. You can see that they have a /128 address mask. These local entries are used for a feature called MTR (Multi Topology Routing). You can forget about this as it's not part of the R&S track.

Also note that you will **never find any link-local addresses** in the routing table. They are **not routable** so there's no point adding them to the routing table.

Let's see if we can get some prefixes in our routing table, to do this we'll start with some static routing.

Static routes are similar to the ones you have seen for IPv4 but there are a few differences. Let's look at an example:



We have two routers, R1 and R2 and each router has a loopback interface. We'll use static routes to make sure they can reach each other. First we'll enable IPv6 routing and configure the loopback interfaces:

```

R1(config)#ipv6 unicast-routing
R1(config)#interface fa0/0
R1(config-if)#ipv6 enable
R1(config-if)#exit
R1(config)#interface l0
R1(config-if)#ipv6 address 2001::1/128
  
```

```

R2(config)#ipv6 unicast-routing
R2(config)#interface fa0/0
R2(config-if)#ipv6 enable
R2(config-if)#exit
R2(config)#interface l0
R2(config-if)#ipv6 address 2001::2/128
  
```

I have used `ipv6 enable` on the interfaces so that the routers will create a link-local IPv6 address on the fastethernet interfaces. We can use these as the next hop address. Let's lookup those link-local addresses:

```

R1#show ipv6 int brief
FastEthernet0/0          [up/up]
FE80::C000:16FF:FE1C:0
  
```

```

R2#show ipv6 int brief
FastEthernet0/0          [up/up]
FE80::C001:16FF:FE1C:0
  
```



Now we can use these addresses as the next hop for our static routes. You can use the **ipv6 route** command for this:

```
R1(config)#ipv6 route 2001::2/128 fa0/0 FE80::C001:16FF:FE1C:0
```

```
R2(config)#ipv6 route 2001::1/128 fa0/0 FE80::C000:16FF:FE1C:0
```

When you use the link-local address as the next hop you also have to specify the interface that you will use to reach the link-local address. Let's look at the routing table:

```
R1#show ipv6 route static
IPv6 Routing Table - 3 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
       O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external
S    2001::2/128 [1/0]
    via FE80::C001:16FF:FE1C:0, FastEthernet0/0
```

```
R2#show ipv6 route static
IPv6 Routing Table - 3 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
       O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external
S    2001::1/128 [1/0]
    via FE80::C000:16FF:FE1C:0, FastEthernet0/0
```

**Show ipv6 route** let's you look at the IPv6 routing table. Here you can see the static route with the link-local address as next hop. So can we reach each others prefix? Let's try it:

```
R1#ping 2001::2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001::2, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/27/32 ms
```

```
R2#ping 2001::1

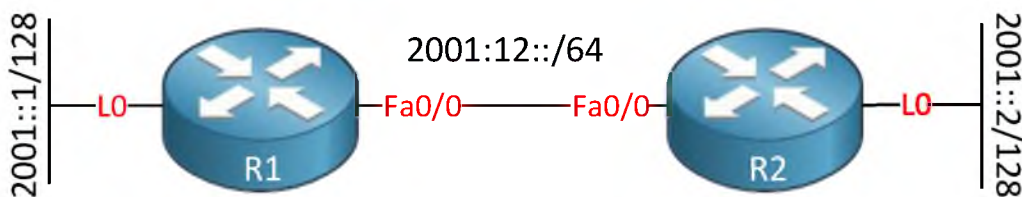
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001::1, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/30/36 ms
```

No problem at all. Just use the **ping** command for a quick test.

You can also use traceroute for IPv6:

```
R1#traceroute 2001::2
Type escape sequence to abort.
Tracing the route to 2001::2
 1 2001::2 32 msec 32 msec 32 msec
```

This is looking good. Instead of using link-local addresses let's try the global unicast addresses. To do this we'll have to add a prefix between R1 and R2:



We will use the 2001:12::/64 prefix, let's configure some IPv6 addresses on R1 and R2:

```
R1(config)#interface fa0/0
R1(config-if)#ipv6 address 2001:12::1/64
```

```
R2(config)#interface fa0/0
R2(config-if)#ipv6 address 2001:12::2/64
```

Now let's get rid of the old static routes:

```
R1(config)#no ipv6 route 2001::2/128 FastEthernet0/0
```

```
R2(config)#no ipv6 route 2001::1/128 FastEthernet0/0
```

Now I'll add new static routes that use the 2001:12::1 and 2001:12::2 addresses as the next hop:

```
R1(config)#ipv6 route 2001::2/128 2001:12::2
```

```
R2(config)#ipv6 route 2001::1/128 2001:12::1
```

Let's look at the routing tables:

```
R1#show ipv6 route static
IPv6 Routing Table - 5 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
        U - Per-user Static route, M - MIPv6
        I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
        O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
        ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
        D - EIGRP, EX - EIGRP external
S    2001::2/128 [1/0]
    via 2001:12::2
```

```
R2#show ipv6 route static
IPv6 Routing Table - 5 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
        U - Per-user Static route, M - MIPv6
        I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
        O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
        ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
        D - EIGRP, EX - EIGRP external
S    2001::1/128 [1/0]
    via 2001:12::1
```

As you can see the 2001:12::2 and 2001:12::1 addresses are now used as the next hop.  
Let's do a quick ping test:

```
R1#ping 2001::2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001::2, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/28/32 ms
```

```
R2#ping 2001::1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001::1, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/28/32 ms
```

Everything is working as it should.

Just like IPv6 we can also use static routes to create a default route. Let's remove the static route on R1:

```
R1(config)#ipv6 route 2001::2/128 2001:12::2
```

And replace it with a default route:

```
R1(config)#ipv6 route ::/0 2001:12::2
```

The :: is a string of zeroes, and the /0 means a prefix length of 0. This is the same as IPv4 0.0.0.0/0. Here's what the routing table looks like:

```
R1#show ipv6 route static
IPv6 Routing Table - 5 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
        U - Per-user Static route, M - MIPv6
        I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
        O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
        ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
        D - EIGRP, EX - EIGRP external
S    ::/0 [1/0]
    via 2001:12::2
```

That's looking good, let's try a quick ping:

```
R1#ping 2001::2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001::2, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/26/28 ms
```

This proves my default route is working.

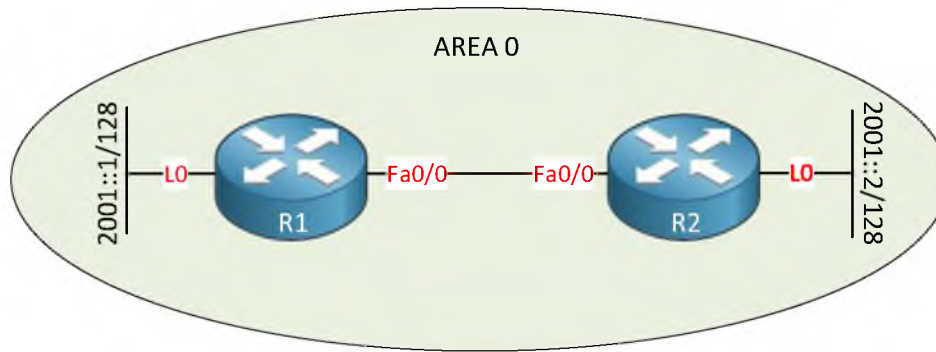
Static routes are fun and useful if you have a couple of prefixes but when networks grow larger you might want to use a dynamic routing protocol. We are going to cover two of them:

- **OSPFv3**
- **EIGRP**

The version of OSPF that you have seen for IPv4 is officially called OSPFv2 and to support IPv6 a new version was created called OSPFv3. This new version of OSPF **only supports IPv6**, not IPv4. This means that if you plan to run IPv4 and IPv6 at the same time, you'll have to configure both OSPFv2 and OSPFv3.

OSPFv3 hasn't changed much, it's still a link-state routing protocol that builds a LSDB using LSAs. When it comes to CCNA, the biggest difference is the configuration...which (luckily) has become easier because there are no network commands anymore.

Let's start with a simple example so I can show you how to configure OSPFv3. I'll use the same topology I used for the static routes:



First we'll prepare the interfaces, enable IPv6 unicast routing, etc:

```

R1(config)#ipv6 unicast-routing
R1(config)#interface fa0/0
R1(config-if)#ipv6 enable
R1(config-if)#exit
R1(config)#interface l0
R1(config-if)#ipv6 address 2001::1/128

```

```

R2(config)#ipv6 unicast-routing
R2(config)#interface fa0/0
R2(config-if)#ipv6 enable
R2(config-if)#exit
R2(config)#interface l0
R2(config-if)#ipv6 address 2001::2/128

```

Note that I don't configure any IPv6 addresses on the FastEthernet interfaces. As you will see in a bit OSPFv3 will use link-local addresses to establish the neighbor adjacency and to exchange LSAs. Let's enable OSPFv3:

```

R1(config)#ipv6 router ospf 1
R1(config-rtr)#router-id 1.1.1.1

```

```

R2(config)#ipv6 router ospf 1
R2(config-rtr)#router-id 2.2.2.2

```

First we have to enable OSPFv3 with the **ipv6 router ospf** command, the "1" is the process ID. The funny thing is that we have to **configure a router-ID ourselves** and it's in "IPv4" format. I guess someone felt nostalgic here and decided to use an IPv4 address for this router ID. If you don't configure a router ID, **OSPFv3 will not work**.

The second step is really easy, instead of using those pesky network commands we just have to tell the interface to enable OSPFv3:

```

R1(config)#interface fa0/0
R1(config-if)#ipv6 ospf 1 area 0

```

```

R2(config)#interface fa0/0
R2(config-if)#ipv6 ospf 1 area 0

```

This will enable OSPFv3 on the FastEthernet interfaces, and we need this to establish the OSPFv3 neighbor adjacency:

```
R1#show ipv6 ospf neighbor
```

| Neighbor ID | Pri | State   | Dead Time | Interface ID | Interface       |
|-------------|-----|---------|-----------|--------------|-----------------|
| 2.2.2.2     | 1   | FULL/DR | 00:00:31  | 4            | FastEthernet0/0 |

This command is the same as for IPv4, even the output looks the same. Our two routers are now neighbors but before they can learn about each others prefix on the loopback interface we need to enable OSPFv3 on the loopback interfaces:

```
R1(config)#interface loopback 0
R1(config-if)#ipv6 ospf 1 area 0
```

```
R2(config)#interface loopback 0
R2(config-if)#ipv6 ospf 1 area 0
```

Activating OSPFv3 on the loopback interfaces will put the prefixes in the LSDB so they can be advertised.



*OSPFv2 for IPv4 also supports these interface commands but unfortunately for CCNA you still have to use and understand the "network" command.*

Let's check the routing tables:

```
R1#show ipv6 route ospf
```

IPv6 Routing Table - 3 entries

Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP

U - Per-user Static route, M - MIPv6

I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary

O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2

ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2

D - EIGRP, EX - EIGRP external

```
O 2001::2/128 [110/10]
   via FE80::C001:16FF:FE1C:0, FastEthernet0/0
```

```
R2#show ipv6 route ospf
```

IPv6 Routing Table - 3 entries

Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP

U - Per-user Static route, M - MIPv6

I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary

O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2

ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2

D - EIGRP, EX - EIGRP external

```
O 2001::1/128 [110/10]
   via FE80::C000:16FF:FE1C:0, FastEthernet0/0
```

Here we can see that the prefixes on the loopback interfaces have been learned by OSPFv3. If you look closely you can see that the link-local addresses are used as the next hop address.

Besides the routing table we have some more commands we can use to check if OSPFv3 is working correctly:

```
R1#show ipv6 ospf
Routing Process "ospfv3 1" with ID 1.1.1.1
SPF schedule delay 5 secs, Hold time between two SPFs 10 secs
Minimum LSA interval 5 secs. Minimum LSA arrival 1 secs
LSA group pacing timer 240 secs
Interface flood pacing timer 33 msecs
Retransmission pacing timer 66 msecs
Number of external LSA 0. Checksum Sum 0x000000
Number of areas in this router is 1. 1 normal 0 stub 0 nssa
Reference bandwidth unit is 100 mbps
Area BACKBONE (0)
Number of interfaces in this area is 2
SPF algorithm executed 2 times
Number of LSA 7. Checksum Sum 0x0278FD
Number of DCbitless LSA 0
Number of indication LSA 0
Number of DoNotAge LSA 0
Flood list length 0
```

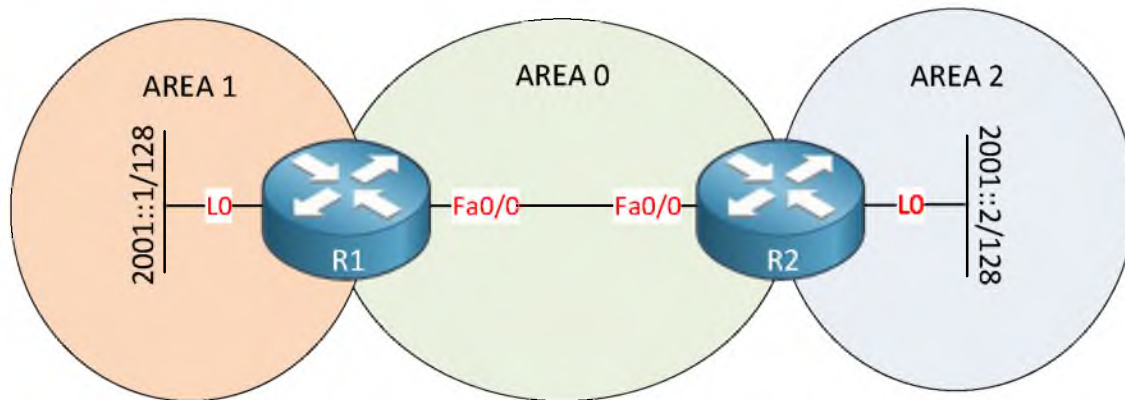
The **show ipv6 ospf** command gives us some useful information about the router ID and the area.

```
R1#show ipv6 ospf interface brief
Interface    PID    Area      Intf ID    Cost    State Nbrs F/C
Lo0          1      0         10         1      LOOP  0/0
Fa0/0        1      0         4          10     BDR   1/1
```

This command shows us OSPFv3 information per interface. And last but not least, show ipv6 protocols is great to get an overview of all routing protocols that are running:

```
R1#show ipv6 protocols
IPv6 Routing Protocol is "connected"
IPv6 Routing Protocol is "ospf 1"
  Interfaces (Area 0):
    Loopback0
    FastEthernet0/0
  Redistribution:
    None
```

If you want multi-area OSPFv3 you just have to specify the correct area when enabling OSPF on the interface. Let me give you an example:



I will use the same two routers but each loopback interface will be in a different area:

```
R1(config)#interface loopback 0
R1(config-if)#no ipv6 ospf 1 area 0
R1(config-if)#ipv6 ospf 1 area 1
```

```
R2(config)#interface loopback 0
R2(config-if)#no ipv6 ospf 1 area 0
R2(config-if)#ipv6 ospf 1 area 1
```

Now let's look at the routing tables:

```
R1#show ipv6 route ospf
IPv6 Routing Table - 3 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
       O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external
OI  2001::2/128 [110/10]
    via FE80::C001:16FF:FE1C:0, FastEthernet0/0
```

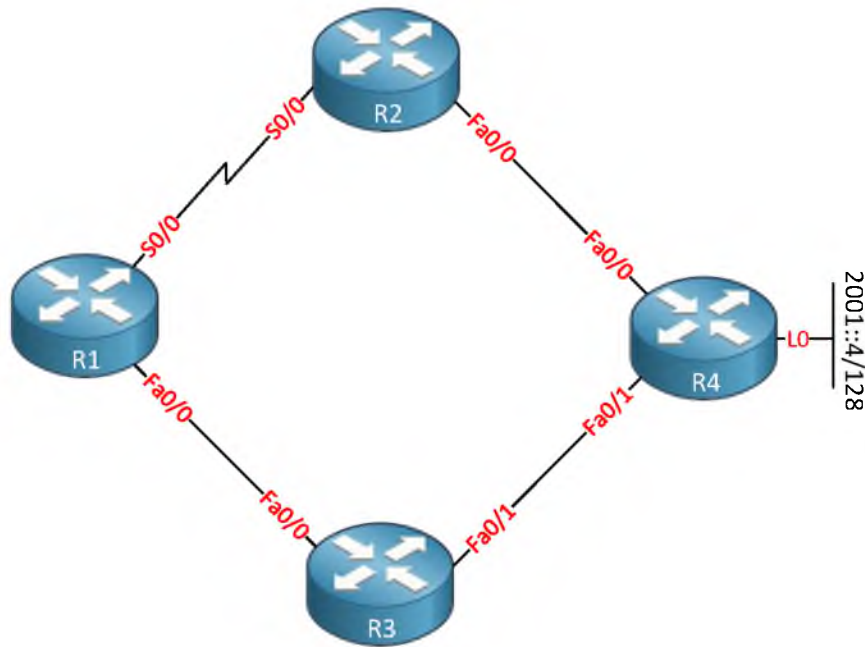
```
R2#show ipv6 route ospf
IPv6 Routing Table - 3 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
       O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external
OI  2001::1/128 [110/10]
    via FE80::C000:16FF:FE1C:0, FastEthernet0/0
```

R1 now sees 2001::2/128 as an inter-area prefix and so does R2 see 2001::1/128. That's all I have about OSPFv3 for you. It might take some time to get used to the IPv6 addressing but the configuration itself is easier than IPv4.



Let's take a look at EIGRP for IPv6! The good thing about EIGRP is that it's 99% the same as for IPv4. All the commands are the same, just replace IPv6 with IPv4 and that's it.

Configuration-wise there are some minor differences. To demonstrate EIGRP I'll use the following topology:



Take a close look at this topology. All interfaces are FastEthernet but the link between R1 and R2 uses slow serial interfaces. Behind R4 we have a loopback with prefix 2001::4/128.

Let's prepare the interfaces for this topology:

```

R1(config)#ipv6 unicast-routing
R1(config)#interface fa0/0
R1(config-if)#ipv6 enable

R1(config)#interface s0/0
R1(config-if)#ipv6 enable
  
```

```

R2(config)#ipv6 unicast-routing
R2(config)#interface fa0/0
R2(config-if)#ipv6 enable

R2(config)#interface s0/0
R2(config-if)#ipv6 enable
  
```

```

R3(config)#ipv6 unicast-routing
R3(config)#interface fa0/0
R3(config-if)#ipv6 enable

R3(config)#interface fa0/1
R3(config-if)#ipv6 enable
  
```

```
R4(config)#ipv6 unicast-routing
R4(config)#interface fa0/0
R4(config-if)#ipv6 enable

R4(config)#interface fa0/1
R4(config-if)#ipv6 enable

R4(config)#interface l0
R4(config-if)#ipv6 address 2001::4/128
```

I enabled IPv6 routing, enabled IPv6 on the interfaces so a link-local address is created and R4 has a loopback with the 2001::4/128 IPv6 address.

Now we can enable EIGRP:

```
R1(config)#ipv6 router eigrp 1
R1(config-rtr)#router-id 1.1.1.1
R1(config-rtr)#no shutdown
```

```
R2(config)#ipv6 router eigrp 1
R2(config-rtr)#router-id 2.2.2.2
R2(config-rtr)#no shutdown
```

```
R3(config)#ipv6 router eigrp 1
R3(config-rtr)#router-id 3.3.3.3
R3(config-rtr)#no shutdown
```

```
R4(config)#ipv6 router eigrp 1
R4(config-rtr)#router-id 4.4.4.4
R4(config-rtr)#no shutdown
```

First we need to specify the AS number and secondly we need to **enable a router ID**. This is the same as for OSPFv3. Keep in mind that the AS number has to match or EIGRP routers won't become neighbors! Something new is that EIGRP for IPv6 can be in **shutdown mode**. Just like an interface you can do a shut or no shut.

Now we can enable EIGRP on the interfaces:

```
R1(config)#interface s0/0
R1(config-if)#ipv6 eigrp 1

R1(config)#interface fa0/0
R1(config-if)#ipv6 eigrp 1
```

```
R2(config)#interface s0/0
R2(config-if)#ipv6 eigrp 1

R2(config)#interface fa0/0
R2(config-if)#ipv6 eigrp 1
```

```
R3(config)#interface fa0/0
R3(config-if)#ipv6 eigrp 1

R3(config)#interface fa0/1
R3(config-if)#ipv6 eigrp 1
```

```
R4(config)#interface fa0/0
R4(config-if)#ipv6 eigrp 1

R4(config)#interface fa0/1
R4(config-if)#ipv6 eigrp 1

R4(config)#interface l0
R4(config-if)#ipv6 eigrp 1
```

Luckily there are no network commands anymore. We can enable EIGRP on the interface level with the **ipv6 eigrp** command. Let's check if we have neighbors:

```
R2#show ipv6 eigrp neighbors
IPv6-EIGRP neighbors for process 1
H   Address                               Interface      Hold Uptime    SRTT   RTO  Q  Seq
                               (sec)          (ms)          Cnt  Num
1   Link-local address: Fa0/0              13 00:02:03    1   3000  0   1
    FE80::C013:DFF:FE5C:0
0   Link-local address: Se0/0              12 00:03:16    8    200  0   1
    FE80::C010:BFF:FE30:0
```

```
R3#show ipv6 eigrp neighbors
IPv6-EIGRP neighbors for process 1
H   Address                               Interface      Hold Uptime    SRTT   RTO  Q  Seq
                               (sec)          (ms)          Cnt  Num
1   Link-local address: Fa0/1              13 00:02:06    1   3000  0   2
    FE80::C013:DFF:FE5C:1
0   Link-local address: Fa0/0              12 00:03:16    1   3000  0   2
    FE80::C010:BFF:FE30:0
```

R2 and R3 each have two neighbors so that means all EIGRP neighbor adjacencies are working. Note that EIGRP also uses the link-local addresses...

Let's see if R1 has learned about the 2001::4/128 prefix from R4:

```
R1#show ipv6 route eigrp
IPv6 Routing Table - 2 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
       O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external
D    2001::4/128 [90/435200]
    via FE80::C012:DFF:FE5C:0, FastEthernet0/0
```

As you can see R1 has learned about 2001::4/128.



*This topology is a great example to see why link-local addresses are useful. If we were using IPv4 I would have had to configure a subnet between each router pair and you would see those subnets in the routing table. Thanks to the link-local addresses I don't care about the links between the routers.*

So why did EIGRP select this path? Let's look at the topology table!

```
R1#show ipv6 eigrp topology
IPv6-EIGRP Topology Table for AS(1)/ID(1.1.1.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
       r - reply Status, s - sia Status

P 2001::4/128, 1 successors, FD is 435200
    via FE80::C012:DFF:FE5C:0 (435200/409600), FastEthernet0/0
    via FE80::C011:BFF:FE30:0 (2323456/409600), Serial0/0
```

The path through R3 is the successor because it has the lowest feasible distance. Just like EIGRP for IPv4 we can enable unequal cost load balancing. To do this, we can use the variance command. The FD of the feasible successor has to be lower than the FD of the successor X multiplier.

The FD of the successor = 435200

The FD of the feasible successor = 2323456

$2323456 / 435200 = 5,3$ .

So if we set the variance at 6, our router should load balance over our feasible successor:

```
R1(config)#ipv6 router eigrp 1
R1(config-rtr)#variance 6
```

Let's verify our work:

```
R1#show ipv6 route eigrp
IPv6 Routing Table - 2 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary
       O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external

D 2001::4/128 [90/435200]
    via FE80::C012:DFF:FE5C:0, FastEthernet0/0
    via FE80::C011:BFF:FE30:0, Serial0/0
```

There we go, the router now uses both links.

Let's look at some more commands that we can use to verify everything is working:

```
R1#show ipv6 eigrp interfaces
IPv6-EIGRP interfaces for process 1
```

| Interface | Peers | Xmit Queue<br>Un/Reliable | Mean<br>SRTT | Pacing Time<br>Un/Reliable | Multicast<br>Flow Timer | Pending<br>Routes |
|-----------|-------|---------------------------|--------------|----------------------------|-------------------------|-------------------|
| Se0/0     | 1     | 0/0                       | 22           | 0/15                       | 111                     | 0                 |
| Fa0/0     | 1     | 0/0                       | 36           | 0/2                        | 50                      | 0                 |

The show ipv6 eigrp interfaces command is useful to see how many neighbors you have. You can add the detail keyword to see even more information:

```
R1#show ipv6 eigrp interfaces detail
IPv6-EIGRP interfaces for process 1
```

| Interface   | Peers | Xmit Queue<br>Un/Reliable | Mean<br>SRTT | Pacing Time<br>Un/Reliable | Multicast<br>Flow Timer | Pending<br>Routes |
|---|-------|---------------------------|--------------|----------------------------|-------------------------|-------------------|
| Se0/0   | 1     | 0/0                       | 22           | 0/15                       | 111                     | 0                 |
| <b>Hello interval is 5 sec</b><br>Next xmit serial <none><br>Un/reliable mcasts: 0/0 Un/reliable ucasts: 2/2<br>Mcast exceptions: 0 CR packets: 0 ACKs suppressed: 0<br>Retransmissions sent: 0 Out-of-sequence rcvd: 0<br><b>Authentication mode is not set</b><br>Use unicast   |       |                           |              |                            |                         |                   |
| Fa0/0   | 1     | 0/0                       | 36           | 0/2                        | 50                      | 0                 |
| <b>Hello interval is 5 sec</b><br>Next xmit serial <none><br>Un/reliable mcasts: 0/1 Un/reliable ucasts: 2/2<br>Mcast exceptions: 0 CR packets: 0 ACKs suppressed: 0<br>Retransmissions sent: 1 Out-of-sequence rcvd: 1<br><b>Authentication mode is not set</b><br>Use multicast |       |                           |              |                            |                         |                   |

You can see whether authentication is enabled or not but also the hello-interval.



*Just like OSPF, you can change the hello interval for EIGRP. One of the differences however is that for OSPF the hello interval has to match between neighbors. For EIGRP this is not required.*

Last but not least, show ip protocols gives us some EIGRP related information:

```
R1#show ipv6 protocols
IPv6 Routing Protocol is "connected"
IPv6 Routing Protocol is "static"
IPv6 Routing Protocol is "eigrp 1"
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 6
  Interfaces:
    FastEthernet0/0
    Serial0/0
  Redistribution:
    None
  Maximum path: 16
  Distance: internal 90 external 170
```

Here you can see the AS number, K –values, administrative distance, etc.

That's all we have about IPv6 routing! If you want to practice configuring OSPFv3, EIGRP or static routes you can try the following labs:

<http://gns3vault.com/IPv6/ipv6-static-route.html>

<http://gns3vault.com/IPv6/ipv6-eigrp.html>

<http://gns3vault.com/IPv6/ipv6-ospfv3.html>

## 24. Virtual Private Networks

In the previous chapter we looked at different WAN protocols that we can use for leased lines or shared networks (like frame-relay).

What if we wanted to use the Internet to connect different sites? One advantage of using the Internet is that it's much cheaper than for example a leased line. One of the downsides however is that the Internet is like the wild wild west...it's a public network and you have no idea who might be snooping on your data.

A VPN (Virtual Private Network) deals with this problem and tries to make a public network like the Internet as safe as a leased line. It does this by providing the following:

- **Confidentiality**
- **Authentication**
- **Integrity**
- **Anti-Replay**

**Confidentiality** means that nobody will be able to read the data that you are sending; we can do this by encrypting our traffic. **Authentication** means that we make sure that the sender of the data is a legitimate device. A simple method to do this is password authentication.

**Integrity** ensures that nobody has altered our data. We can do this with hashing. If you ever downloaded an ISO file from a website you might have seen some websites specify a MD5 Hash.

### 11.04

(Natty Narwal): April 2011 (Supported until October 2012)

| <i>md5 Hash</i>                  | <i>Version</i>                   |
|----------------------------------|----------------------------------|
| 8468300a61ea1e3b7607f46f7643b57a | ubuntu-11.04-alternate-amd64.iso |
| e6a29ce3dccb0ab12332036dcff7d9e4 | ubuntu-11.04-alternate-i386.iso  |
| 7de611b50c283c1755b4007a4feb0379 | ubuntu-11.04-desktop-amd64.iso   |
| 8b1085bed498b82ef1485ef19074c281 | ubuntu-11.04-desktop-i386.iso    |
| 355ca2417522cb4a77e0295bf45c5cd5 | ubuntu-11.04-server-amd64.iso    |
| b1a479c6593a90029414d201cb83a9cc | ubuntu-11.04-server-i386.iso     |

This is an example I got from the Ubuntu website (<http://www.ubuntu.com>). You can see they specify MD5 hashes for each ISO file you can download. What does it do?

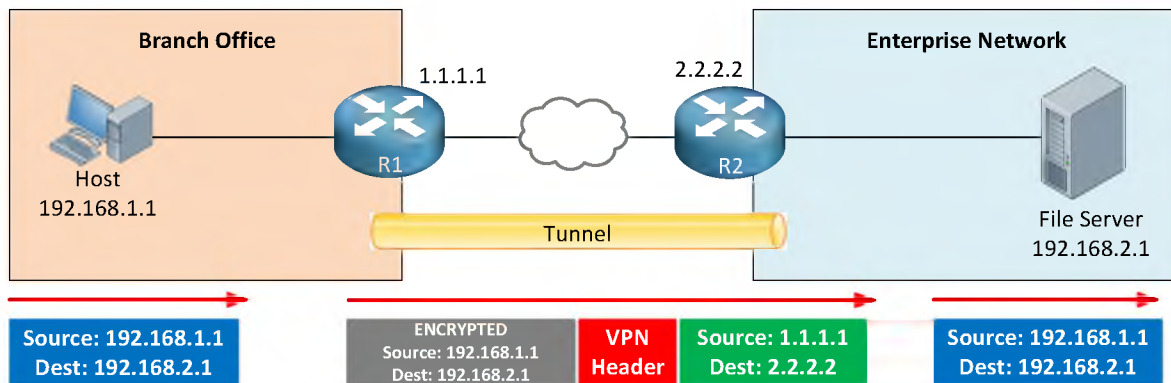
When the people from Ubuntu created their ISO files they created those MD5 hashes. Now you download one of their ISO files and use a MD5 program to calculate the hash for the ISO you just downloaded. If everything is ok you will get the same value which means

nothing in the file has changed since you downloaded it. If the hash is different than somebody altered the data or your file is corrupt!

**Anti-replay** prevents attackers to copy certain packets and send them later trying to impersonate a legitimate user. An attacker might be able to capture and figure out which IP packet(s) contain the encrypted and/or hashed password.

They might be unable to decrypt the contents of the IP packet(s) but they can try to send them again and fool a network device to grant access. Normally a sequence number is used to detect if certain IP packets have been received before or not.

Below is an example of what a typical VPN looks like:



Above you see a branch office that is connected to the enterprise network using an internet connection. The VPN has been established between the two routers. This is how it works:

1. The host on the left side has some data that has to reach the file server on the right side. The IP packet will have a source IP address of 192.168.1.1 and a destination IP address of 192.168.2.1. The host will send this IP packet to its default gateway (R1).
2. R1 will encrypt the packet and it will add a VPN header. The encrypted IP packet with the VPN header will be placed in a new IP packet that has a source IP address of 1.1.1.1 (that's the public IP address on R1) and a destination IP address of 2.2.2.2 (the public IP address of R2).
3. R2 will receive the IP packet, look at the VPN header and decrypts the IP packet. It will then be forwarded to the file server.
4. The file server receives a normal IP packet.

Putting an IP packet into another IP packet is called **tunneling**. In the example above we have a VPN tunnel between two routers but there are many other methods. Many mobile workers use VPN client software on their laptops, smartphones and tablets to create a secure connection to the enterprise network.

Using a VPN and the Internet is cheap, secure and also very scalable. It's easier to get an internet connection through DSL or cable somewhere than a leased line.

There are a number of different protocols that we can use to create a VPN. A very common method to build a VPN is **IPSEC**.



IPSEC is not a protocol but a *framework* for security on the network layer. It was created because there is no security on the network layer (layer 3). The IP protocol has nothing that will **authenticate** or **encrypt** our IP packets or check their **integrity**.

When you configure IPSEC you can choose between multiple protocols, for example for encryption you can select DES, 3DES or AES. For integrity you can choose between MD5 and SHA and for authentication you also have multiple options like password authentication or certificates.

IPSEC is very flexible, throughout the years newer and better protocols have been added and older protocols might be removed in the future. When you use your VPN client on your laptop or setup a VPN between two routers or firewalls, you are probably using IPSEC.

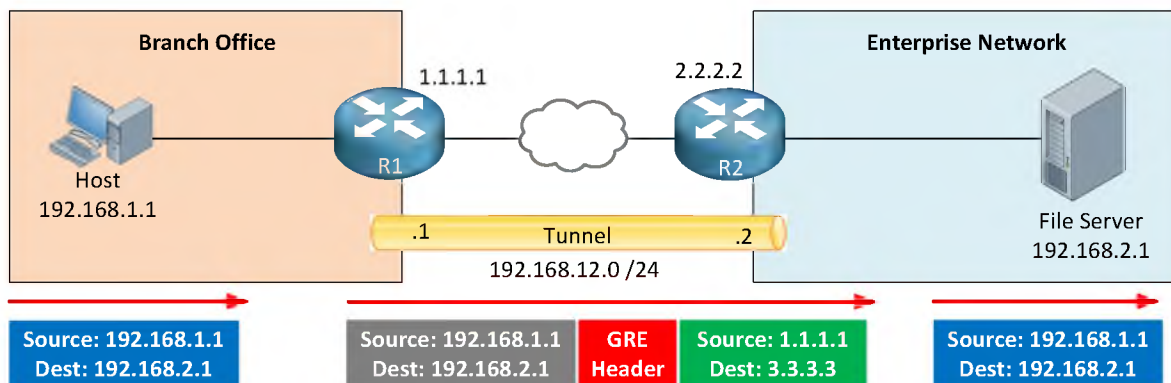
Another popular VPN method is **SSL VPN**. Maybe you haven't heard about SSL VPN before but you probably have used it before. When you use your webbrowser to setup a secure connection to a website you are using HTTPS, which uses SSL.

Besides using SSL for secure browsing it can also be used for secure remote access. One of the differences between using HTTPS and SSL VPN is that when you browse a website, the SSL encryption is probably done by the webserver, while for SSL VPN the encryption is done by a dedicated network like a firewall.

Luckily the CCNA routing & switching exam doesn't cover the configuration of a VPN with IPSEC or SSL VPN, there's a whole security track for that!

What you do need to know is **how to configure a tunnel**, without worrying about the security details like encryption, integrity and authentication. To do this we are going to discuss **GRE tunneling**.

GRE stands for Generic Routing Encapsulation and simply said, it puts IP packets into another IP packet so that they are tunneled. Let's look at an example:



The picture above is almost the same as the picture I just showed you but there are some differences. The GRE tunnel looks like a normal link but it's **virtual**. We have to configure an IP address on it.

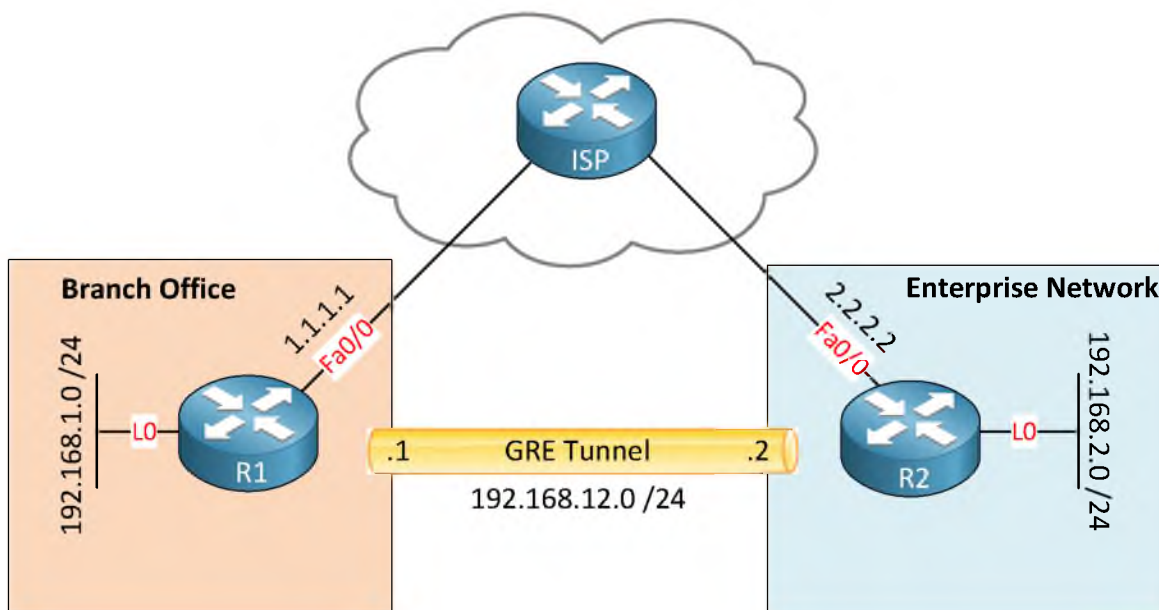
In the picture you can see that I added the 192.168.12.0 /24 subnet on the tunnel interface:

- R1 has IP address 192.16.12.1
- R2 has IP address 192.168.12.2

Now whenever the host sends something to the fileserver, R1 will put the IP packet from our host into another IP packet and it will use the IP addresses on the tunnel interface as the source and destination IP address.

You might be wondering what the point is of configuring a tunnel like this....if you take a close look at the picture you can see that the IP addresses from the host and fileserver are private IP addresses and these are **not routable** on the Internet. By building a tunnel between the routers (using their public IP addresses!) we can encapsulate the traffic between the host and fileserver and route it over the Internet.

Let's take a look at the configuration of a GRE tunnel so you can see how it actually works, here's the topology again:



R1 and R2 are able to reach each other through the ISP router. To make things a little bit more interesting we will not only configure the GRE tunnel but also configure OSPF so that R1 is able to reach the 192.168.0 /24 network and R2 is able to reach the 192.168.1.0 /24 network. Let's get started!

Before we start with the GRE tunnel configuration it's not a bad idea to check if R1 can reach R2:

```
R1#ping 2.2.2.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/32/44 ms
```

No problems there so let's create the GRE tunnel:

```
R1(config)#interface tunnel 1
R1(config-if)#tunnel source 1.1.1.1
R1(config-if)#tunnel destination 2.2.2.2
```

```
R2(config)#interface tunnel 1
R2(config-if)#tunnel source 2.2.2.2
R2(config-if)#tunnel destination 1.1.1.1
```

First you have to specify a tunnel interface number, you can pick whatever you like. The next step is to configure the source and destination IP address for the tunnel. I'm using IP address 1.1.1.1 on R1 and 2.2.2.2 on R2 to build the tunnel. You will see the following message on the consoles:

```
R1#
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnell1, changed state to up
```

```
R2#
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnell1, changed state to up
```

The tunnel is up, that's looking good. We still have to configure an IP address on it however:

```
R1(config)#interface tunnell1
R1(config-if)#ip address 192.168.12.1 255.255.255.0
```

```
R2(config)#interface tunnell1
R2(config-if)#ip address 192.168.12.2 255.255.255.0
```

Just like any other interface we can assign an IP address to the GRE tunnel.

Let's see if we can ping between R1 and R2 using these IP addresses...

```
R1#ping 192.168.12.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.12.2, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/52/72 ms
```

No problems there, we have reachability through the tunnel! Besides the ping there are a couple of useful commands to verify that the tunnel is working:

```
R1#show ip int brief
Interface                IP-Address      OK? Method Status
Protocol
FastEthernet0/0          1.1.1.1         YES manual up
up
FastEthernet0/1          unassigned      YES unset  administratively down
down
Loopback0                192.168.1.1     YES manual up
up
Tunnel1                 192.168.12.1    YES manual up
up
```

The show ip interface brief command shows the tunnel interface and its IP address. You can see that the status says up/up.

```
R1#show int tunnel 1
Tunnel1 is up, line protocol is up
  Hardware is Tunnel
  Internet address is 192.168.12.1/24
  MTU 1514 bytes, BW 9 Kbit/sec, DLY 500000 usec,
    reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation TUNNEL, loopback not set
  Keepalive not set
  Tunnel source 1.1.1.1, destination 2.2.2.2
  Tunnel protocol/transport GRE/IP
    Key disabled, sequencing disabled
    Checksumming of packets disabled
  Tunnel TTL 255
  Fast tunneling enabled
  Tunnel transmit bandwidth 8000 (kbps)
  Tunnel receive bandwidth 8000 (kbps)
  Last input 00:02:40, output 00:02:40, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/0 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
    5 packets input, 620 bytes, 0 no buffer
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
    5 packets output, 620 bytes, 0 underruns
```

```
0 output errors, 0 collisions, 0 interface resets
0 unknown protocol drops
0 output buffer failures, 0 output buffers swapped out
```

**Show interface tunnel** is useful to see the status of the tunnel (up/up) but also to check the source and destination IP address that are used to build the tunnel and the tunnel type (GRE).

With the tunnel up and running we can continue and configure OSPF so that R1 and R2 can reach each other's networks. This is how we do it:

```
R1(config)#router ospf 1
R1(config-router)#network 192.168.1.0 0.0.0.255 area 0
R1(config-router)#network 192.168.12.0 0.0.0.255 area 0
```

I will use two network commands. The first one is to advertise the 192.168.1.0 /24 network to R2 and the second one is to make sure OSPF hello packets are sent through the GRE tunnel to R2 so that we can form an OSPF neighbor adjacency.

```
R2(config)#router ospf 1
R2(config-router)#network 192.168.2.0 0.0.0.255 area 0
R2(config-router)#network 192.168.12.0 0.0.0.255 area 0
```

We will apply a similar configuration to R2. After doing this you will see that R1 and R2 have become OSPF neighbors:

```
R1#
%OSPF-5-ADJCHG: Process 1, Nbr 192.168.2.2 on Tunnel1 from LOADING to FULL,
Loading Done
R2#
%OSPF-5-ADJCHG: Process 1, Nbr 192.168.1.1 on Tunnel1 from LOADING to FULL,
Loading Done
```

Note that the tunnel interface has been used to form the OSPF neighbor adjacency. Now let's take a look at the routing tables:

```
R1#show ip route ospf
    192.168.2.0/32 is subnetted, 1 subnets
O       192.168.2.2 [110/11112] via 192.168.12.2, 00:02:16, Tunnel1
```

As you can see R1 now knows about the 192.168.2.2 network and it has learned this network through the tunnel1 interface. The next hop IP address is 192.168.12.2 (the IP address of the tunnel interface on R2).

```
R2#show ip route ospf
    192.168.1.0/32 is subnetted, 1 subnets
O       192.168.1.1 [110/11112] via 192.168.12.1, 00:03:25, Tunnel1
```

R2 also has learned about the network behind R1 through the tunnel interface. Let's try a quick ping to verify connectivity!

```
R1#ping 192.168.2.2 source 192.168.1.1
```

Type escape sequence to abort.

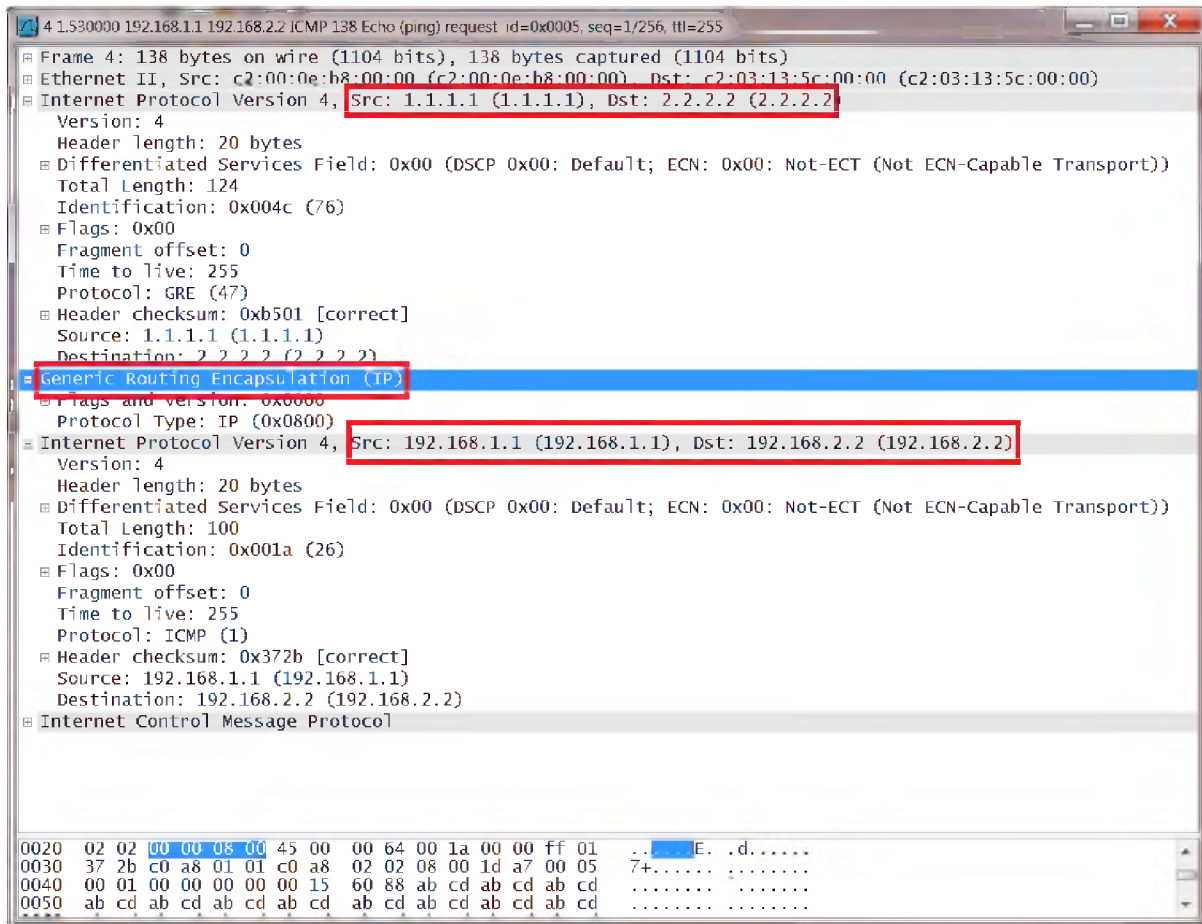
Sending 5, 100-byte ICMP Echos to 192.168.2.2, timeout is 2 seconds:

Packet sent with a source address of 192.168.1.1

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 36/52/60 ms

And we are able to ping from 192.168.2.0 /24 from 192.168.1.0 /24. In case you are wondering, here's a wireshark capture that shows what a tunneled IP packet looks like:



In this wireshark capture of the ping that I just sent you can see the IP packet with source IP address 1.1.1.1 and destination IP address 2.2.2.2, then the GRE header and finally the encapsulated IP packet that carries our ping from 192.168.1.1 to 192.168.2.2.

This concludes our chapter about VPNs and tunneling. If you want to practice how to configure a GRE tunnel then you can try the following lab:

<http://gns3vault.com/Tunneling-GRE/gre-tunnel-basic.html>

## 25. Network Management

In this chapter we are going to talk about network management. We will look at different protocols that help to manage your entire network using a **NMS (Network Management Software)**, how to copy IOS and configuration files from and to your routers and switch and we will discuss CDP which is a protocol that detects Cisco devices that are connected to your router or switch.

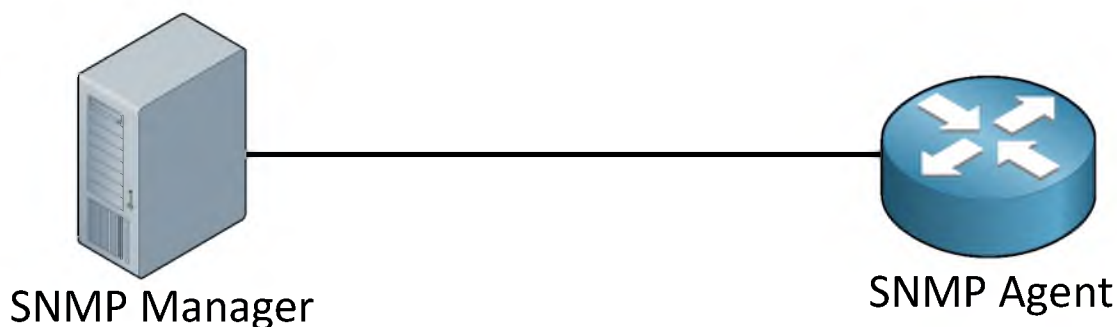
Imagine you have a large network that has many switches and routers, a dozen servers and hundreds of workstations...wouldn't it be great if you could monitor all those devices somehow? Using NMS it's possible to monitor all devices in your network. Whenever something bad happens (like an interface that goes down) you will receive an e-mail or text message on your phone so you can respond to it immediately.

Sounds good?

Back in the 80s some smart folks figured out that we should have something to monitor all IP based network devices. The idea was that most devices like computers, printers and routers share some characteristics. They all have an interface, an IP address, a hostname, buffers and so on.

They created a database with variables that could be used to monitor different items of network devices and this resulted in **SNMP (Simple Network Management Protocol)**.

SNMP runs on the application layer and consists of a **SNMP manager** and a **SNMP agent**. The SNMP manager is the software that is running on a pc or server that will monitor the network devices, the SNMP agent runs on the network device.



The database that I just described is called the **MIB (Management Information Base)** and an example of a variable could be the interface status on the router (up or down) or perhaps the CPU load at a certain moment. The SNMP manager will be able to send periodic polls to the router and it will use store this information. This way it's possible to create graphs to show you the CPU load or interface load from the last 24 hours, week, month or whatever you like.

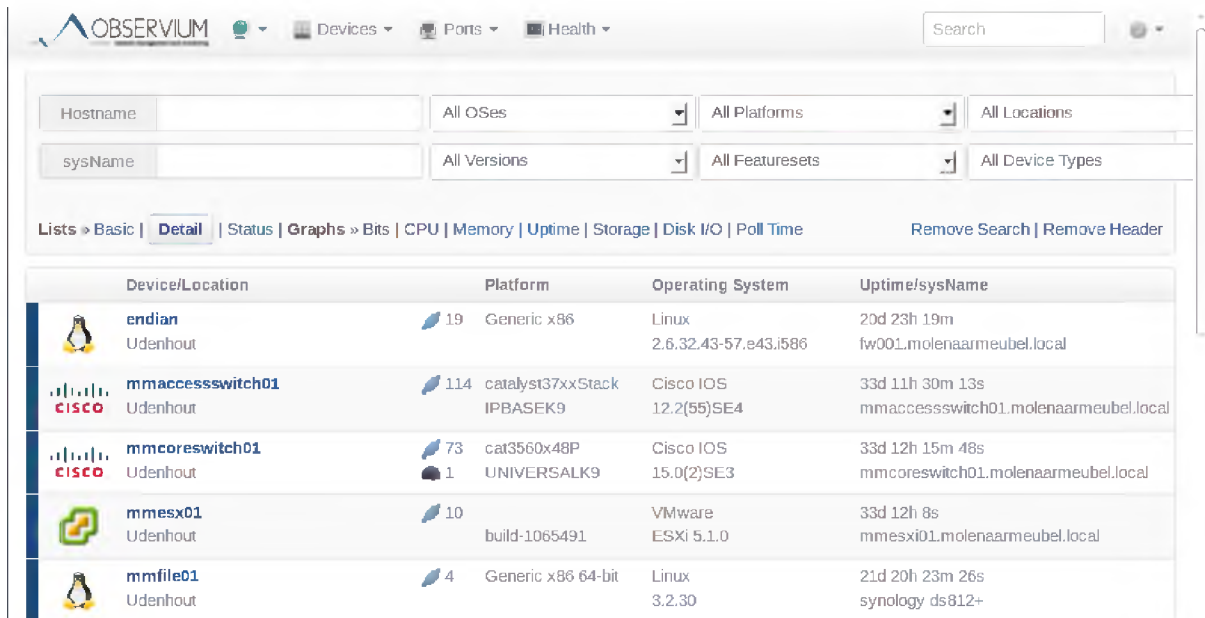
It's also possible to configure your network devices through SNMP. This might be useful to configure a large amount of switches or routers from your network management software so you don't have to telnet/ssh into each device separately to make changes.

The packet that we use to poll information is called a **SNMP GET message** and the packet that is used to write a configuration is a **SNMP SET message**.






To give you an example of what a NMS looks like, I'll show you some screenshots of Observium.

Observium is a free SNMP based network monitoring platform which can monitor Cisco, Linux, Windows and some other devices. It's easy to install so if you never worked with SNMP or monitoring network devices before I can highly recommend giving it a try. You can download it at <http://www.observium.org>.

Here's what it looks like:



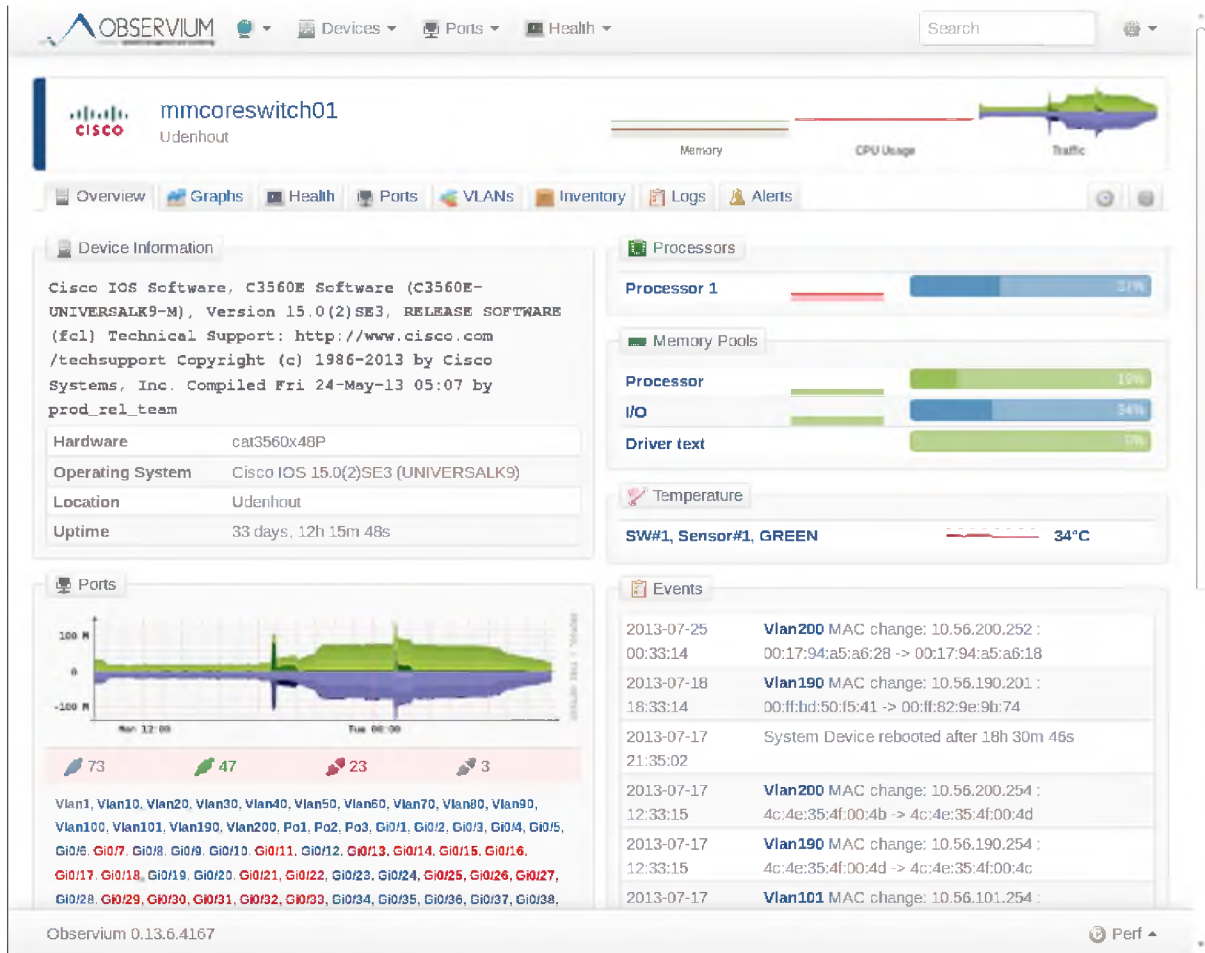
The screenshot shows the Observium web interface. At the top, there's a navigation bar with 'OBSERVIMUM' logo, a search bar, and tabs for 'Devices', 'Ports', and 'Health'. Below this is a filter section with dropdown menus for 'All OSes', 'All Platforms', 'All Locations', 'All Versions', 'All Featuresets', and 'All Device Types'. A 'Search' button is also present. Below the filters, there's a 'Lists' section with tabs for 'Basic' and 'Detail'. The 'Detail' tab is selected, showing a table of devices. The table has columns for 'Device/Location', 'Platform', 'Operating System', and 'Uptime/sysName'. The table lists five devices: 'endian' (Linux), 'mmaccessswitch01' (Cisco IOS), 'mmcoreswitch01' (Cisco IOS), 'mmesx01' (VMware ESXi), and 'mmfile01' (Linux).

| Device/Location  | Platform                          | Operating System               | Uptime/sysName  |
|--|-----------------------------------|--------------------------------|---|
|  <b>endian</b><br>Udenhout            | 19 Generic x86                    | Linux<br>2.6.32.43-57.e43.i586 | 20d 23h 19m<br>fw001.molenaar.meubel.local                |
|  <b>mmaccessswitch01</b><br>Udenhout | 114 catalyst37xxStack<br>IPBASEK9 | Cisco IOS<br>12.2(55)SE4       | 33d 11h 30m 13s<br>mmaccessswitch01.molenaar.meubel.local |
|  <b>mmcoreswitch01</b><br>Udenhout  | 73 cat3560x48P<br>1 UNIVERSALK9   | Cisco IOS<br>15.0(2)SE3        | 33d 12h 15m 48s<br>mmcoreswitch01.molenaar.meubel.local   |
|  <b>mmesx01</b><br>Udenhout         | 10 build-1065491                  | VMware<br>ESXi 5.1.0           | 33d 12h 8s<br>mmesx01.molenaar.meubel.local               |
|  <b>mmfile01</b><br>Udenhout        | 4 Generic x86 64-bit              | Linux<br>3.2.30                | 21d 20h 23m 26s<br>synology ds812+                        |

Above you see an overview of all the devices that our NMS manages. There are two linux devices, two Cisco devices and there's a VMWare ESXi server. You can see the uptime of all devices.



Let's take a closer look at one of the Cisco devices:



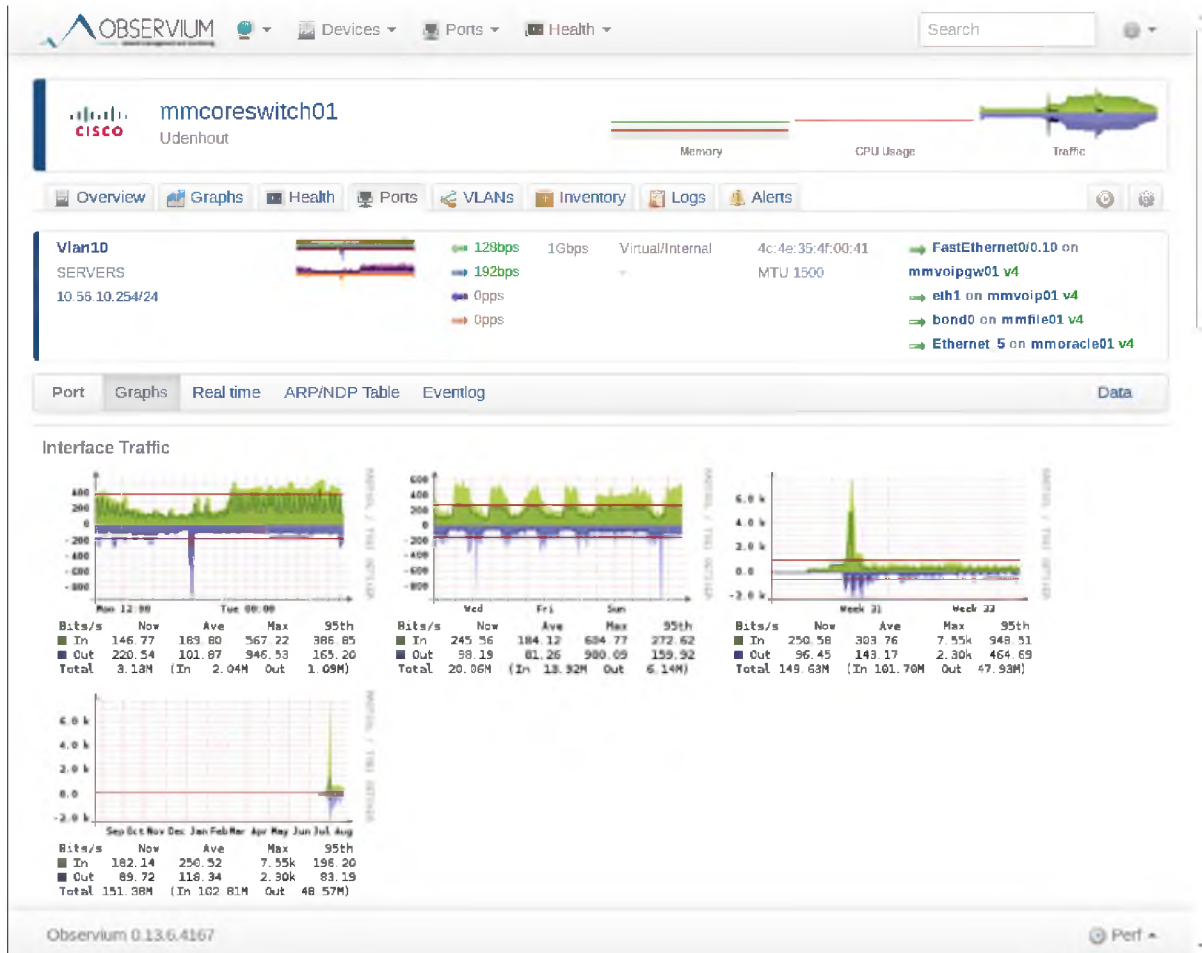
This switch is called "mmcoreswitch01" and it's a Cisco Catalyst 3560E. It gives us a nice overview of the CPU load, the temperature and the interfaces that are up or down.

Let's take a closer look at the temperature of this switch:



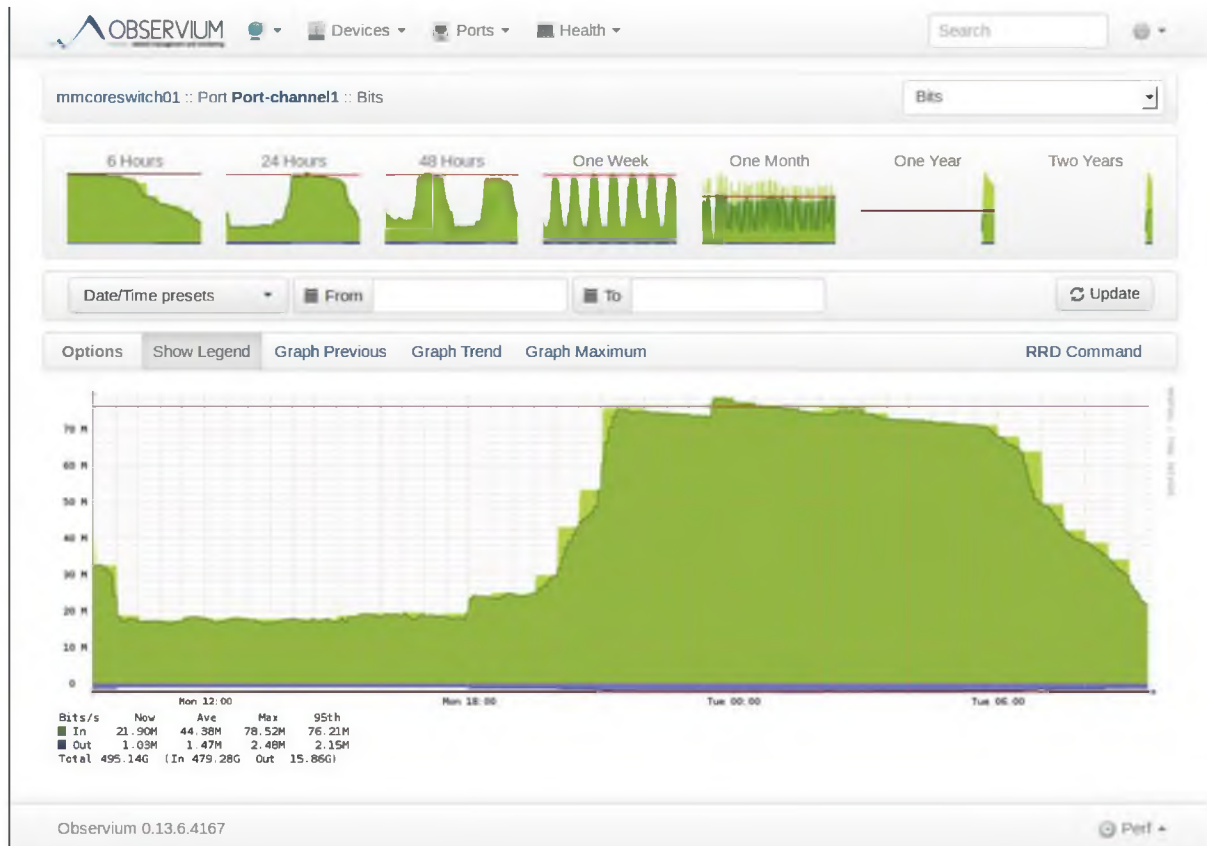
Here's the temperature of this switch from the last month. When the temperature exceeds a certain value (let's say 50 degrees celcius) then we can tell our NMS to send us an e-mail.

Let's take a look at an interface of this switch:



Here's an overview of the VLAN 10 interface. You can see how much traffic is sent and received on this interface.

We can zoom in on one of the graphs if we want:



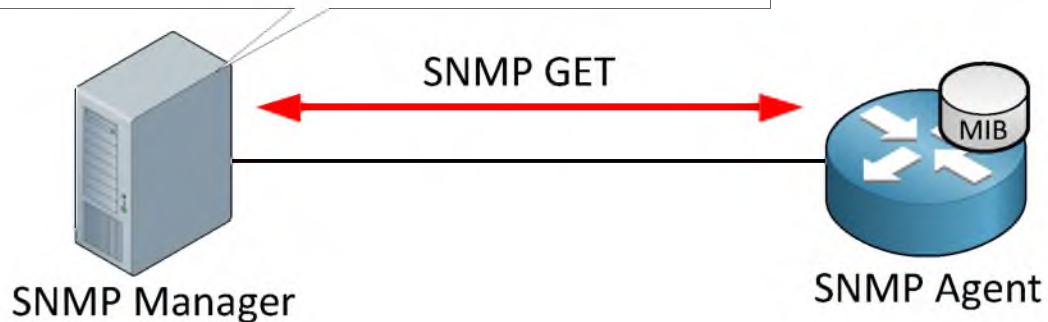
This gives a nice overview of how much traffic was sent in the last 24 hours of this particular interface.

I hope this gives you an idea of what a NMS looks like and why this might be useful. If you want to take a look at Observium yourself you can use the live demo on their website:

<http://demo.observium.org/>

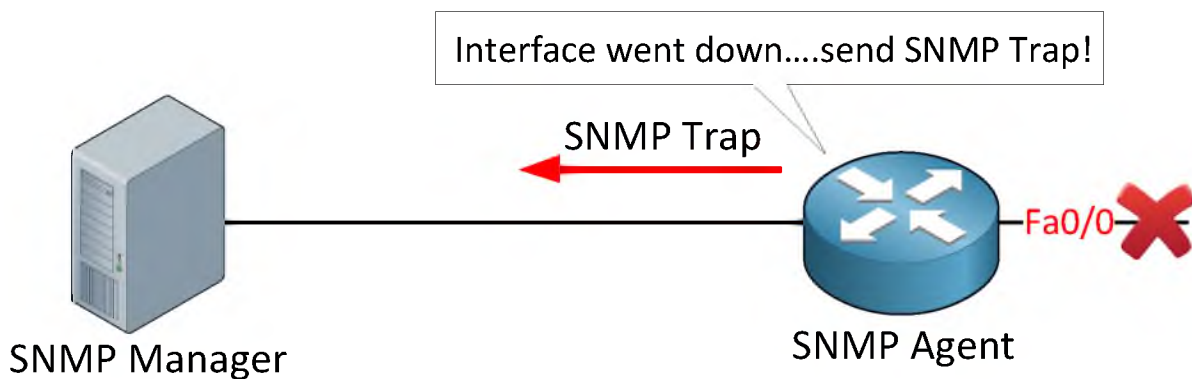
All the information that Observium shows us is retrieved by using SNMP GET messages.

Check MIB variable to discover current temperature.



The NMS will send SNMP GET messages to request the current state of certain MIB variables every few minutes or so. This is great for monitoring the temperature or traffic statistics but the downside of using these SNMP GET messages is that it might take a few minutes for the NMS to discover that an interface is down.

Besides using SNMP GET messages, a SNMP agent can also send **SNMP traps**. A trap is a notification that it **sents immediately** as soon as something occurs, for example an interface that goes down.



As soon as something bad happens (like the interface that goes down) the SNMP agent will send a SNMP trap immediately to the NMS. The NMS will respond by sending you an e-mail, text message or a notification on the screen.

These SNMP trap messages sound like a good idea but there's one problem with them...there is **no acknowledgment** for the SNMP trap, so you never know if the trap made it to the NMS or not. SNMP version 3 deals with this problem with an alternative message which uses an acknowledgment called the **inform message**.

SNMP has three versions:

- Version 1
- Version 2c
- Version 3

Version 1 is so old that it's very unlikely that you will encounter it on a production network. Version 1 and 2 both use **community-strings** as a password to authenticate access to the

SNMP agent. These community-strings are sent in **clear-text** which makes SNMP version 1 and 2 very insecure.

SNMP version 3 is a better choice nowadays because it supports username based authentication instead of a community-string and also supports encryption. There are 3 different security modes:

- **noAutoNoPriv**: username authentication but no encryption.
- **authNoPriv**: MD5 or SHA authentication but no encryption.
- **authPriv**: MD5 or SHA authentication and encryption.

Even if you decide to use SNMP version 3 without authentication or encryption, you can still track activity down to a username.

Now you have an idea what SNMP is about, let's see how we can configure it on a Cisco router. I'll give you an example for SNMP version 2c:

```
Router(config)#snmp-server community MYSTRING ro 10
Router(config)#access-list 10 permit host 192.168.1.2
```

The first thing to do is configure a community-string. I called mine "MYSTRING" and to make things a little bit more secure I made it read-only and attached an access-list to it so that only my NMS at IP address 192.168.1.2 is able to poll this router.

If I want to configure this router from my NMS I can make it read-write by exchanging "ro" for "rw".

Technically configuring the community-string is the only thing you have to do to enable SNMP but it's a good idea to add the following information:

```
Router(config)#snmp-server location Amsterdam
Router(config)#snmp-server contact info@gns3vault.com
```

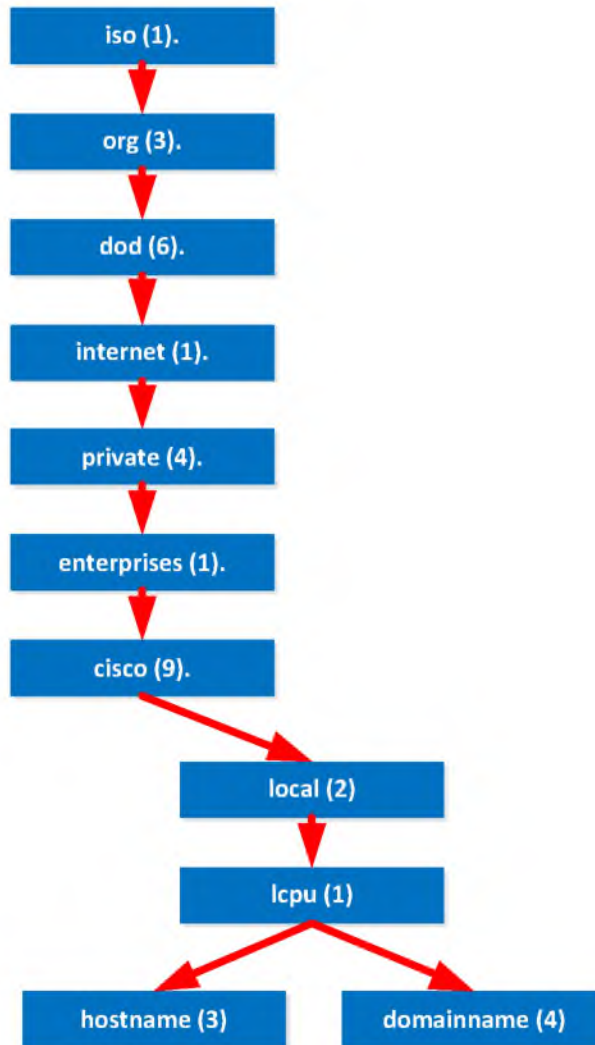
These two commands will allow the NMS to read the location and contact information of your router.

This router can now be managed from a MIB but how do we know which variables we need to monitor things like the interface status, temperature, number of packets sent and so on?

To understand how this works we'll take a closer look at the MIB variables. Each variable in the MIB is called an **OID (Object ID)**.

Since there are so many OIDs, the MIB is organized into a hierarchy that looks like a tree. In this tree you will find a number of branches with OIDs that are based on RFC standards but you will also find some vendor specific variables. Cisco for example, has variables to monitor EIGRP and other Cisco protocols.

Let me give you an example of this tree by showing where the 'hostname' and 'domainname' objects are located. These objects can be used to discover the hostname and domainname of the router.



The tree starts with the “iso” branch and then we drill our way down to org, dod, internet, private, enterprises, cisco, local, lcpu and there we find the hostname and domainname objects. Note that the branches have numbers...instead of typing out the names I can just use the numbers.

1.3.6.1.4.1.9.2.1.3 will be used to get information about the hostname and  
1.3.6.1.4.1.9.2.1.4 for the domainname.

The MIB is huge and knowing where to find the right objects can be troublesome, that’s why most NMSes have a nice GUI that lets you select the things you want to monitor without having to worry about the object numbers.

If you want to test SNMP you don’t have to install a NMS, you can use SNMPGET which is a free tool that you can download here:

<http://sourceforge.net/projects/net-snmp/>

Here’s an example of SNMPGET to query the router that I just configured for SNMP:



```
# snmpget -v2c -c MYSTRING 192.168.1.1 1.3.6.1.4.1.9.2.1.3.0
iso.3.6.1.4.1.9.2.1.3.0 = STRING: "Router"
```

The community string that I used is MYSTRING, the IP address of the router is 192.168.1.1 and the object I'm interested in is 1.3.6.1.4.1.9.2.1.3. As a result the router reports its hostname. Here's another example for the domainname:

```
# snmpget -v2c -c MYSTRING 192.168.1.1 1.3.6.1.4.1.9.2.1.4.0
iso.3.6.1.4.1.9.2.1.4.0 = STRING: "localdomain"
```

I didn't configure any domainname on this router so the result is "localdomain".

That's all I wanted to show you on SNMP for now. Something else we have to consider when managing our network is **syslog**.

Even if you never heard about syslog before, you have seen it for sure. The messages like the one below are produced by syslog:

```
*Mar  1 00:00:22.015: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed
state to down
```

These messages are quite useful when you are logged in on the router or switch because they are shown by default on the console. What if the interface went down and we are not around?

This is no problem as we can export these syslog messages to an external syslog server. Note that there is a timestamp in front of these messages, if you want your logging messages to be useful you have to make sure that the time and date has been configured correctly on your router.

You have seen the clock command before but it's better to use a NTP (Network Time Protocol) server to make sure all devices in your network have their clocks synchronized. It's easy to configure this on a Cisco device:

```
Router(config)#ntp server pool.ntp.org
```

The **ntp server** command lets me configure an NTP server. I'm using one from ntp.org as they are free to use. You can verify that the clock is synchronized like this:

```
Router#show ntp status
Clock is synchronized, stratum 3, reference is 217.121.137.227
nominal freq is 250.0000 Hz, actual freq is 250.0000 Hz, precision is 2**18
reference time is D35E243C.A5B38A7C (13:03:56.647 UTC Wed May 16 2012)
clock offset is -10.1544 msec, root delay is 29.10 msec
root dispersion is 52.75 msec, peer dispersion is 2.40 msec
```

Above you can see that the clock has been synchronized. Instead of timestamps we can also use sequence numbers:

```
Router(config)#no service timestamps
Router(config)#service sequence-numbers
```



The same message about the interface will now look like this:

```
000009: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to down
```

There are different severity levels for logging information. An interface that goes down is probably more important to know than a message that tells us we exited the global configuration.

Here are the severity levels:

0. Emergencies
1. Alerts
2. Critical
3. Errors
4. Warnings
5. Notifications
6. Informational
7. Debugging

By default you'll see all of these messages on the console. If you don't want to see everything you can change this behavior:

```
Router(config)#logging console errors
```

For example you can use configure the **logging console** command so it only shows you severity level 3 (errors) and lower. The message about the interface that was going down is a **notification** in case you were wondering.

```
Router#show logging history
Syslog History Table:1 maximum table entries,
saving level warnings or higher
29 messages ignored, 11 dropped, 0 recursion drops
2 table entries flushed
SNMP notifications not enabled
  entry number 3 : LINK-3-UPDOWN
    Interface FastEthernet0/0, changed state to up
    timestamp: 1688
```

We can use the **show logging history** command to see the logging history of this Cisco router. It doesn't store everything. You can see it says "saving level warnings or higher". This logging information is saved in the RAM of your device. Once you reboot it you will lose this logging history.

```
Router(config)#logging buffered 4096
```

You can change the size of the buffer if needed. As soon as the buffer is full old logging information will be discarded. In the example above we have 4096 bytes we can use for logging information.

If you want to see the logging information you can look at them with the show logging command:

```
R1#show logging
Syslog logging: enabled (12 messages dropped, 0 messages rate-limited,
                  0 flushes, 0 overruns, xml disabled, filtering disabled)

No Active Message Discriminator.

No Inactive Message Discriminator.

Console logging: level debugging, 22 messages logged, xml disabled,
                  filtering disabled
Monitor logging: level debugging, 0 messages logged, xml disabled,
                  filtering disabled
Buffer logging:  level debugging, 4 messages logged, xml disabled,
                  filtering disabled
Logging Exception size (4096 bytes)
Count and timestamp logging messages: disabled
Persistent logging: disabled

No active filter modules.

ESM: 0 messages dropped

Trap logging: level informational, 27 message lines logged

Log Buffer (4096 bytes):

*Mar  1 00:13:25.883: %SYS-5-CONFIG I: Configured from console by console
*Mar  1 00:13:31.219: %LINK-5-CHANGED: Interface FastEthernet0/0, changed
state to administratively down
*Mar  1 00:13:32.219: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/0, changed state to down
```

It's not a good idea to store logging information locally on your device. One reboot and you'll lose valuable information. It's best to use an external server for this.

```
Router(config)#logging 192.168.1.100
```

Use the **logging** command to set the IP address for your logging server. All logging information will be sent towards this server with the exception of debugging (level 7) messages by default.

Normally you probably only want to see debug information on your console or telnet/SSH session but you can store it in on your logging server too if you want:

```
Router(config)#logging trap 7
```

You need to use the **logging trap** command to 7 so it will also store debug information on your logging server.

If you want to try storing some syslog messages to an external syslog server you can take a look at "Solarwinds Kiwi Syslog server". It's a syslog server that has a 30 day trial which is perfect to test the commands that you have just seen yourself. You can download it from:

<http://www.kiwisyslog.com>

The next item we are going to look at is **netflow**. Tools like SNMP are able to tell us how much traffic is going through our interface but they don't really tell what kind of traffic it is. Simply said, netflow gives you very accurate information about all IP packets that are flowing through your network.

A "flow" is a stream of packets between a **source and destination**. This means that the 4 following items have to be the same in order to group IP packets to the same flow:

- Source IP address
- Destination IP address
- Source port number
- Destination port number

A host that creates a connection to a webserver will be seen as a single flow. When this host creates another connection to the same webserver then this will be seen as another flow since the source port number will be different for the second TCP connection.

Throughout the years there have been multiple versions of netflow. Besides the source and destination there are some other fields that could be used to classify packets to a certain flow:

- Layer 3 protocol type
- Type of Service marking (ToS byte)
- Input logical Interface

The layer 3 protocol type is the layer 3 header after the IP header. The ToS byte is used for QoS (Quality of Service) which is meant to give priority to some IP packets over others.

Netflow is very useful, you can use it to detect what kind of traffic is running through your network, it's used to bill customers based on network usage and even to detect DoS attacks.

When you enable netflow on a router you have to do a couple of things:

- Configure the direction to monitor, inbound, outbound or both.
- Configure the netflow version.
- Configure a netflow collector.

The **netflow collector** is an external server where you can send your netflow statistics to. This system will collect all netflow data and organizes it in a useful way. A netflow collector can give you information about the most visited websites, the most downloaded content, top talkers in your network and much more. It will depend on the netflow version however how much information it can give you.

There is plenty of free netflow collector software out there. If you want an example can do a google search for "[Freeware Netflow Software](#)" which gives you an overview of free collectors on the Cisco website.

The configuration of netflow is quite easy, here's an example:

```
Router(config)#interface fastEthernet 0/0
Router(config-if)#ip flow ingress
Router(config-if)#ip flow egress
```

This is how you enable netflow on an interface, in this example I'm enabling it inbound and outbound.

The version of netflow can be configured like this:

```
Router(config)#ip flow-export version 9
```

Here's how to configure the netflow collector:

```
Router(config)#ip flow-export destination 192.168.1.2 100
```

And you can see some netflow information on the local router:

```
MMVOIPGW01#show ip cache flow
IP packet size distribution (2933 total packets):
 1-32  64  96 128 160 192 224 256 288 320 352 384 416 448 480
 .000 .000 .004 .000 .000 .000 .995 .000 .000 .000 .000 .000 .000 .000 .000

 512 544 576 1024 1536 2048 2560 3072 3584 4096 4608
 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000

IP Flow Switching Cache, 278544 bytes
 3 active, 4093 inactive, 4 added
 195 age polls, 0 flow alloc failures
 Active flows timeout in 30 minutes
 Inactive flows timeout in 15 seconds
IP Sub Flow Cache, 34056 bytes
 0 active, 1024 inactive, 0 added, 0 added to flow
 0 alloc failures, 0 force free
 1 chunk, 1 chunk added
 last clearing of statistics never
```

| Protocol  | Total Flows | Flows /Sec | Packets /Flow | Bytes /Pkt | Packets /Sec | Active(Sec) /Flow | Idle(Sec) /Flow |
|-----------|-------------|------------|---------------|------------|--------------|-------------------|-----------------|
| UDP-other | 1           | 0.0        | 2             | 565        | 0.0          | 0.0               | 15.5            |
| Total:    | 1           | 0.0        | 2             | 565        | 0.0          | 0.0               | 15.5            |

| SrcIf | SrcIPaddress | DstIf | DstIPaddress | Pr | SrcP | DstP | Pkts |
|-------|--------------|-------|--------------|----|------|------|------|
| SrcIf | SrcIPaddress | DstIf | DstIPaddress | Pr | SrcP | DstP | Pkts |
| Fa0/1 | 10.56.10.31  | Fa0/0 | 10.56.10.30  | 11 | 448A | 7680 | 3000 |

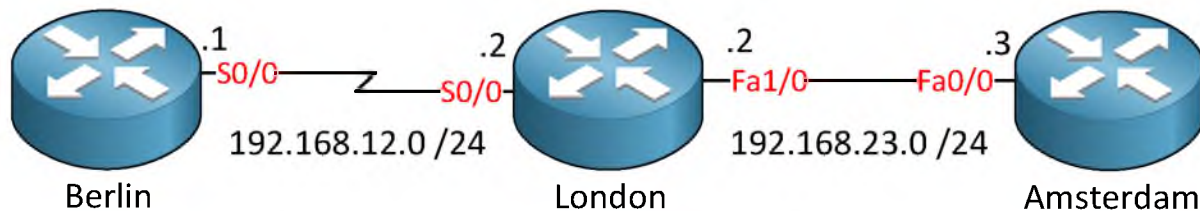
This tells us that netflow is working and monitoring packets, 2933 so far. It shows us the size of the packets and at the bottom you can see a flow.

That's all you need to know about netflow for CCNA, the next tool we will discuss is CDP.

**CDP (Cisco Discovery Protocol)** to help you build network maps, we'll take a detailed look at the router bootup process and you will learn the easiest method to back up your configurations and/or IOS images.

Most networks have multiple switches and/or routers and to make our life easier it's good to have a network map that shows us how everything is connected to each other, what kind of devices we have, to what VLAN they belong and the IP addresses that we are using. CDP is a Cisco protocol that runs on all Cisco devices that helps us **discover Cisco devices on the network**. CDP is Cisco proprietary, runs on the data-link layer and is enabled by default.

Let's take a look at a network map:



Above we have 3 routers. Now if I had no idea what the network looked like we could use CDP to build the network map that you see above. Let me show you how:

```
Berlin#show cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater
```

| Device ID | Local Intrfce | Holdtme | Capability | Platform | Port ID |
|-----------|---------------|---------|------------|----------|---------|
| London    | Ser 0/0       | 167     | R S I      | 3640     | Ser 0/0 |

Use the **show cdp neighbors** command to see all **directly connected** neighbors. Above you see that router Berlin is connected to router London and you can also see the platform (3640 router) and the interfaces on both sides. Let me show you the other routers as well:

```
London#show cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater
```

| Device ID | Local Intrfce | Holdtme | Capability | Platform | Port ID |
|-----------|---------------|---------|------------|----------|---------|
| Berlin    | Ser 0/0       | 144     | R S I      | 3640     | Ser 0/0 |
| Amsterdam | Fas 1/0       | 164     | R S I      | 3640     | Fas 0/0 |

```
Amsterdam#show cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater
```

| Device ID | Local Intrfce | Holdtme | Capability | Platform | Port ID |
|-----------|---------------|---------|------------|----------|---------|
| London    | Fas 0/0       | 135     | R S I      | 3640     | Fas 1/0 |

Now we have all the information we need to build a network map with the router names and interfaces. CDP can tell us even more however...

```
Berlin#show cdp neighbors detail
-----
Device ID: London
```

```
Entry address(es):
  IP address: 192.168.12.2
Platform: Cisco 3640, Capabilities: Router Switch IGMP
Interface: Serial0/0, Port ID (outgoing port): Serial0/0
Holdtime : 136 sec

Version :
Cisco IOS Software, 3600 Software (C3640-JK9O3S-M), Version 12.4(16), RELEASE
SOFTWARE (fc1)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2007 by Cisco Systems, Inc.
Compiled Wed 20-Jun-07 11:43 by prod_rel_team

advertisement version: 2
VTP Management Domain: ''
```

Use **show cdp neighbors detail** to reveal even more information. For example you can see the IP address and the IOS version. This can be very useful to us but it's also a security risk. By default CDP is enabled and runs on all interfaces so it might be a good idea to disable it on certain interfaces:

```
Berlin(config)#interface serial 0/0
Berlin(config-if)#no cdp enable
```

This is how you can disable it for a single interface, just type **no cdp enable**. This is how you can do it globally for all interfaces:

```
Berlin(config)#no cdp run
```

Hit **no cdp run** to disable it completely.

That's it for CDP. In the remaining of this chapter we'll talk about the router components, how to backup your configuration and how to deal with different IOS images.

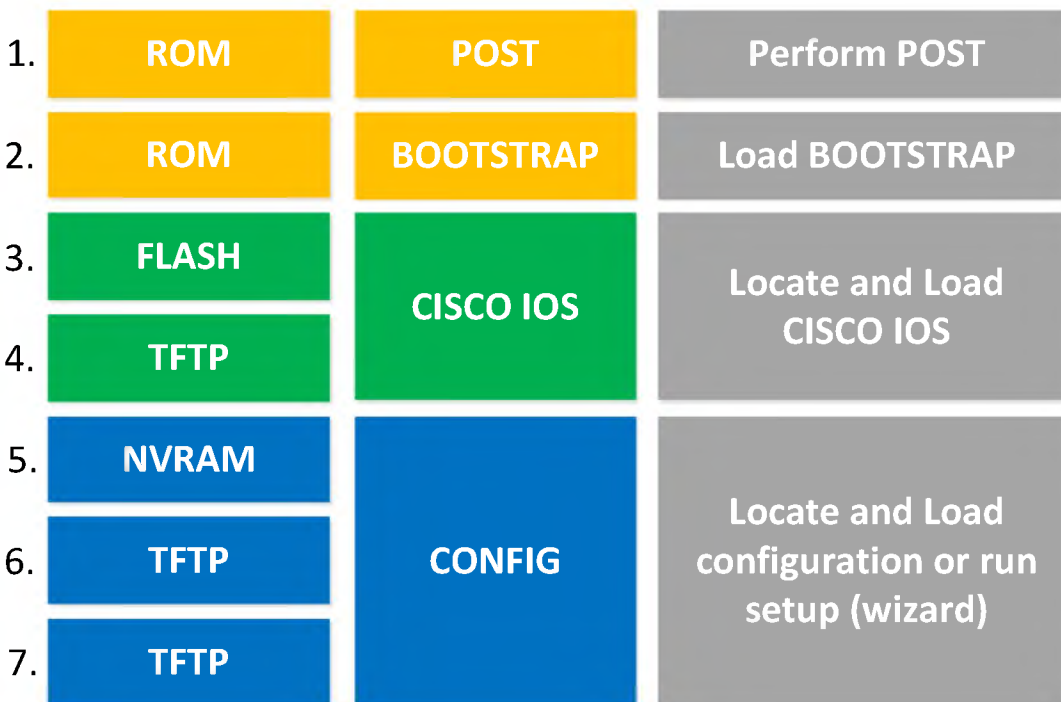
In the beginning of this book you learned that:

- The running-configuration is in the RAM.
- The startup-configuration is in the NVRAM.
- The IOS image is stored on the flash.
- The running IOS is in the RAM.

Besides the RAM, NVRAM and flash a router also has a **ROM** (Read-Only Memory) where a number of things are stored:

- **Bootstrap code:** this is used to bring up the router, read the configuration register and load the Cisco IOS software.
- **POST (Power On Self-Test):** Checks the hardware and detects the interfaces/modules that the router has.
- **ROMMON (ROM Monitor):** This is a very simple operating system that we use for troubleshooting and **password recovery**. When the router can't load the IOS you will end up in rommon mode.

Let's take a detailed look at what exactly happens when you power on the router:



1. The POST runs from our ROM and will check if the hardware is ok and what interfaces / modules are present in our router.
2. The bootstrap code will be loaded which is required to locate and load the Cisco IOS software.
3. The bootstrap code will determine the location of Cisco IOS that we need to run. Normally you will have an IOS image on your flash but you can also boot one from a TFTP server. If the bootstrap code can't find any Cisco IOS image it will make sure that it boots ROMMON.
4. The Cisco IOS image will be loaded. It will be copied from our flash or TFTP server to the RAM.
5. Once Cisco IOS has been loaded the bootstrap code will search for the configuration file in the NVRAM.
6. The bootstrap code will copy the startup-config on the NVRAM to the running-config in the RAM. If no startup-config has been found it will start the setup (wizard) or attempt autoconfiguration by using a TFTP server.
7. Run Cisco IOS. We are now done and we can start typing in commands at the CLI.

When the bootstrap code searches for the Cisco IOS image it will use the following sequence:

1. Check the **configuration register**.
2. Use configuration for boot system command.
3. Default to first IOS filename on the flash.

If I can't find the IOS image it will continue like this:

4. Attempt to boot Cisco IOS image from TFTP server.
5. If available, run the boot helper image.
6. Start ROMMON.

The configuration register is used for a number of tasks on the router. We can use it to change the baud rate, tell the router to ignore the startup-config for password recovery and to tell the bootstrap code which Cisco IOS image to boot. By default the configuration register is set to **0x2102** which means that the Cisco IOS image should be retrieved from the flash and that the running-config can be found on the NVRAM.



When you use show version you can see some of the things we just talked about:

```
Router#show version
Cisco IOS Software, 2800 Software (C2800NM-ADVENTERPRISEK9-M), Version
12.4(6)T, RELEASE SOFTWARE (fc1)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2006 by Cisco Systems, Inc.
Compiled Wed 22-Feb-06 22:54 by ccai

ROM: System Bootstrap, Version 12.4(1r) [hqluong 1r], RELEASE SOFTWARE (fc1)

Cisco2811 uptime is 3 minutes
System returned to ROM by power-on
System restarted at 19:52:10 UTC Mon Jan 14 2013
System image file is "flash:c2800nm-adventerprisek9-mz.124-6.t.bin"

This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and
use. Delivery of Cisco cryptographic products does not imply
third-party authority to import, export, distribute or use encryption.
Importers, exporters, distributors and users are responsible for
compliance with U.S. and local country laws. By using this product you
agree to comply with applicable laws and regulations. If you are unable
to comply with U.S. and local laws, return this product immediately.

A summary of U.S. laws governing Cisco cryptographic products may be found
at:
http://www.cisco.com/wwl/export/crypto/tool/stqrg.html

If you require further assistance please contact us by sending email to
export@cisco.com.

Cisco 2811 (revision 53.50) with 247808K/14336K bytes of memory.
Processor board ID FCZ101872TQ
2 FastEthernet interfaces
1 Serial(sync/async) interface
1 terminal line
1 Virtual Private Network (VPN) Module
1 cisco Wireless LAN Controller(s)
DRAM configuration is 64 bitswide with parity enabled.
239K bytes of non-volatile configuration memory.
62720K bytes of ATA CompactFlash (Read/Write)

Configuration register is 0x2102
```

Above you see the bootstrap code, the Cisco IOS image that was loaded from flash, the NVRAM and the configuration register. You can take a look at the flash like this:

```
Router#show flash:

CompactFlash directory:
File Length Name/status
  1  37740020 c2800nm-adventerprisek9-mz.124-6.t.bin
  2   3701   RENEWIFI.txt
```

```
[37743852 bytes used, 26481424 available, 64225276 total]
62720K bytes of ATA CompactFlash (Read/Write)
```

Above you see my flash contents; I only have 1 IOS image here. If I had another one I could use a command to select which one I want to load:

```
Router(config)#boot system flash:c2800nm-name-of-ios.bin
```

The configuration register can be changed as well. One of the reasons you might want to do this is if you boot your router and find yourself staring at this screen:

```
Router>enable
Password:
Password:
Password:
% Bad passwords
```

To **recover the password** we will change the configuration register so that it won't copy the startup-config from the NVRAM to the running-config in the RAM. Here's how to do it:

1. Reboot your router.
2. Press CTRL + BREAK together.
3. You will end up in ROMMON and it will look like this:

```
rommon 1 >
```

Now we can change the configuration register:

```
rommon 1 > confreg 0x2142
```

Now we can reset the router:

```
rommon 1 > reset
```

Now the router will boot just like normal but it will ignore the startup-config from the NVRAM, it will start with the setup. Type "no" for the setup and you will end up at the command prompt:

```
Router>enable
Router#
```

Without a startup-config there is no enable password.

Now we are in privileged mode we will copy the startup-config to the running-config ourselves:

```
Router#copy startup-config running-config
```



*Be careful not to type "copy running-config startup-config" or you will delete your startup-config!*

Now our config is active we can change the password:

```
Router#configure terminal
Router(config)#enable secret NEWPASSWORD
```

That's all there is to it. Don't forget to change the configuration register back to its default settings, or the next time you boot the router it will still ignore the startup-config:

```
Router(config)#config-register 0x2102
```

In the final part of this chapter I'll show you how to copy configuration files and Cisco IOS images from and to our router.

You know about the **copy** command because you probably used "copy running-config startup-config" a couple of times.

You can use it to copy from and to other things however. Take a look at this:

```
Router#show file systems
File Systems:

      Size(b)      Free(b)      Type  Flags  Prefixes
      -          -          -      -      -
      -          -          opaque rw    archive:
      -          -          opaque rw    system:
      129016      128823      nvram  rw    nvram:
      -          -          opaque rw    null:
      -          -          network rw    tftp:
*    8388604      8388604      flash  rw    flash:
      -          -          flash  rw    slot0:
      -          -          flash  rw    slot1:
      -          -          opaque wo    syslog:
      -          -          opaque rw    xmodem:
      -          -          opaque rw    ymodem:
      -          -          network rw    rcp:
      -          -          network rw    pram:
      -          -          network rw    http:
      -          -          network rw    ftp:
      -          -          network rw    scp:
      -          -          network rw    https:
      -          -          opaque ro    cns:
```

The **show file systems** command reveals to us all the file systems this router knows about. We can also use the copy command for HTTP, TFTP or FTP just to name a few.

Let me show you how we can copy the IOS image on our flash to an external TFTP server with the copy command. I will be using the TFTP32 software for this, it's a free TFTP server and you can download it right here:

<http://tftpd32.jounin.net/>

Let's check what the filename is called on our flash:

```
Router#show flash:

CompactFlash directory:
File Length Name/status
  1  37740020 c2800nm-adventerprisek9-mz.124-6.t.bin
  2   3701 RENEWIFI.txt
[37743852 bytes used, 26481424 available, 64225276 total]
62720K bytes of ATA CompactFlash (Read/Write)
```

Let's make a backup of the c2800nm-adventerprisek9-mz.124-6.t.bin file:

```
Router#copy flash: c2800nm-adventerprisek9-mz.124-6.t.bin tftp
Address or name of remote host []? 192.168.1.100
Destination filename [c2800nm-adventerprisek9-mz.124-6.t.bin]?
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
37740020 bytes copied in 24.192 secs (1533 bytes/sec)
```

Use the **copy flash tftp** command to copy the Cisco IOS image to the TFTP server. We can use this to backup the startup-config:

```
Router#copy startup-config tftp
Address or name of remote host []? 192.168.1.100
Destination filename [router-config]?
!!
141 bytes copied in 0.092 secs (1533 bytes/sec)
```

This is how you do it with **copy startup-config tftp**. You can also do it the other way around:

```
Router#copy tftp: startup-config
Address or name of remote host []? 192.168.1.100
Source filename []? startup-config
Destination filename [startup-config]?
Accessing tftp://192.168.1.100/startup-config...
Loading router-config from 192.168.1.100 (via FastEthernet0/0): !
[OK - 141 bytes]
[OK]
141 bytes copied in 0.032 secs (4406 bytes/sec)
```

Use **copy tftp startup-config** to copy something from the TFTP server to your router. This will **overwrite** your startup-config in the NVRAM.

If you want you can also copy something to the running-config:

```
Router#copy tftp: running-config
Address or name of remote host [192.168.1.100]?
Source filename [router-config]?
Destination filename [running-config]?
Accessing tftp://192.168.1.100/router-config...
Loading router-config from 192.168.1.100 (via FastEthernet0/0): !
[OK - 141 bytes]

141 bytes copied in 0.064 secs (2203 bytes/sec)
```

Use **copy tftp running-config** to do this. You need to be aware that this does **not overwrite** the running-config, but the two configurations will **merge together**.

Let me show you an example...let's say the following configuration is stored on your TFTP server:

```
Interface FastEthernet0/0
Ip address 192.168.1.1 255.255.255.0

Interface FastEthernet1/0
Ip address 192.168.2.1 255.255.255.0
```

And this is currently active in your running-config:

```
Interface FastEthernet0/0
Ip address 192.168.1.99 255.255.255.0

Interface FastEthernet1/0
Ip address 192.168.200.1 255.255.255.0

Interface Serial0/0
ip address 172.16.1.1 255.255.255.0
```

Now when I do a **copy tftp running-config** this is what we end up with:

```
Interface FastEthernet0/0
Ip address 192.168.1.1 255.255.255.0

Interface FastEthernet1/0
Ip address 192.168.2.1 255.255.255.0

Interface Serial0/0
ip address 172.16.1.1 255.255.255.0
```

Both configurations are merged together:

- The IP address on the FastEthernet0/0 interface is overwritten by the config from the TFTP server.
- The same thing happens for the FastEthernet1/0 interface.
- Nothing happens with the Serial0/0 interface.

That's all I wanted to show you about network management. I would suggest to play a bit with CDP and copying your config/IOS image while you are doing other labs to become familiar with the commands.

## 26. IOS Licensing

Ever since Cisco created IOS, they shipped it as a single image file. This made installation very easy as you just download an image, copy it to your router or switch and configure your device to boot using the new image. When you want a newer version you'll have to download a new IOS image...there are no patches or bugfixes.

Ever since Cisco was founded there has been an IOS image for each model, but there's a different IOS image for the different versions of each model.

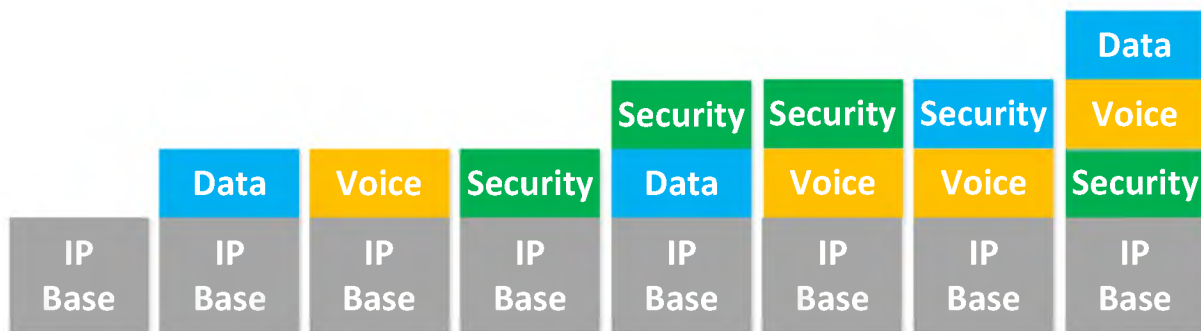
For example, the Cisco 1800 series integrated services router has the following models:

- 1801,1802,1803 and 1805.
- 1811 and 1812.
- 1841
- 1861 and 1861E

You might think that there is one IOS image just for the "1800 series" but this is not the case. There are 4 different IOS image. The 1801, 1802, 1803 and 1805 share a IOS image, so do the 1811 and 1812. For the 1841 there's a separate IOS image and the 1861 and 1861E also share an IOS image.

To make things worse, there are also different IOS images for the different **feature sets**. Depending on the features you require you have to pay for a certain IOS image. For example if you want to run a VPN you might require the "security features" or if you want to use your router for voice over IP you might need the "voice features".

Here's what it looks like:



This is an example of the different IOS images for one router model, let's say the 1861. You can get the IP base image which has some basic features. If you want voice features then you can buy the IOS image with just the voice feature set or one of the images on the right side that also has other feature sets. Of course, the more feature sets the more expensive the IOS image will be...

This is how Cisco ended up with many different IOS images. Different models, feature sets and versions.

Nowadays Cisco ships a **universal image that has all feature sets included**. We still have different IOS images depending on the model and version, but no longer different IOS images with feature sets. Instead of all these different IOS images there's just one:



When you buy a Cisco device nowadays it will include an IOS image that has all feature sets but you will have to unlock them.

Previously it was possible to download just any IOS image from the Cisco website. Once you have a CCO account with download access you could download whatever you want. The problem was that many Cisco customers would just buy a router with the IP base IOS image and download the most advanced IOS image for it. There was no check to see if you had permission to run the IOS image that you downloaded.

Since the introduction of the 1900, 2900 and 3900 routers Cisco introduced the **universal IOS image**. These newer routers called Integrated Services Routers Generation 2 (ISR G2) use these newer IOS images.

When you buy any of these routers it will run the IP Base image by default and if you want extra features you can unlock them with a license key. The feature sets are now called **technology packages**:

- IP Base
- Data
- Unified Communications
- Security

IP Base has the default IOS commands. Data supports features like MPLS, ATM and some others. Unified Communications has voice over IP features and security offers the IOS firewall, intrusion prevention system, IPSEC, etc.

If you buy a router with one of these technology packages then Cisco will activate them for you in the factory. Of course you can always buy and activate them later too.

The technology packages can be activated manually but for customers with large networks Cisco also released an application called **CLM (Cisco License Manager)**. This free tool runs on Windows and Linux and communicates with the Cisco product license registration portal on the Internet to install license keys on your devices.

Let's take a look how we can activate a license for one of the technology packages manually!



The routers that support the new licensing model have a **unique device identifier (UDI)**. This number is a combination of the product ID (PID) and a serial number (SN). You can view this number on your router:

```
Router#show license udi
```

| Device# | PID       | SN          | UDI                   |
|---------|-----------|-------------|-----------------------|
| -----   |           |             |                       |
| *0      | CISCO2951 | FHH1211P025 | CISCO2951:FHH1212P052 |

The **show license udi** command gives us the PID, SN and UDI.

In order to proof that we paid for a license we need something called a **PAK (Product Authorization Key)**. This PAK has a unique number and Cisco uses it to check what license you have bought.

This PAK will be connected to the UDI of the router to create a license key. This can be done by going to the Cisco Product License Registration Portal on the website where you enter the PAK and the UDI. Cisco will check if your PAK and UDI are valid and that you haven't activated the PAK before for another router. If everything is OK, they will e-mail you the license key.

The next step will be to copy the license file to your router; you can use any method you like for this...TFTP, USB flash drive, etc. Once the license file is on your router you need to use the **license install** command to install it. Let's see what licenses are active on this router:

```
Router#show license
```

|         |   |
|---------|---|
| Index 1 | <b>Feature: ipbasek9</b><br><b>Period left: Life time</b><br>License Type: Permanent<br>License State: Active, In Use<br>License Count: Non-Counted<br>License Priority: Medium   |
| Index 2 | <b>Feature: securityk9</b><br><b>Period left: Not Activated</b><br>Period Used: 0 minute 0 second<br>License Type: EvalRightToUse<br>License State: Not in Use, EULA not accepted<br>License Count: Non-Counted<br>License Priority: None |
| Index 3 | <b>Feature: uck9</b><br><b>Period left: Not Activated</b><br>Period Used: 0 minute 0 second<br>License Type: EvalRightToUse<br>License State: Not in Use, EULA not accepted<br>License Count: Non-Counted<br>License Priority: None       |
| Index 4 | <b>Feature: datak9</b><br><b>Period left: Not Activated</b><br>Period Used: 0 minute 0 second<br>License Type: EvalRightToUse<br>License State: Not in Use, EULA not accepted   |

```
License Count: Non-Counted
License Priority: Medium
[OUTPUT OMITTED]
```

First we'll use the **show license** command to verify what licenses are enabled. This router only has the default IP base image and none of the technology packages are enabled right now.

```
Router#show license feature
```

| Feature name   | Enforcement | Evaluation | Subscription | Enabled |
|----------------|-------------|------------|--------------|---------|
| ipbasek9       | no          | no         | no           | yes     |
| securityk9     | yes         | yes        | no           | no      |
| uc             | yes         | yes        | no           | no      |
| data           | yes         | yes        | no           | no      |
| gatekeeper     | yes         | yes        | no           | no      |
| LI             | yes         | no         | no           | no      |
| SSL_VPN        | yes         | yes        | no           | no      |
| ios-ips-update | yes         | yes        | no           | no      |
| SNASw          | yes         | yes        | no           | no      |

You can also use the **show license feature** command. This gives a better overview of the technology packages.

**Show version** will also show you license information:

```
Router#show version
```

```
[OUTPUT OMITTED]
```

```
License Info:
```

```
License UDI:
```

```
-----
Device#   PID                SN
-----
*0        CISCO2951          FHH1222P031
```

```
Technology Package License Information for Module:'c2951'
```

| Technology | Technology-package<br>Current | Technology-package<br>Type | Technology-package<br>Next reboot |
|------------|-------------------------------|----------------------------|-----------------------------------|
| ipbase     | ipbasek9                      | None                       | ipbasek9                          |
| security   | None                          | None                       | None                              |
| uc         | None                          | None                       | None                              |
| data       | None                          | None                       | None                              |

Now let's install the license file that we got from the Cisco website on this router.

I already copied it on the flash of the router:

```
Router# dir flash:
Directory of usbflash1:/
1 -rw-      4096 Aug 10 2013 14:11:00 FTX1628738P_201301131433455187.lic
```

This is the license file that we received from Cisco. It's on the flash of the router. Let's install it:

```
Router#license install usbflash1:FTX1628738P_201301131433455187.lic
Installing...Feature:datak9...Successful:Supported
1/1 licenses were successfully installed
0/1 licenses were existing licenses
0/1 licenses were failed to install
R1#
Feb 11 22:35:20.786: %LICENSE-6-INSTALL: Feature datak9 1.0 was installed in
this
device. UDI=CISCO2901/K9:FTX1628838P; StoreIndex=1:Primary License Storage
Aug 10 21:31:21.038: %IOS_LICENSE_IMAGE_APPLICATION-6-LICENSE_LEVEL: Module
name =
c2900 Next reboot level = datak9 and License = datak9
```

The license for the "data" features was successfully installed. After a reboot this router will support the features. We can confirm this with some show commands:

```
Router#show license
Index 1 Feature: ipbasek9
      Period left: Life time
      License Type: Permanent
      License State: Active, In Use
      License Count: Non-Counted
      License Priority: Medium
Index 2 Feature: securityk9
      Period left: Not Activated
      Period Used: 0 minute 0 second
      License Type: EvalRightToUse
      License State: Not in Use, EULA not accepted
      License Count: Non-Counted
      License Priority: None
Index 3 Feature: uck9
      Period left: Not Activated
      Period Used: 0 minute 0 second
      License Type: EvalRightToUse
      License State: Not in Use, EULA not accepted
      License Count: Non-Counted
      License Priority: None
Index 4 Feature: datak9
      Period left: Life time
      License Type: Permanent
      License State: Active, In Use
      License Count: Non-Counted
      License Priority: Medium
[OUTPUT OMITTED]
```

After a reload you can see that the data features are enabled.

This new method of using licenses to activate features will help Cisco to reduce the number of illegal IOS images out there but it also has a downside. Previously it was easy to test some features by downloading a more advanced IOS image and copying it to your router. Honest customers would pay for the new IOS image to use the new features.

It's still possible to test new features because Cisco has implemented a 60 day trial for the different features. You can enable these features without paying for a PAK. The funny thing however is that after 60 days the feature won't be disabled...the features will remain active and Cisco expects its customers to behave and not take advantage of it.

Using the features without a PAK is called a **right-to-use license**. Let me show you how to activate them:

```
Router(config)# license boot module c3900 technology-package securityk9
```

```
PLEASE READ THE FOLLOWING TERMS CAREFULLY. INSTALLING THE LICENSE OR  
LICENSE KEY PROVIDED FOR ANY CISCO PRODUCT FEATURE OR USING SUCH  
PRODUCT FEATURE CONSTITUTES YOUR FULL ACCEPTANCE OF THE FOLLOWING  
TERMS. YOU MUST NOT PROCEED FURTHER IF YOU ARE NOT WILLING TO BE BOUND  
BY ALL THE TERMS SET FORTH HEREIN.
```

```
You hereby acknowledge and agree that the product feature license  
is terminable and that the product feature enabled by such license  
may be shut down or terminated by Cisco after expiration of the  
applicable term of the license (e.g., 30-day trial period). Cisco  
reserves the right to terminate or shut down any such product feature  
electronically or by any other means available. While alerts or such  
messages may be provided, it is your sole responsibility to monitor  
your terminable usage of any product feature enabled by the license  
and to ensure that your systems and networks are prepared for the shut  
down of the product feature. You acknowledge and agree that Cisco will  
not have any liability whatsoever for any damages, including, but not  
limited to, direct, indirect, special, or consequential damages related  
to any product feature being shutdown or terminated. By clicking the  
"accept" button or typing "yes" you are indicating you have read and  
agree to be bound by all the terms provided herein.
```

```
ACCEPT? [yes/no]: yes
```

You will see the following on your console:

```
%IOS_LICENSE_IMAGE_APPLICATION-6-LICENSE_LEVEL: Module name = c3900
```

```
Next reboot level = securityk9 and License = securityk9
```

```
%LICENSE-6-EULA_ACCEPTED: EULA for feature securityk9 1.0 has been  
accepted. UDI=C3900-SPE150/K9:FHH12250057; StoreIndex=1:Evaluation License  
Storage
```

After accepting the EULA, save your configuration and do a reload. The security features are now enabled and it will be the exact same thing as when you would have bought the PAK. The only difference is that after 60 days you are not allowed to use them anymore.

When you look at the show license command you can see how much time you have left:

```
Router#show license
Index 1 Feature: ipbasek9
      Period left: Life time
      License Type: Permanent
      License State: Active, In Use
      License Count: Non-Counted
      License Priority: Medium
Index 2 Feature: securityk9
      Period left: 8 weeks 4 days
      Period Used: 0 minute 0 second
      License Type: EvalRightToUse
      License State: Not in Use, EULA not accepted
      License Count: Non-Counted
      License Priority: None
Index 3 Feature: uck9
      Period left: Not Activated
      Period Used: 0 minute 0 second
      License Type: EvalRightToUse
      License State: Not in Use, EULA not accepted
      License Count: Non-Counted
      License Priority: None
Index 4 Feature: datak9
      Period left: Life time
      License Type: Permanent
      License State: Active, In Use
      License Count: Non-Counted
      License Priority: Medium
[OUTPUT OMITTED]
```

You can see that the security features are valid for 8 weeks and 4 days (60 days) and that it's an evaluation period.

That's all that I have about IOS licensing for you!

## 27. Final Thoughts

Here we are, you worked your way through all the different chapters that showed you how you can master the CCNA exam. There is only one thing left for you to do and that's *labs, labs and even more labs!* The CCNA exam is very hands-on minded so you need to get experience with Cisco routers and switches to master it! If you want labs just visit <http://gns3vault.com> where I have plenty of CCNA labs for you.

One last word of advice: If you do a Cisco exam you always do the tutorial before you start the exam which takes 15 minutes. These 15 minutes are not taken from your exam time so this is valuable time you can spend creating your own cheat sheet for subnetting questions or anything else you would like to dump from your brain onto paper.

I hope you enjoyed reading my book and truly learned something! If you have any questions or comments how you feel I could improve the book please let me know by sending an e-mail to [info@gns3vault.com](mailto:info@gns3vault.com) or drop a message at my website: <http://gns3vault.com>.

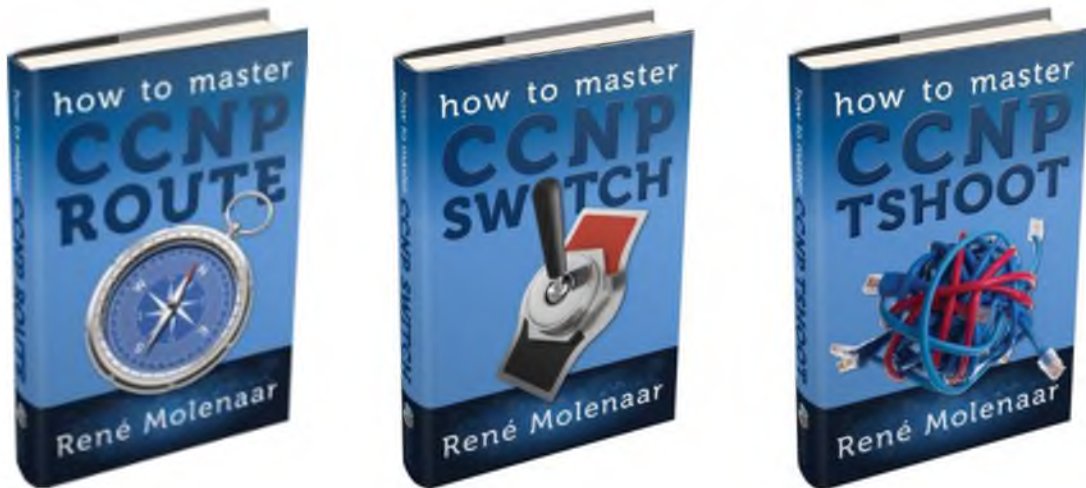
There are two more labs I have for you. They are a mix of different CCNA commands you need to know and I think they are a great way to practice:

<http://gns3vault.com/Combined-Labs/icnd1-assesment-lab.html>  
<http://gns3vault.com/Combined-Labs/icnd2-assessment-lab.html>

I wish you good luck practicing and mastering your CCNA exam!

René Molenaar

PS – If you enjoyed this book and are looking to continue your networking journey by becoming CCNP I'd like to invite you to check out my other books. As a valued customer you will receive a \$5 discount for each book...just click on the picture below to get started!



## Appendix A – How to create mindmaps

A mindmap is a diagram which consists of text, images or relationships between different items. Everything is ordered in a tree-like structure. In the middle of the mindmap you write down your subject. All the topics that have to do with your subject can be written down as a branch of your main subject. Each branch can have multiple branches where the pieces of information are leaves. Mindmaps are great because they show the relationship between different items where notes are just lists...

You can create mindmaps by drawing them yourself or use your computer. I prefer the second method because I can save / print them but also because I'm faster at typing than writing.

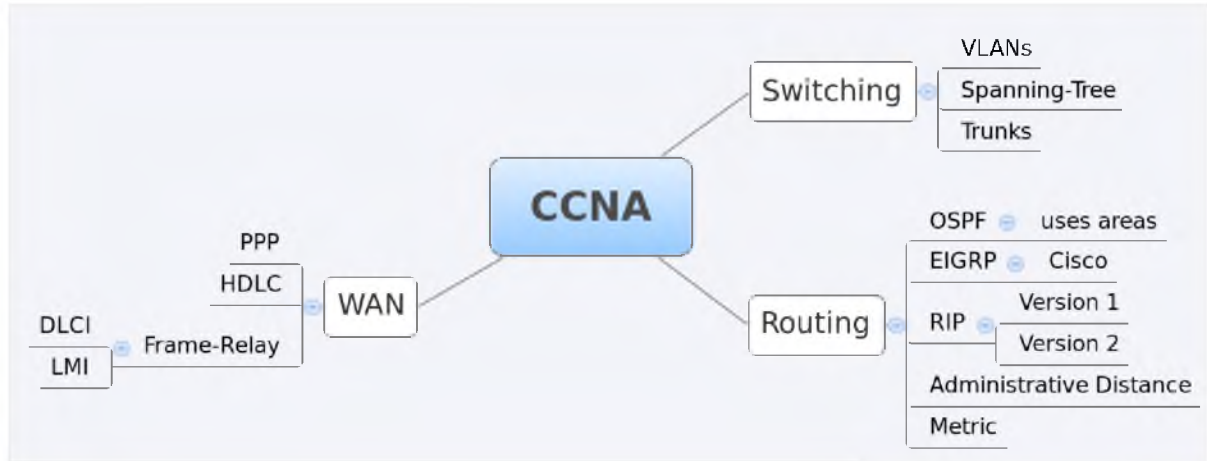
You can download Xmind over here, it's free:

<http://xmind.net>

Once you have installed it and started a new project you can add some items.

You don't have to use the mouse to add new items, just use ENTER to add a new branch or press INSERT to add a new sub-branch.

Here's an example I created for CCNA with some of the items, just to give you an impression:



Just add all the items and build your own mind-map using your own words. Now you have a nice overview with all the stuff you need to remember but also the relationship between items. Give it a shot and see if you like it!