CISCO CERTIFIED NETWORK ASSOCIATE

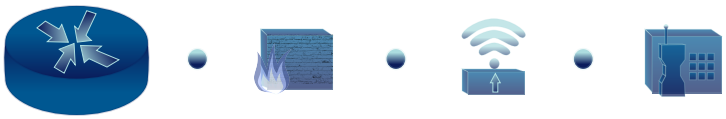
COMPUTER SCIENCE PORTFOLIO

GABRIEL ROSAS

CONTENTS

OSPFv2

Configuring Single Area OSPF in Packet Tracer



Purpose

The purpose of this lab was to set up single area OSPFv2 across four Cisco catalyst 4321 routers. Since OSPF is a widely used *Interior Gateway Protocol* (IGP), knowledge of how OSPF functions, how to configure OSPF, even specifics such as pathfinding algorithms are impactful in the field of computer science. I tried to focus on building good habits, such as creating *IP schemes*, designing *topologies*, and debugging with *show* commands**,** which can be critical in troubleshooting and proofing.

Background Information

Routing

Routing is a significant process in networking as it allows hosts on different IP networks to connect to each other. *Open Shortest Path First* (OSPF) is a routing protocol simplifying the process of creating routes by using algorithms to figure out the directions automatically. OSPF excels in interior networks, which are smaller in scale, but would crash in large networks with hundreds of routes. In networking, routes are ultimately just *directions* for packets.

There are two options when dealing with traffic on a network; you can configure *static routes*, or you can set up a *routing protocol*. I like to think of static routes as absolute directions drawn onto a map, set in stone and unchangeable. The map can't be altered unless it is manually redrawn. If you were to follow the map, you might find some of the routes to be outdated.

It would be nice if routes were adaptable, if they could update based on the fastest paths available. This is the difference between *static routing* and *routing protocols*. Routing protocols update their routing directions automatically based on information sent from neighbors. This is the magic of routing protocols: automatic updates and directions – like google maps – for packets. Routes are stored in a database on the router, known as a *routing table*.

Routing tables

Like a signpost at a fork in the road, routers contain directions for different destinations. These directions are stored in RAM memory on the router, which means that they are temporary; RAM memory can be accessed much faster than hard drives or SSDs but is not saved after the device shuts down. Let's look at an example of a packet arriving at a router.

A packet arrives at a router. This router has three interfaces: north, south, and east. The packet arrived on the east interface, so it either must turn north or south, assuming one of these paths lead to the destination. Luckily, there are directions in the router: *10.0.0.0/24* out interface *north*; *172.16.0.0/24* out interface *south*. The packet has a destination address of *10.0.0.3*, which matches up with the *north* interface. The router sends the packet out the north interface. Routes

are either generated statically, by the admin, or automatically by routing protocols such as OSPF, BGP, etc.

Here is an example of a routing table:

```
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 11 subnets, 2 masks
O IA    10.10.10.0/30 [110/128] via 10.10.10.5, 01:03:27, Serial0/1/1
C       10.10.10.4/30 is directly connected, Serial0/1/1
L       10.10.10.6/32 is directly connected, Serial0/1/1
C       10.10.10.8/30 is directly connected, Serial0/1/0
L       10.10.10.9/32 is directly connected, Serial0/1/0
O IA    10.10.10.12/30 [110/128] via 10.10.10.10, 01:03:27, Serial0/1/0
C       10.10.10.16/30 is directly connected, Serial0/2/0
L       10.10.10.17/32 is directly connected, Serial0/2/0
O IA    10.10.10.20/30 [110/128] via 10.10.10.18, 01:03:27, Serial0/2/0
O IA    10.10.10.24/30 [110/192] via 10.10.10.18, 01:03:27, Serial0/2/0
O E2    10.10.10.28/30 [110/100] via 10.10.10.18, 01:03:27, Serial0/2/0
```

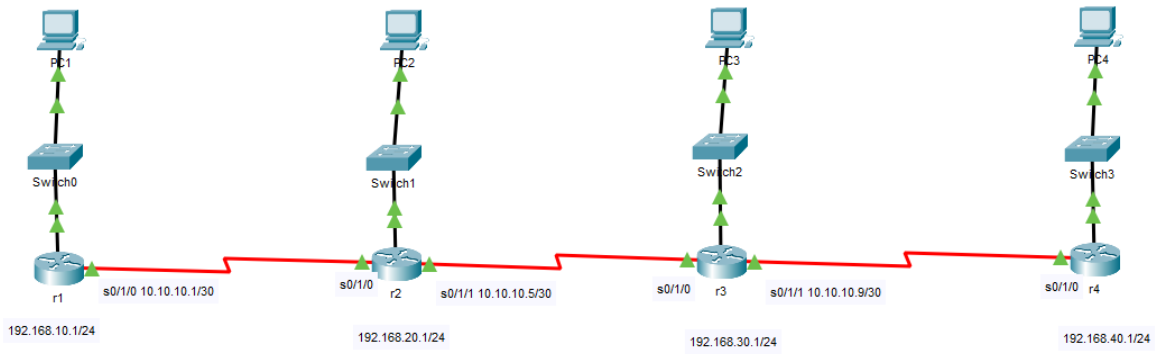
Ignoring the letters on the left (the origin of the route), we can see a range of addresses and the corresponding interface leading towards them. For example, *10.10.10.0/30* addresses direct out the *Serial0/1/1* interface. "Via *ip*", is also commonly seen as a direction, indicating that a packet should be sent to the specified neighboring router. Sometimes there is a combination of directions: both *interface* and *neighboring ips*.

OSPF

Since we've defined routing and routing tables, I can go into more detail on how OSPF functions. Each router is like a junction for packets; packets usually have multiple roads they can turn down to reach further junctions, ultimately ending at their destination. Every router running OSPF will communicate with neighbor OSPF routers to relay statuses and updates about new routes and preferred paths. By sharing information to neighbor OSPF routers, information can spread through an OSPF network regardless of hop counts between routers. The packets OSPF broadcasts to relay information are known as *Link-State Advertisements* (LSAs). To see more on LSAs, check out *LSA Background Information*.

OSPF configured routers rely on *cost* to commute the shortest path through a network. While you can set the cost manually, OSPF will automatically determine the cost value per interface based on a *reference bandwidth* – usually the bandwidth of the fastest interface in your network – and *interface bandwidth***,** the bandwidth of the interface being assessed.

Network Diagram



IP Scheme

PC	IPv4 address
1	192.168.10.10/24
2	192.168.20.20/24
3	192.168.30.30/24
4	192.168.40.40/24

Router	Interface	IPv4 address
1	G0/0/0:	192.168.10.1/24
	S0/1/0:	10.10.10.1/30
2	G0/0/0:	192.168.20.1/24
	S0/1/0:	10.10.10.2/30
	S0/1/1:	10.10.10.5/30
3	G0/0/0:	192.168.30.1/24
	S0/1/0:	10.10.10.6/30
	S0/1/1:	10.10.10.9/30
4	G0/0/0:	192.168.40.1/24
	S0/1/0:	10.10.10.10/30

Summary

In this lab, I created an OSPF network routing IPv4 traffic across four routers. I began by constructing an IP scheme: for devices to be able to communicate, they need IP addresses. Then, I designed a topology tailored to the IP scheme. Once all the IPv4 addresses were allocated correctly on my network, I configured an OSPF router and network statements on each router.

Before OSPF was configured, the pings from the different end PCs couldn't reach each other. After OSPF was set up, routes were created, and the pings could now extend outside of their routers.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

Router(config)# **interface [*interface*] [*id*]**

- Enables configuration on a specific interface

// OSPF

Router(config)# **router ospf [*process id*]**

- Enables the OSPF routing protocol and enters OSPF router configuration mode

Generally, OSPF process ids should be the same, though OSPF should still form adjacencies with different process ids. Each OSPF process retains a different routing table; depending on the configuration, process ID could determine what routes are redistributed. A router can run multiple OSPF processes but will contain a separate OSPF database per process.

Router(config-router)# **router-id [*router id*]**

- Uniquely determines an OSPF router within a domain

Router ids are automatically determined by the highest loopback interface if they are not manually defined. Router ids can play a part in DR/BDR elections.

Router(config-router)# **network [*network address*] [*wildcard mask*] area [*area number*]**

- Advertises the specified subnet to neighbor routers

The defined subnet should match an interface on the router. Routers in an area share a complete topological database and have route summaries of external areas.

// Show Commands

Router# **show [*ipv4/ipv6*] ospf database**

- Displays the routing database

Router# **show [*ipv4/ipv6*] ospf neighbor**

- Displays information about adjacent routers configured with OSPF

Router# **show [*ipv4/ipv6*] ospf interface**

- Displays information about each interface configured with OSPF

Configurations

Router 1

```
r1#show running-config
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname r1
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 ip address 192.168.10.1 255.255.255.0
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:1::A/64
 ipv6 ospf 10 area 0
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.1 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 1::1/64
 ipv6 ospf 10 area 0
interface Serial0/1/1
 no ip address
```

```
 clock rate 2000000
 shutdow
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 1.1.1.1
 log-adjacency-changes
 network 192.168.10.0 0.0.0.255 area 0
 network 10.10.10.0 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
no cdp run
line con 0
line aux 0
line vty 0 4
 login
end
```

r1#show ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C       10.10.10.0/30 is directly connected, Serial0/1/0
L       10.10.10.1/32 is directly connected, Serial0/1/0
O       10.10.10.4/30 [110/128] via 10.10.10.2, 00:11:10, Serial0/1/0
O       10.10.10.8/30 [110/192] via 10.10.10.2, 00:11:10, Serial0/1/0
     192.168.10.0/24 is variably subnetted, 2 subnets, 2 masks
C       192.168.10.0/24 is directly connected, GigabitEthernet0/0/0
L       192.168.10.1/32 is directly connected, GigabitEthernet0/0/0
O    192.168.20.0/24 [110/65] via 10.10.10.2, 00:11:10, Serial0/1/0
O    192.168.30.0/24 [110/129] via 10.10.10.2, 00:11:10, Serial0/1/0
O    192.168.40.0/24 [110/193] via 10.10.10.2, 00:11:10, Serial0/1/0
```

Router 2

r2#show running-config

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname r2
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 ip address 192.168.20.1 255.255.255.0
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:2::A/64
 ipv6 ospf 10 area 0
interface GigabitEthernet0/0/1
 no ip address
```

```
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.2 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 1::2/64
 ipv6 ospf 10 area 0
 clock rate 2000000
interface Serial0/1/1
 ip address 10.10.10.5 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 2::1/64
 ipv6 ospf 10 area 0
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 2.2.2.2
 log-adjacency-changes
 network 192.168.20.0 0.0.0.255 area 0
 network 10.10.10.0 0.0.0.3 area 0
 network 10.10.10.4 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
no cdp run
line con 0
line aux 0
line vty 0 4
 login
end
```

r2#show ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
C       10.10.10.0/30 is directly connected, Serial0/1/0
L       10.10.10.2/32 is directly connected, Serial0/1/0
C       10.10.10.4/30 is directly connected, Serial0/1/1
L       10.10.10.5/32 is directly connected, Serial0/1/1
O       10.10.10.8/30 [110/128] via 10.10.10.6, 00:13:45, Serial0/1/1
O    192.168.10.0/24 [110/65] via 10.10.10.1, 00:13:45, Serial0/1/0
     192.168.20.0/24 is variably subnetted, 2 subnets, 2 masks
C       192.168.20.0/24 is directly connected, GigabitEthernet0/0/0
L       192.168.20.1/32 is directly connected, GigabitEthernet0/0/0
O    192.168.30.0/24 [110/65] via 10.10.10.6, 00:13:45, Serial0/1/1
O    192.168.40.0/24 [110/129] via 10.10.10.6, 00:13:45, Serial0/1/1
```

Router 3

r3#show running-config

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
```

```
hostname r3
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 ip address 192.168.30.1 255.255.255.0
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:3::A/64
 ipv6 ospf 10 area 0
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.6 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 2::2/64
 ipv6 ospf 10 area 0
 clock rate 2000000
interface Serial0/1/1
 ip address 10.10.10.9 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 3::1/64
 ipv6 ospf 10 area 0
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 3.3.3.3
 log-adjacency-changes
 network 192.168.30.0 0.0.0.255 area 0
 network 10.10.10.4 0.0.0.3 area 0
 network 10.10.10.8 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
no cdp run
line con 0
line aux 0
line vty 0 4
 login
end
```

r3#show ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
O       10.10.10.0/30 [110/128] via 10.10.10.5, 00:15:13, Serial0/1/0
C       10.10.10.4/30 is directly connected, Serial0/1/0
L       10.10.10.6/32 is directly connected, Serial0/1/0
C       10.10.10.8/30 is directly connected, Serial0/1/1
L       10.10.10.9/32 is directly connected, Serial0/1/1
```

```
O    192.168.10.0/24 [110/129] via 10.10.10.5, 00:15:03, Serial0/1/0
O    192.168.20.0/24 [110/65] via 10.10.10.5, 00:15:13, Serial0/1/0
     192.168.30.0/24 is variably subnetted, 2 subnets, 2 masks
C       192.168.30.0/24 is directly connected, GigabitEthernet0/0/0
L       192.168.30.1/32 is directly connected, GigabitEthernet0/0/0
O    192.168.40.0/24 [110/65] via 10.10.10.10, 00:15:13, Serial0/1/1
```

Router 4

```
r4#show running-config
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname r4
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 ip address 192.168.40.1 255.255.255.0
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:4::A/64
 ipv6 ospf 10 area 0
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.10 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 3::2/64
 ipv6 ospf 10 area 0
 clock rate 2000000
interface Serial0/1/1
 no ip address
 clock rate 2000000
 shutdown
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 4.4.4.4
 log-adjacency-changes
 network 192.168.40.0 0.0.0.255 area 0
 network 10.10.10.8 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
no cdp run
line con 0
line aux 0
line vty 0 4
 login
end
```

```
r4#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
```

```
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
O       10.10.10.0/30 [110/192] via 10.10.10.9, 00:16:28, Serial0/1/0
O       10.10.10.4/30 [110/128] via 10.10.10.9, 00:16:38, Serial0/1/0
C       10.10.10.8/30 is directly connected, Serial0/1/0
L       10.10.10.10/32 is directly connected, Serial0/1/0
O    192.168.10.0/24 [110/193] via 10.10.10.9, 00:16:28, Serial0/1/0
O    192.168.20.0/24 [110/129] via 10.10.10.9, 00:16:28, Serial0/1/0
O    192.168.30.0/24 [110/65] via 10.10.10.9, 00:16:38, Serial0/1/0
     192.168.40.0/24 is variably subnetted, 2 subnets, 2 masks
C       192.168.40.0/24 is directly connected, GigabitEthernet0/0/0
L       192.168.40.1/32 is directly connected, GigabitEthernet0/0/0
```

Ping Statistics – Testing Connectivity

PC 1

```
Packet Tracer PC Command Line 1.0
C:\>ping 192.168.10.10
Pinging 192.168.10.10 with 32 bytes of data:
Reply from 192.168.10.10: bytes=32 time=7ms TTL=128
Reply from 192.168.10.10: bytes=32 time=3ms TTL=128
Reply from 192.168.10.10: bytes=32 time=4ms TTL=128
Reply from 192.168.10.10: bytes=32 time=9ms TTL=128
Ping statistics for 192.168.10.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 3ms, Maximum = 9ms, Average = 5ms
C:\>ping 192.168.20.20
Pinging 192.168.20.20 with 32 bytes of data:
Request timed out.
Reply from 192.168.20.20: bytes=32 time=3ms TTL=126
Reply from 192.168.20.20: bytes=32 time=4ms TTL=126
Reply from 192.168.20.20: bytes=32 time=3ms TTL=126
Ping statistics for 192.168.20.20:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 3ms, Maximum = 4ms, Average = 3ms
C:\>ping 192.168.30.30
Pinging 192.168.30.30 with 32 bytes of data:
Request timed out.
Reply from 192.168.30.30: bytes=32 time=10ms TTL=125
Reply from 192.168.30.30: bytes=32 time=2ms TTL=125
Reply from 192.168.30.30: bytes=32 time=4ms TTL=125
Ping statistics for 192.168.30.30:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 10ms, Average = 5ms
C:\>ping 192.168.40.40
Pinging 192.168.40.40 with 32 bytes of data:
Request timed out.
Reply from 192.168.40.40: bytes=32 time=18ms TTL=124
Reply from 192.168.40.40: bytes=32 time=8ms TTL=124
Reply from 192.168.40.40: bytes=32 time=3ms TTL=124
Ping statistics for 192.168.40.40:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
```

```
Minimum = 3ms, Maximum = 18ms, Average = 9ms
```

Problems

The first obstacle I overcame was creating my own IP scheme. Since this was my first ever lab, it was up to me to determine everything up to the minor details. Constructing these foundations significantly helped later in the lab as I could refer to devices and IP addresses with ease. The IP scheme can be found under the network diagram portion of this lab.

After a couple of months without configuring Cisco devices, I found myself forgetting a lot of commands I had learnt my previous year. We had not learnt about routing protocols, so I had to research what they do and how to set OSPF up. I found myself stumbling across the *network* command on multiple example configurations online. Network statements are key for advertising and enabling OSPF on a network, written in the form *network [network address] [wildcard mask] area [area number]*. Once network statements are configured on adjacent routers, OSPF adjacencies can form, and I was able to ping any PC on the network.

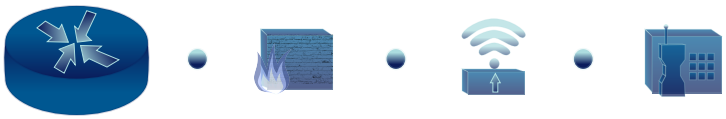
Personally, I think documenting and creating IP schemes will be an important takeaway of this lab for me, so I don't make mistakes early on that will be detrimental shortly after.

Conclusion

In this lab, I set up OSPFv2 in Cisco packet tracer. Though it was just a simulation, I think I could take this knowledge and configure OSPFv2 on a physical catalyst 4321 router. Configuring OSPF went quite smoothly for me, which can probably be credited to extensive documentation. Overall, this was a nice refresher back into the world of networking, and I will be documenting all topologies from this point forth.

OSPFv3

Configuring Multi Area OSPFv3 in Packet Tracer



Purpose

The purpose of this lab was to set up a multi-area OSPFv3 network across five Cisco catalyst 4321 routers. Three of the routers are in area 0, and two are in area 1. We had to figure out how to configure OSPFv3 with our general knowledge on OSPFv2, allowing IPv6 connectivity across all the networks.

Background Information

Routing

Routing is a significant process in networking as it allows hosts on different IP networks to connect to each other. *Open Shortest Path First* (OSPF) is a routing protocol simplifying the process of creating routes by using algorithms to figure out the directions automatically. OSPF excels in interior networks, which are smaller in scale, but would crash in large networks with hundreds of routes. In networking, routes are ultimately just *directions* for packets.

There are two options when dealing with traffic on a network; you can configure *static routes*, or you can set up a *routing protocol*. I like to think of static routes as absolute directions drawn onto a map, set in stone and unchangeable. The map can't be altered unless it is manually redrawn. If you were to follow the map, you might find some of the routes to be outdated.

It would be nice if routes were adaptable, if they could update based on the fastest paths available. This is the difference between *static routing* and *routing protocols*. Routing protocols update their routing directions automatically based on information sent from neighbors. This is the magic of routing protocols: automatic updates and directions – like google maps – for packets. Routes are stored in a database on the router, known as a *routing table*.

Routing tables

Like a signpost at a fork in the road, routers contain directions for different destinations. These directions are stored in RAM memory on the router, which means that they are temporary; RAM memory can be accessed much faster than hard drives or SSDs but is not saved after the device shuts down. Let's look at an example of a packet arriving at a router.

A packet arrives at a router. This router has three interfaces: north, south, and east. The packet arrived on the east interface, so it either must turn north or south, assuming one of these paths lead to the destination. Luckily, there are directions in the router: *10.0.0.0/24* out interface *north*; *172.16.0.0/24* out interface *south*. The packet has a destination address of *10.0.0.3*, which matches up with the *north* interface. The router sends the packet out the north interface. Routes are either generated statically, by the admin, or automatically by routing protocols such as OSPF, BGP, etc.

Here is an example of a routing table:

```
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 11 subnets, 2 masks
```

```
O IA    10.10.10.0/30 [110/128] via 10.10.10.5, 01:03:27, Serial0/1/1
C       10.10.10.4/30 is directly connected, Serial0/1/1
L       10.10.10.6/32 is directly connected, Serial0/1/1
C       10.10.10.8/30 is directly connected, Serial0/1/0
L       10.10.10.9/32 is directly connected, Serial0/1/0
O IA    10.10.10.12/30 [110/128] via 10.10.10.10, 01:03:27, Serial0/1/0
C       10.10.10.16/30 is directly connected, Serial0/2/0
L       10.10.10.17/32 is directly connected, Serial0/2/0
O IA    10.10.10.20/30 [110/128] via 10.10.10.18, 01:03:27, Serial0/2/0
O IA    10.10.10.24/30 [110/192] via 10.10.10.18, 01:03:27, Serial0/2/0
O E2    10.10.10.28/30 [110/100] via 10.10.10.18, 01:03:27, Serial0/2/0
```

Ignoring the letters on the left (the origin of the route), we can see a range of addresses and the corresponding interface leading towards them. For example, *10.10.10.0/30* addresses direct out the *Serial0/1/1* interface. "Via *ip*", is also commonly seen as a direction, indicating that a packet should be sent to the specified neighboring router. Sometimes there is a combination of directions: both *interface* and *neighboring ips*.

OSPF

Since we've defined routing and routing tables, I can go into more detail on how OSPF functions. Each router is like a junction for packets; packets usually have multiple roads they can turn down to reach further junctions, ultimately ending at their destination. Every router running OSPF will communicate with neighbor OSPF routers to relay statuses and updates about new routes and preferred paths. By sharing information to neighbor OSPF routers, information can spread through an OSPF network regardless of hop counts between routers. The packets OSPF broadcasts to relay information are known as *Link-State Advertisements* (LSAs). To see more on LSAs, check out *LSA Background Information*.

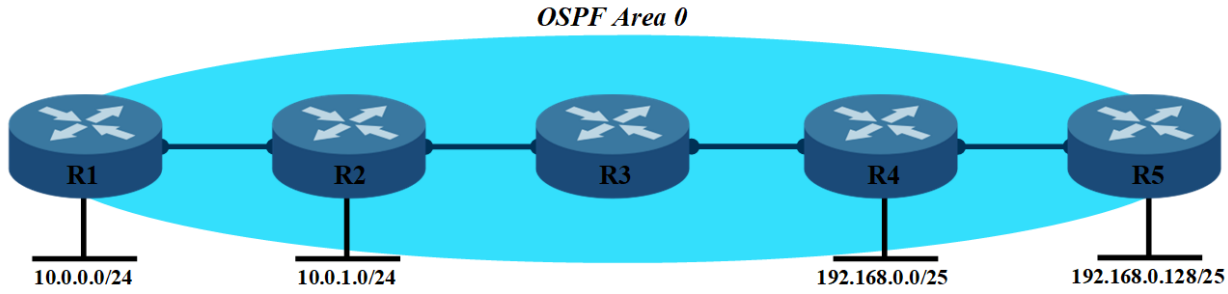
OSPF configured routers rely on *cost* to commute the shortest path through a network. While you can set the cost manually, OSPF will automatically determine the cost value per interface based on a *reference bandwidth* – usually the bandwidth of the fastest interface in your network – and *interface bandwidth***,** the bandwidth of the interface being assessed.

There are two OSPF protocols that can be configured on a router: OSPFv2 and OSPFv3, the main difference being *OSPFv2* routes *IPv4* and *OSPFv3* routes *IPv6*. OSPFv3 has nine *Link-State Advertisements*. LSAs are used to communicate different states and information of an OSPF router, such as a neighbor's local topology, to build the *routing table*. Although there are other routing protocols such as RIP or EIGRP, OSPF is massively adopted in large enterprise networks because of its many benefits: route redundancy, the ability to run on most routers, classless routing, and loop-free topologies.

Multi-area OSPF

OSPF routers communicate to each other using LSA packets, but this communication comes at a cost: bandwidth. When OSPF runs across a large network, LSA packets consume more bandwidth, as there are more routers that send updates. If the network has low-bandwidth interfaces, LSA traffic could hinder the performance. But what if we could limit the amount of LSA traffic on a network?

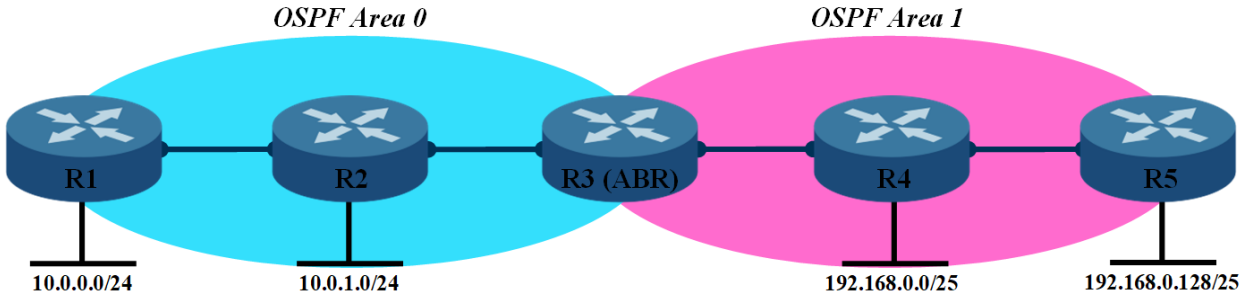
Multi-area OSPF is the process of dividing routers into multiple groups, known as *areas*, to reduce the size of LSA packets that need to be sent. Routes need to be specific and abundant for each network advertised within a local area. However, routers in a local area only need a broad definition of networks in external areas. Therefore, LSA packets across areas transmit summarizations by compressing multiple lines of routes into a single subnet. Let's compare a single-area OSPF network to a multi-area OSPF network.



In this topology, each router needs a specific route for every network in the area. LSA packets would be abundant in this area for each network OSPF is advertising. There would be no summarizations. A routing table on R2 may look something like the following:

- *10.0.0.0/24 out interface GigabitEthernet0*
- *192.168.0.0/25 out interface GigabitEthernet1*
- *192.168.0.128/25 out interface GigabitEthernet1*

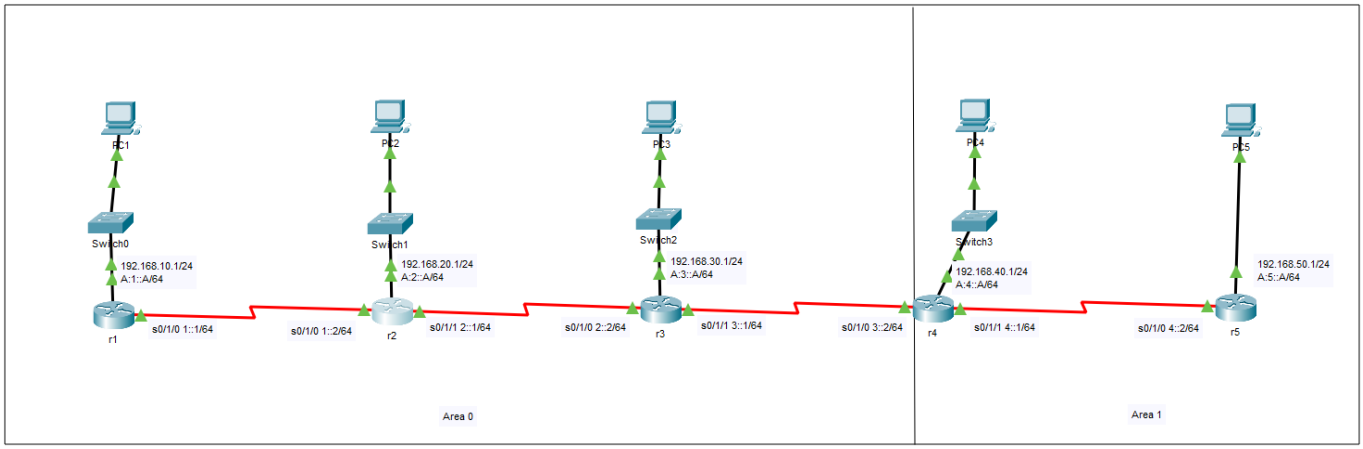
Now let's divide this network into two areas.



Now that there are multiple areas, we can summarize networks for each area. Instead of routers four and five having routes to the *10.0.0.0/24* and *10.0.1.0/24* networks, they can share a singular, summarized route, *10.0.0.0/23*, that points towards the ASBR. When a packet enters the destination area, more precise routes will direct it to the destined network. An LSA packet to *area 0* from *area 1* might be distributing the following network, *192.168.0.0/24*, only consisting of one prefix. Compared to LSAs containing all the specific routes, *route summarization* helps reduce LSA packet sizes. Now, the routing table on R2 may look something like the following:

- *10.0.0.0/24 out interface GigabitEthernet0*
- *192.168.0.0/24 out interface GigabitEthernet1*

Network Diagram



PC	IPv4 address	IPv6 address
1	192.168.10.10/24	A:1::1/64
2	192.168.20.20/24	A:2::1/64
3	192.168.30.30/24	A:3::1/64
4	192.168.40.40/24	A:4::1/64
5	192.168.50.50/24	A:5::1/64

Router	Interface	IPv4 Address	IPv6 Global Address	IPv6 Link-Local Address
1	G0/0/0	192.168.10.1/24	A:1::A/64	FE80::1
	S0/1/0	10.10.10.1/30	1::1	FE80::1
2	G0/0/0	192.168.20.1/24	A:2::A/64	FE80::1
	S0/1/0	10.10.10.2/30	1::2	FE80::2
	S0/1/1	10.10.10.5/30	2::1	FE80::1
3	G0/0/0	192.168.30.1/24	A:3::A/64	FE80::1
	S0/1/0	10.10.10.6/30	2::2	FE80::2
	S0/1/1	10.10.10.9/30	3::1	FE80::1
4	G0/0/0	192.168.40.1/24	A:4::A/64	FE80::1
	S0/1/0	10.10.10.10/30	3::2	FE80::2
	S0/1/1	10.10.10.13/30	4::1	FE80::1
5	G0/0/0	192.168.50.1/24	A:5::A/64	FE80::1
	S0/1/0	10.10.10.14/30	4::2	FE80::2

Summary

In this lab, I set up multi-area OSPF networks routing both IPv4 and IPv6 traffic across five Cisco catalyst 4321 routers. I began by constructing a topology and then building an IP scheme. After allocating all my IPv4 and IPv6 addresses to their corresponding PC's and routers, I tested connectivity. Pings from PCs could reach their default gateways, but nothing past that. Time to form some routes with OSPF. I configured each active interface on every router, advertising OSPF networks in the correct areas. OSPFv3 is similar in configuration to OSPFv2, except OSPFv3 is advertised in *interface configuration mode* and OSPFv2 is advertised using network statements in *router configuration mode*. Once OSPF adjacencies had formed, routes were automatically distributed, and pings could reach anywhere in the network.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

Router(config)# **interface [*interface*] [*id*]**

- Enables configuration on a specific interface

// OSPF

Router(config)# **router ospf [*process id*]**

- Enables the OSPF routing protocol and enters OSPF router configuration mode

Generally, OSPF process ids should be the same, though OSPF should still form adjacencies with different process ids. Each OSPF process retains a different routing table; depending on the configuration, process ID could determine what routes are redistributed. A router can run multiple OSPF processes but will contain a separate OSPF database per process.

Router(config-router)# **router-id [*router id*]**

- Uniquely determines an OSPF router within a domain

Router ids are automatically determined by the highest loopback interface if they are not manually defined. Router ids can play a part in DR/BDR elections.

Router(config-router)# **network [*network address*] [*wildcard mask*] area [*area number*]**

- Advertises the specified subnet to neighbor routers

// OSPFv3

Router(config)# **ipv6 router ospf [*process id*]**

- Enables the OSPF routing protocol and enters OSPF router configuration mode

Generally, OSPF process ids should be the same, though OSPF should still form adjacencies with different process ids. Each OSPF process retains a different routing table; depending on the configuration, process ID could determine what routes are redistributed. A router can run multiple OSPF processes but will contain a separate OSPF database per process.

Router(config-rtr)# **router-id [*router id*]**

- Uniquely determines an OSPF router within a domain

Router ids are automatically determined by the highest loopback interface if they are not manually defined. Router ids can play a part in DR/BDR elections.

Router(config-if)# **ipv6 ospf [*process id*] area [*number*]**

- Advertises an IPv6 OSPFv3 network in a local interface

Written in interface configuration mode. Generally, OSPF process ids should be the same, though OSPF should still form adjacencies with different process ids.

// Show Commands

Router# **show [*ipv4/ipv6*] ospf database**

- Displays the routing database

Router# **show [*ipv4/ipv6*] ospf neighbor**

- Displays information about adjacent routers configured with OSPF

Router# **show [*ipv4/ipv6*] ospf interface**

- Displays information about each interface configured with OSPF

Configurations

Router 1

```
r1#show run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname r1
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 ip address 192.168.10.1 255.255.255.0
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:1::A/64
 ipv6 ospf 10 area 0
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.1 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 1::1/64
 ipv6 ospf 10 area 0
interface Serial0/1/1
 no ip address
 clock rate 2000000
 shutdown
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 1.1.1.1
 log-adjacency-changes
 network 192.168.10.0 0.0.0.255 area 0
 network 10.10.10.0 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
no cdp run
line con 0
line aux 0
line vty 0 4
 login
end
```

```
r1#show ipv6 route
IPv6 Routing Table - 12 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   1::/64 [0/0]
     via Serial0/1/0, directly connected
L   1::1/128 [0/0]
     via Serial0/1/0, receive
O   2::/64 [110/128]
     via FE80::2, Serial0/1/0
O   3::/64 [110/192]
     via FE80::2, Serial0/1/0
OI  4::/64 [110/256]
     via FE80::2, Serial0/1/0
C   A:1::/64 [0/0]
     via GigabitEthernet0/0/0, directly connected
L   A:1::A/128 [0/0]
     via GigabitEthernet0/0/0, receive
O   A:2::/64 [110/65]
     via FE80::2, Serial0/1/0
O   A:3::/64 [110/129]
     via FE80::2, Serial0/1/0
OI  A:4::/64 [110/193]
     via FE80::2, Serial0/1/0
OI  A:5::/64 [110/257]
     via FE80::2, Serial0/1/0
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 2

```
r2#show run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname r2
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 ip address 192.168.20.1 255.255.255.0
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:2::A/64
 ipv6 ospf 10 area 0
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.2 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 1::2/64
 ipv6 ospf 10 area 0
 clock rate 2000000
```

```
interface Serial0/1/1
 ip address 10.10.10.5 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 2::1/64
 ipv6 ospf 10 area 0
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 2.2.2.2
 log-adjacency-changes
 network 192.168.20.0 0.0.0.255 area 0
 network 10.10.10.0 0.0.0.3 area 0
 network 10.10.10.4 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
no cdp run
line con 0
line aux 0
line vty 0 4
 login
end
```

r2#show ipv6 route

```
IPv6 Routing Table - 13 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   1::/64 [0/0]
     via Serial0/1/0, directly connected
L   1::2/128 [0/0]
     via Serial0/1/0, receive
C   2::/64 [0/0]
     via Serial0/1/1, directly connected
L   2::1/128 [0/0]
     via Serial0/1/1, receive
O   3::/64 [110/128]
     via FE80::2, Serial0/1/1
OI  4::/64 [110/192]
     via FE80::2, Serial0/1/1
O   A:1::/64 [110/65]
     via FE80::1, Serial0/1/0
C   A:2::/64 [0/0]
     via GigabitEthernet0/0/0, directly connected
L   A:2::A/128 [0/0]
     via GigabitEthernet0/0/0, receive
O   A:3::/64 [110/65]
     via FE80::2, Serial0/1/1
OI  A:4::/64 [110/129]
     via FE80::2, Serial0/1/1
OI  A:5::/64 [110/193]
     via FE80::2, Serial0/1/1
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 3

```
r3#show run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname r3
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 ip address 192.168.30.1 255.255.255.0
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:3::A/64
 ipv6 ospf 10 area 0
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.6 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 2::2/64
 ipv6 ospf 10 area 0
 clock rate 2000000
interface Serial0/1/1
 ip address 10.10.10.9 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 3::1/64
 ipv6 ospf 10 area 0
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 3.3.3.3
 log-adjacency-changes
 network 192.168.30.0 0.0.0.255 area 0
 network 10.10.10.4 0.0.0.3 area 0
 network 10.10.10.8 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
no cdp run
line con 0
line aux 0
line vty 0 4
 login
end

r3#show ipv6 route
IPv6 Routing Table - 13 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   1::/64 [110/128]
     via FE80::1, Serial0/1/0
C   2::/64 [0/0]
```

```
     via Serial0/1/0, directly connected
L    2::2/128 [0/0]
     via Serial0/1/0, receive
C    3::/64 [0/0]
     via Serial0/1/1, directly connected
L    3::1/128 [0/0]
     via Serial0/1/1, receive
OI   4::/64 [110/128]
     via FE80::2, Serial0/1/1
O    A:1::/64 [110/129]
     via FE80::1, Serial0/1/0
O    A:2::/64 [110/65]
     via FE80::1, Serial0/1/0
C    A:3::/64 [0/0]
     via GigabitEthernet0/0/0, directly connected
L    A:3::A/128 [0/0]
     via GigabitEthernet0/0/0, receive
OI   A:4::/64 [110/65]
     via FE80::2, Serial0/1/1
OI   A:5::/64 [110/129]
     via FE80::2, Serial0/1/1
L    FF00::/8 [0/0]
     via Null0, receive
```

Router 4

```
r4#show run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname r4
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 ip address 192.168.40.1 255.255.255.0
 ip ospf 10 area 1
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:4::A/64
 ipv6 ospf 10 area 1
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.10 255.255.255.252
 ip ospf 10 area 0
 ipv6 address FE80::2 link-local
 ipv6 address 3::2/64
 ipv6 ospf 10 area 0
 clock rate 2000000
interface Serial0/1/1
 ip address 10.10.10.13 255.255.255.252
 ip ospf 10 area 1
 ipv6 address FE80::1 link-local
 ipv6 address 4::1/64
 ipv6 ospf 10 area 1
interface Vlan1
```

```
 no ip address
 shutdown
router ospf 10
 router-id 4.4.4.4
 log-adjacency-changes
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

r4#show ipv6 route

```
IPv6 Routing Table - 13 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   1::/64 [110/192]
     via FE80::1, Serial0/1/0
O   2::/64 [110/128]
     via FE80::1, Serial0/1/0
C   3::/64 [0/0]
     via Serial0/1/0, directly connected
L   3::2/128 [0/0]
     via Serial0/1/0, receive
C   4::/64 [0/0]
     via Serial0/1/1, directly connected
L   4::1/128 [0/0]
     via Serial0/1/1, receive
O   A:1::/64 [110/193]
     via FE80::1, Serial0/1/0
O   A:2::/64 [110/129]
     via FE80::1, Serial0/1/0
O   A:3::/64 [110/65]
     via FE80::1, Serial0/1/0
C   A:4::/64 [0/0]
     via GigabitEthernet0/0/0, directly connected
L   A:4::A/128 [0/0]
     via GigabitEthernet0/0/0, receive
O   A:5::/64 [110/65]
     via FE80::2, Serial0/1/1
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 5

r5#show run

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname r5
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
```

```
 ip address 192.168.50.1 255.255.255.0
 ip ospf 10 area 1
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address A:5::A/64
 ipv6 ospf 10 area 1
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.14 255.255.255.252
 ip ospf 10 area 1
 ipv6 address FE80::2 link-local
 ipv6 address 4::2/64
 ipv6 ospf 10 area 1
 clock rate 2000000
interface Serial0/1/1
 no ip address
 clock rate 2000000
 shutdown
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 5.5.5.5
 log-adjacency-changes
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

r5#show ipv6 route

```
IPv6 Routing Table - 12 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  1::/64 [110/256]
     via FE80::1, Serial0/1/0
OI  2::/64 [110/192]
     via FE80::1, Serial0/1/0
OI  3::/64 [110/128]
     via FE80::1, Serial0/1/0
C   4::/64 [0/0]
     via Serial0/1/0, directly connected
L   4::2/128 [0/0]
     via Serial0/1/0, receive
OI  A:1::/64 [110/257]
     via FE80::1, Serial0/1/0
OI  A:2::/64 [110/193]
     via FE80::1, Serial0/1/0
OI  A:3::/64 [110/129]
     via FE80::1, Serial0/1/0
```

```
O   A:4::/64 [110/65]
     via FE80::1, Serial0/1/0
C   A:5::/64 [0/0]
     via GigabitEthernet0/0/0, directly connected
L   A:5::A/128 [0/0]
     via GigabitEthernet0/0/0, receive
L   FF00::/8 [0/0]
     via Null0, receive
```

Problems

This lab was very similar to the single-area IPv4 OSPF lab which I had completed previously, so I didn't run into much trouble. However, I ran into an interesting problem where no device could reach router five from area 0.

First, I tested if I could ping to router five from router one. I could reach its neighbor, router four, but could not reach beyond to router five. Since there were routes to all the networks except the one on router five, I concluded that this was likely an issue on router five's serial interface – the interface bridging router four to router five. To troubleshoot the interface, I went into router five and double checked the IP addresses and the OSPFv3 configuration.

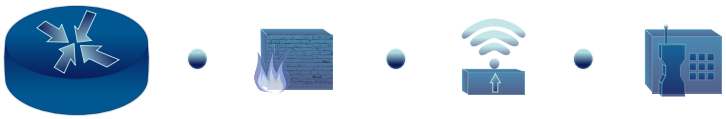
After many show commands, I noticed router five's OSPFv3 process ID didn't match the other routers. The process ID on router five was 1, whereas all the other routers had a process ID of 10. This was a typo I had made, but my research indicated that process ID didn't have to match for OSPF to function. This isn't totally wrong; however, with further investigation, I realized each process retains a different routing table. While connectivity had been established, it was understandable how routes could have messed up. I decided to wipe router 5 clean and start over, correcting the process IDs, which solved the problem.

Conclusion

In this lab, I set up multi-area OSPFv2 and OSPFv3 networks in CISCO packet tracer. Though it was just a simulation, I think I could take this knowledge and configure OSPFv3 on a physical catalyst 4321 router. Being able to configure an IGP is important for networks with multiple subnets and can be used alongside more complicated protocols like iBGP. An insight I took from this lab was to narrow down a problem before pasting in unnecessary commands found online.

Specialized Area OSPF

Implementing Stubby, Totally Stubby and Not so Stubby Areas



Purpose

The purpose of this lab was to use our general knowledge of OSPF to figure out how to set up a Stubby, Totally Stubby, and Not so Stubby area. During our deeper dive into OSPF, we learnt more about the information packets OSPF routers send to one another so each router can form proper routes. These packets are known as *Link-state Advertisements* (LSAs) and there are roughly seven important types of them. Only certain types of LSA packets will show up in certain areas which can help network engineers diagnose the type of the area. For example, LSA type 7 packets will show up in *Not so Stubby* areas, but not in *Stubby* nor *Totally Stubby* areas. We also configured EIGRP to advertise external routes into the OSPF network.

Background Information

Routing

Routing is a significant process in networking as it allows hosts on different IP networks to connect to each other. *Open Shortest Path First* (OSPF) is a routing protocol simplifying the process of creating routes by using algorithms to figure out the directions automatically. OSPF excels in interior networks, which are smaller in scale, but would crash in large networks with hundreds of routes. In networking, routes are ultimately just *directions* for packets.

There are two options when dealing with traffic on a network; you can configure *static routes*, or you can set up a *routing protocol*. I like to think of static routes as absolute directions drawn onto a map, set in stone and unchangeable. The map can't be altered unless it is manually redrawn. If you were to follow the map, you might find some of the routes to be outdated.

It would be nice if routes were adaptable, if they could update based on the fastest paths available. This is the difference between *static routing* and *routing protocols*. Routing protocols update their routing directions automatically based on information sent from neighbors. This is the magic of routing protocols: automatic updates and directions – like google maps – for packets. Routes are stored in a database on the router, known as a *routing table*.

Routing tables

Like a signpost at a fork in the road, routers contain directions for different destinations. These directions are stored in RAM memory on the router, which means that they are temporary; RAM memory can be accessed much faster than hard drives or SSDs but is not saved after the device shuts down. Let's look at an example of a packet arriving at a router.

A packet arrives at a router. This router has three interfaces: north, south, and east. The packet arrived on the east interface, so it either must turn north or south, assuming one of these paths lead to the destination. Luckily, there are directions in the router: *10.0.0.0/24* out interface *north*; *172.16.0.0/24* out interface *south*. The packet has a destination address of *10.0.0.3*, which

matches up with the *north* interface. The router sends the packet out the north interface. Routes are either generated statically, by the admin, or automatically by routing protocols such as OSPF, BGP, etc.

Here is an example of a routing table:

```
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 11 subnets, 2 masks
O IA    10.10.10.0/30 [110/128] via 10.10.10.5, 01:03:27, Serial0/1/1
C       10.10.10.4/30 is directly connected, Serial0/1/1
L       10.10.10.6/32 is directly connected, Serial0/1/1
C       10.10.10.8/30 is directly connected, Serial0/1/0
L       10.10.10.9/32 is directly connected, Serial0/1/0
O IA    10.10.10.12/30 [110/128] via 10.10.10.10, 01:03:27, Serial0/1/0
C       10.10.10.16/30 is directly connected, Serial0/2/0
L       10.10.10.17/32 is directly connected, Serial0/2/0
O IA    10.10.10.20/30 [110/128] via 10.10.10.18, 01:03:27, Serial0/2/0
O IA    10.10.10.24/30 [110/192] via 10.10.10.18, 01:03:27, Serial0/2/0
O E2    10.10.10.28/30 [110/100] via 10.10.10.18, 01:03:27, Serial0/2/0
```

Ignoring the letters on the left (the origin of the route), we can see a range of addresses and the corresponding interface leading towards them. For example, *10.10.10.0/30* addresses direct out the *Serial0/1/1* interface. "Via *ip*", is also commonly seen as a direction, indicating that a packet should be sent to the specified neighboring router. Sometimes there is a combination of directions: both *interface* and *neighboring ips*.

OSPF

Since we've defined routing and routing tables, I can go into more detail on how OSPF functions. Each router is like a junction for packets; packets usually have multiple roads they can turn down to reach further junctions, ultimately ending at their destination. Every router running OSPF will communicate with neighbor OSPF routers to relay statuses and updates about new routes and preferred paths. By sharing information to neighbor OSPF routers, information can spread through an OSPF network regardless of hop counts between routers. The packets OSPF broadcasts to relay information are known as *Link-State Advertisements* (LSAs).

- Type 1: *Router LSA*
  - Generated by every router.
  - Contains information about the router and lists links to other routers or networks *in the same area*.
  - Appears in a local area only and will be dropped by *Area Border Routers* (ABR).
  - The link state ID is the router ID of the router who originated the LSA packet.
- Type 2: *Network LSA*
  - Only generated by the Designated Router (DR) in a broadcast network type. *For example, if four routers are connected to the same switch, this system becomes a broadcast network. In a broadcast network, one router will be designated to handle most of the updates between the other routers, conserving bandwidth.*
  - Contains *the subnet of the broadcast segment*.

- Appears in a local area only and will be dropped by *Area Border Routers*.
- The link state ID is the IP address of the DR.

❖ Type 3: *Summary LSA*
- Generated by an *Area Border Router*.
- Informs *external areas* about networks in a *local area*.
- Forwarded by *Area Border Routers*.
- The link state ID is the network address of the advertising ABR.

❖ Type 4: *ASBR Summary LSA*
- Generated by an *Area Border Router* in an area containing an *Autonomous System Border Router* (ASBR).
  ASBRs are routers that bridge different routing protocols.
- Advertises routes to the *Autonomous System Border Router* in the area.
- Flooded in all areas except the area containing the ASBR.
- The link state is the ASBR's router ID.

❖ Type 5: *ASBR External LSA*
- Generated by an *Autonomous System Border Router.*
- Advertises external routes connected to the ASBR.
- Flooded through all areas.
- The link state ID is the external network number.

❖ Type 6: *Group Membership LSA*
- Designed for *Multicast OSPF* (MOSPF) but is no longer supported by Cisco.
- MOSPF is deprecated as of OSPFv3 and is not widely used.

❖ Type 7: *Not so Stubby Area LSA*
- Generated for external routes that enter a *Not So Stubby Area* (NSSA).
  NSSAs block externally distributed routes to save bandwidth.
- LSA type 5 packets are blocked or translated to LSA type 7 packets when entering an NSSA. Once the packets exit an *Area Border Router* in the NSSA, they are retranslated back to type 5 LSA packets.

Cost and OSPFv3

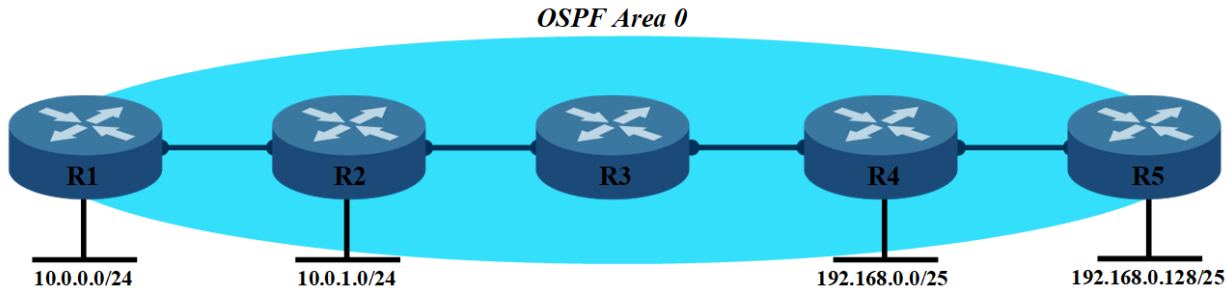
OSPF configured routers rely on *cost* to commute the shortest path through a network. While you can set the cost manually, OSPF will automatically determine the cost value per interface based on a *reference bandwidth* – usually the bandwidth of the fastest interface in your network – and *interface bandwidth***,** the bandwidth of the interface being assessed.

There are two OSPF protocols that can be configured on a router: OSPFv2 and OSPFv3, the main difference being *OSPFv2* routes *IPv4* and *OSPFv3* routes *IPv6*. OSPFv3 has nine *Link-State Advertisements*. LSAs are used to communicate different states and information of an OSPF router, such as a neighbor's local topology, to build the *routing table*. Although there are other routing protocols such as EIGRP, OSPF is massively adopted in large enterprise networks because of its many benefits: route redundancy, the ability to run on most routers, classless routing, and loop-free topologies.

Multi-area OSPF

OSPF routers communicate to each other using LSA packets, but this communication comes at a cost: bandwidth. When OSPF runs across a large network, LSA packets consume more bandwidth, as there are more routers that send updates. If the network has low-bandwidth interfaces, LSA traffic could hinder the performance. But what if we could limit the amount of LSA traffic on a network?

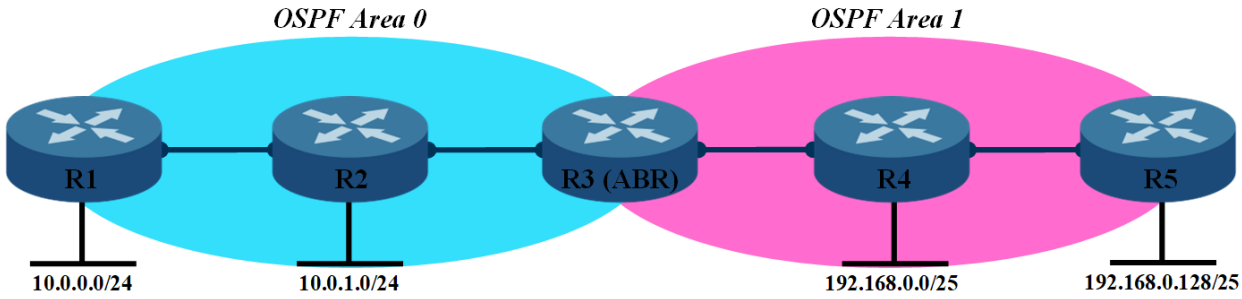
Multi-area OSPF is the process of dividing routers into multiple groups, known as *areas*, to reduce the size of LSA packets that need to be sent. Routes need to be specific and abundant for each network advertised within a local area. However, routers in a local area only need a broad definition of networks in external areas. Therefore, LSA packets across areas transmit summarizations by compressing multiple lines of routes into a single subnet. Let's compare a single-area OSPF network to a multi-area OSPF network.



In this topology, each router needs a specific route for every network in the area. LSA packets would be abundant in this area for each network OSPF is advertising. There would be no summarizations. A routing table on R2 may look something like the following:

- *10.0.0.0/24 out interface GigabitEthernet0*
- *192.168.0.0/25 out interface GigabitEthernet1*
- *192.168.0.128/25 out interface GigabitEthernet1*

Now let's divide this network into two areas.



Now that there are multiple areas, we can summarize networks for each area. Instead of routers four and five having routes to the *10.0.0.0/24* and *10.0.1.0/24* networks, they can share a singular, summarized route, *10.0.0.0/23*, that points towards the ASBR. When a packet enters the destination area, more precise routes will direct it to the destined network. An LSA packet to *area 0* from *area 1* might be distributing the following network, *192.168.0.0/24*, only consisting

of one prefix. Compared to LSAs containing all the specific routes, *route summarization* helps reduce LSA packet sizes. Now, the routing table on R2 may look something like the following:

- *10.0.0.0/24 out interface GigabitEthernet0*
- *192.168.0.0/24 out interface GigabitEthernet1*

Specialized Areas

While multi-area OSPF may appear to be the most optimal bandwidth conservation option, we can push OSPF further. *Specialized Areas* are additional OSPF configurations that *block* certain LSA types to further limit the amount of LSA traffic. However, specialized areas are circumstantial, requiring specific topologies to properly function. There are three specialized area types that I will cover in this paper: *Stubby*, *Totally Stubby*, and *Not So Stubby* areas.

OSPF Area Types and Accepted LSAs						
Area Types	LSA 1	LSA 2	LSA 3	LSA 4	LSA 5	LSA 7
Backbone Area	Yes	Yes	Yes	Yes	Yes	No
Non-Backbone Area	Yes	Yes	Yes	Yes	Yes	No
Stub Area	Yes	Yes	Yes	No	No	No
Totally Stubby Area	Yes	Yes	No	No	No	No
Not-So-Stubby Area	Yes	Yes	Yes	No	No	Yes

LSA types found per area

Totally Stubby Area

Totally Stubby Areas block the most LSA traffic by dropping all LSA packets except types 1 and 2, conserving the most bandwidth of all the specialized areas. A Totally Stubby area must contain only one *Area Border Router* so all external area traffic can flood out of a default gateway. There are no external routes because there is only one destination for a packet: out the ABR. All external area routes can be replaced by a singular default route. Totally Stubby areas

cannot contain an ASBR; this is important because ASBRs generate LSA type 4 and 5 traffic, which are not permitted in the area. While Totally Stubby areas conserve the most bandwidth, they are also very situational.

Stub/Stubby Area

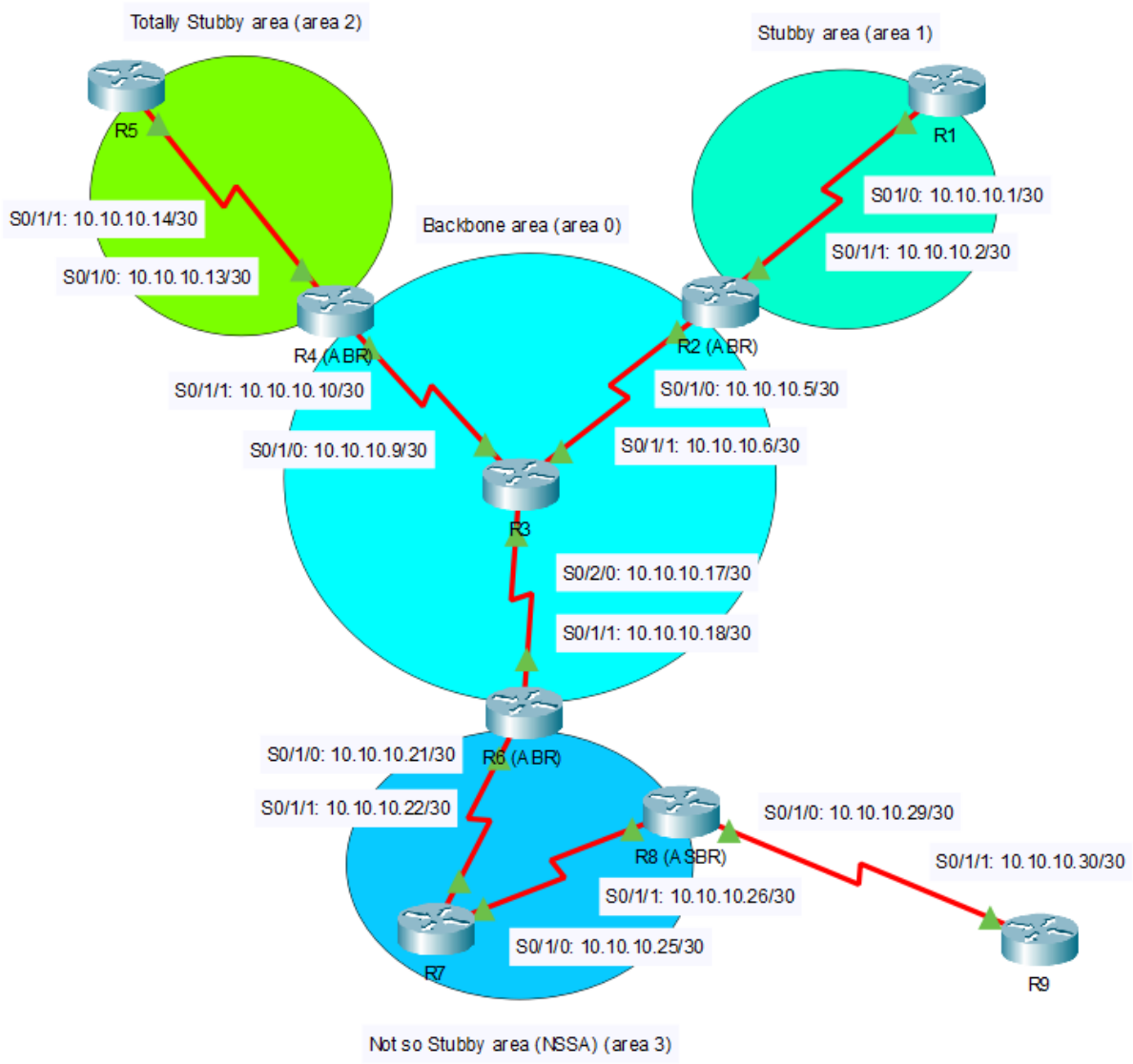
Stubby Areas have all the properties of Totally Stubby areas with a slight change in their topologies: they contain more than one *Area Border Router*. This change does come at a cost: LSA type 3 – *summary* – traffic is permitted throughout the area. By allowing more than one ABR in the area, ensuing uncertainty to where external area routes are directed, the area can no longer replace all external area routes with a singular default route. A Stubby area may need external area routes out some of the ABRs, but there will likely be an ABR that routes to the internet. In this case, the Stubby area will have a default route, much like a Totally Stubby area. Stubby areas are used since they retain smaller databases by excluding external network routes, though they do permit summary LSA traffic.

Not so Stubby Area

Not so Stubby areas (NSSA) are designed for topologies that include *Autonomous System Border Routers*. ASBRs are routers that run multiple routing protocols, bridging the routing protocols using *redistribution*. Like Stubby and Totally Stubby areas, NSSAs block external routes. But if ASBRs redistribute external routes, then how can NSSAs *block* those external routes?

NSSAs *translate* external routing information into type 7 LSA packets to camouflage external routes that ASBRs produce. Routers in the NSSA ignore type 7 LSA packets and forward them out of the area. Once the type 7 LSAs exit an *Area Border Router* of the Not so Stubby area, they are translated to type 5 LSAs, containing external routes.

Network Diagram



Summary

In this lab, I set up four specialized OSPF areas: a *Stubby*, *Totally Stubby*, *Not so Stubby*, and a *Backbone* area. Attached to the Not so Stubby area was an external EIGRP network to produce type 5 LSA traffic.

I started off by creating a basic topology of my network. It contained four areas, each designed to support a particular specialized OSPF area. Referencing the topology, I created an IP scheme composed of many small [/30] subnets to link the serial interfaces between neighboring routers. I configured the IP addresses in my network and activated the interfaces with a *no shutdown* command.

With the router interfaces set up, I began configuring my first area, the *backbone* area. I configured OSPFv2 like OSPFv3: advertising the networks in *interface-configuration* mode, which should have been all I needed to configure for routers in the backbone area. However, configuring OSPFv2 in such a way – omitting *network* statements – would later cause me many problems.

I configured the stubby area, using the *area [area number] stub* command; I configured the totally stubby area, using the *area [area number] stub no-summary* command; and I configured the not so stubby area, using the *area [area number] nssa* command on each router in the areas. Once my OSPF network was implemented, I needed to configure a small EIGRP network that connected to my NSSA. This was my first time configuring EIGRP, so I was glad to find similarities it had with OSPF configuration.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

Router(config)# **interface [*interface*] [*id*]**

- Enables configuration on a specific interface

// OSPF

Router(config)# **router ospf [*process id*]**

- Enables the OSPF routing protocol and enters OSPF router configuration mode

Generally, OSPF process ids should be the same, though OSPF should still form adjacencies with different process ids. Each OSPF process retains a different routing table; depending on the configuration, process ID could determine what routes are redistributed. A router can run multiple OSPF processes but will contain a separate OSPF database per process.

Router(config-router)# **router-id [*router id*]**

- Uniquely determines an OSPF router within a domain

Router ids are automatically determined by the highest loopback interface if they are not manually defined. Router ids can play a part in DR/BDR elections.

Router(config-router)# **network [*network address*] [*wildcard mask*] area [*area number*]**

- Advertises the specified subnet to neighbor OSPF routers

Router(config-router)# **area [*area number*] [stub/nssa]** *no-summary*

- Limits LSA traffic based on a specialized area definition

This command is typed in router configuration mode. Use the following to define specialized areas:

- *Totally Stubby area | stub no-summary*
- *Stubby area | stub*
- *Totally Not so Stubby area | nssa no-summary*
- *Not so Stubby area | nssa*

// EIGRP

Router(config)# **router eigrp [*instance*]**

- Enables EIGRP of a particular instance and enters router configuration mode.

There can be multiple instances of EIGRP running on a router, however, adjacent routers will only communicate if they are using the same instance.

Router(config-router)# **network [*network address*] [*wildcard mask*]**

- Advertises the specified subnet to neighbor routers

Other EIGRP routers will gain knowledge of this network and form routes to it.

// Show Commands

Router# **show [*ipv4/ipv6*] ospf database**

- Displays the routing database

Router# **show [*ipv4/ipv6*] ospf neighbor**

- Displays information about adjacent routers configured with OSPF

Router# **show [*ipv4/ipv6*] ospf interface**

- Displays information about each interface configured with OSPF

// Redistribution

Router(config-router)# **redistribute [*routing protocol*] [*protocol instance*]** *<metric <value>> subnets*

- Redistributes routes from a routing protocol into another local routing protocol

The routing protocol defined will be distributed in the local router that the user is in. There are many different additional options when redistributing routes, but I've found the metric and subnets to be the most useful. Each routing protocol has a different metric, so when redistributing be sure to use the right one. Subnets usually refers to redistributing classless networks.

Configurations

Router 1

```
R1#show run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R1
no ip cef
no ipv6 cef
```

```
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.1 255.255.255.252
interface Serial0/1/1
 no ip address
 clock rate 2000000
 shutdown
interface GigabitEthernet0/2/0
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/1
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/2
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/3
 switchport mode access
 switchport nonegotiate
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 1.1.1.1
 log-adjacency-changes
 area 1 stub
 network 10.10.10.0 0.0.0.3 area 1
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

R1#sh ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is 10.10.10.2 to network 0.0.0.0
     10.0.0.0/8 is variably subnetted, 8 subnets, 2 masks
C       10.10.10.0/30 is directly connected, Serial0/1/0
L       10.10.10.1/32 is directly connected, Serial0/1/0
O IA    10.10.10.4/30 [110/128] via 10.10.10.2, 00:59:41, Serial0/1/0
O IA    10.10.10.8/30 [110/192] via 10.10.10.2, 00:59:21, Serial0/1/0
O IA    10.10.10.12/30 [110/256] via 10.10.10.2, 00:59:21, Serial0/1/0
O IA    10.10.10.16/30 [110/192] via 10.10.10.2, 00:59:21, Serial0/1/0
O IA    10.10.10.20/30 [110/256] via 10.10.10.2, 00:59:21, Serial0/1/0
O IA    10.10.10.24/30 [110/320] via 10.10.10.2, 00:59:21, Serial0/1/0
```

```
O*IA 0.0.0.0/0 [110/65] via 10.10.10.2, 00:59:41, Serial0/1/0
```

Router 2

```
R2#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R2
no ip cef
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.5 255.255.255.252
interface Serial0/1/1
 ip address 10.10.10.2 255.255.255.252
 clock rate 2000000
interface GigabitEthernet0/2/0
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/1
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/2
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/3
 switchport mode access
 switchport nonegotiate
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 2.2.2.2
 log-adjacency-changes
 area 1 stub
 network 10.10.10.0 0.0.0.3 area 1
 network 10.10.10.4 0.0.0.3 area 0
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end

R2#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
```

```
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 10 subnets, 2 masks
C       10.10.10.0/30 is directly connected, Serial0/1/1
L       10.10.10.2/32 is directly connected, Serial0/1/1
C       10.10.10.4/30 is directly connected, Serial0/1/0
L       10.10.10.5/32 is directly connected, Serial0/1/0
O       10.10.10.8/30 [110/128] via 10.10.10.6, 01:01:04, Serial0/1/0
O IA    10.10.10.12/30 [110/192] via 10.10.10.6, 01:01:04, Serial0/1/0
O       10.10.10.16/30 [110/128] via 10.10.10.6, 01:01:04, Serial0/1/0
O IA    10.10.10.20/30 [110/192] via 10.10.10.6, 01:01:04, Serial0/1/0
O IA    10.10.10.24/30 [110/256] via 10.10.10.6, 01:01:04, Serial0/1/0
O E2    10.10.10.28/30 [110/100] via 10.10.10.6, 01:01:04, Serial0/1/0
```

Router 3

```
R3#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R3
no ip cef
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.9 255.255.255.252
interface Serial0/1/1
 ip address 10.10.10.6 255.255.255.252
 clock rate 2000000
interface Serial0/2/0
 ip address 10.10.10.17 255.255.255.252
interface Serial0/2/1
 no ip address
 clock rate 2000000
 shutdown
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 3.3.3.3
 log-adjacency-changes
 network 10.10.10.4 0.0.0.3 area 0
 network 10.10.10.16 0.0.0.3 area 0
 network 10.10.10.8 0.0.0.3 area 0
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

```
R3#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 11 subnets, 2 masks
O IA    10.10.10.0/30 [110/128] via 10.10.10.5, 01:03:27, Serial0/1/1
C       10.10.10.4/30 is directly connected, Serial0/1/1
L       10.10.10.6/32 is directly connected, Serial0/1/1
C       10.10.10.8/30 is directly connected, Serial0/1/0
L       10.10.10.9/32 is directly connected, Serial0/1/0
O IA    10.10.10.12/30 [110/128] via 10.10.10.10, 01:03:27, Serial0/1/0
C       10.10.10.16/30 is directly connected, Serial0/2/0
L       10.10.10.17/32 is directly connected, Serial0/2/0
O IA    10.10.10.20/30 [110/128] via 10.10.10.18, 01:03:27, Serial0/2/0
O IA    10.10.10.24/30 [110/192] via 10.10.10.18, 01:03:27, Serial0/2/0
O E2    10.10.10.28/30 [110/100] via 10.10.10.18, 01:03:27, Serial0/2/0
```

Router 4

```
R4#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R4
no ip cef
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.13 255.255.255.252
interface Serial0/1/1
 ip address 10.10.10.10 255.255.255.252
 clock rate 2000000
interface GigabitEthernet0/2/0
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/1
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/2
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/3
 switchport mode access
 switchport nonegotiate
interface Vlan1
 no ip address
```

```
 shutdown
router ospf 10
 router-id 4.4.4.4
 log-adjacency-changes
 area 2 stub no-summary
 network 10.10.10.12 0.0.0.3 area 2
 network 10.10.10.8 0.0.0.3 area 0
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

R4#sh ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 10 subnets, 2 masks
O IA    10.10.10.0/30 [110/192] via 10.10.10.9, 01:05:29, Serial0/1/1
O       10.10.10.4/30 [110/128] via 10.10.10.9, 01:05:29, Serial0/1/1
C       10.10.10.8/30 is directly connected, Serial0/1/1
L       10.10.10.10/32 is directly connected, Serial0/1/1
C       10.10.10.12/30 is directly connected, Serial0/1/0
L       10.10.10.13/32 is directly connected, Serial0/1/0
O       10.10.10.16/30 [110/128] via 10.10.10.9, 01:05:29, Serial0/1/1
O IA    10.10.10.20/30 [110/192] via 10.10.10.9, 01:05:29, Serial0/1/1
O IA    10.10.10.24/30 [110/256] via 10.10.10.9, 01:05:29, Serial0/1/1
O E2    10.10.10.28/30 [110/100] via 10.10.10.9, 01:05:29, Serial0/1/1
```

Router 5

R5#sh run

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R5
no ip cef
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 no ip address
 clock rate 2000000
 shutdown
interface Serial0/1/1
 ip address 10.10.10.14 255.255.255.252
```

```
 clock rate 2000000
interface GigabitEthernet0/2/0
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/1
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/2
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/3
 switchport mode access
 switchport nonegotiate
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 5.5.5.5
 log-adjacency-changes
 area 2 stub no-summary
 network 10.10.10.12 0.0.0.3 area 2
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

R5#sh ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is 10.10.10.13 to network 0.0.0.0
     10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C       10.10.10.12/30 is directly connected, Serial0/1/1
L       10.10.10.14/32 is directly connected, Serial0/1/1
O*IA 0.0.0.0/0 [110/65] via 10.10.10.13, 01:07:50, Serial0/1/1
```

Router 6

R6#sh run

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R6
no ip cef
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
```

```
 shutdown
interface Serial0/1/0
 ip address 10.10.10.21 255.255.255.252
interface Serial0/1/1
 ip address 10.10.10.18 255.255.255.252
 clock rate 2000000
interface GigabitEthernet0/2/0
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/1
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/2
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/3
 switchport mode access
 switchport nonegotiate
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 6.6.6.6
 log-adjacency-changes
 area 3 nssa
 network 10.10.10.20 0.0.0.3 area 3
 network 10.10.10.16 0.0.0.3 area 0
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

R6#sh ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 10 subnets, 2 masks
O IA    10.10.10.0/30 [110/192] via 10.10.10.17, 01:09:32, Serial0/1/1
O       10.10.10.4/30 [110/128] via 10.10.10.17, 01:09:32, Serial0/1/1
O       10.10.10.8/30 [110/128] via 10.10.10.17, 01:09:32, Serial0/1/1
O IA    10.10.10.12/30 [110/192] via 10.10.10.17, 01:09:32, Serial0/1/1
C       10.10.10.16/30 is directly connected, Serial0/1/1
L       10.10.10.18/32 is directly connected, Serial0/1/1
C       10.10.10.20/30 is directly connected, Serial0/1/0
L       10.10.10.21/32 is directly connected, Serial0/1/0
O       10.10.10.24/30 [110/128] via 10.10.10.22, 01:09:37, Serial0/1/0
O N2    10.10.10.28/30 [110/100] via 10.10.10.22, 01:09:37, Serial0/1/0
```

Router 7

R7#sh run

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
```

```
hostname R7
no ip cef
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.25 255.255.255.252
interface Serial0/1/1
 ip address 10.10.10.22 255.255.255.252
 clock rate 2000000
interface GigabitEthernet0/2/0
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/1
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/2
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/3
 switchport mode access
 switchport nonegotiate
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 7.7.7.7
 log-adjacency-changes
 area 3 nssa
 network 10.10.10.24 0.0.0.3 area 3
 network 10.10.10.20 0.0.0.3 area 3
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

```
R7#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 10 subnets, 2 masks
O IA    10.10.10.0/30 [110/256] via 10.10.10.21, 01:11:31, Serial0/1/1
O IA    10.10.10.4/30 [110/192] via 10.10.10.21, 01:11:31, Serial0/1/1
O IA    10.10.10.8/30 [110/192] via 10.10.10.21, 01:11:31, Serial0/1/1
O IA    10.10.10.12/30 [110/256] via 10.10.10.21, 01:11:31, Serial0/1/1
O IA    10.10.10.16/30 [110/128] via 10.10.10.21, 01:11:31, Serial0/1/1
```

```
C       10.10.10.20/30 is directly connected, Serial0/1/1
L       10.10.10.22/32 is directly connected, Serial0/1/1
C       10.10.10.24/30 is directly connected, Serial0/1/0
L       10.10.10.25/32 is directly connected, Serial0/1/0
O N2    10.10.10.28/30 [110/100] via 10.10.10.26, 00:15:59, Serial0/1/0
```

Router 8

```
R8#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R8
no ip cef
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.29 255.255.255.252
interface Serial0/1/1
 ip address 10.10.10.26 255.255.255.252
 clock rate 2000000
interface GigabitEthernet0/2/0
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/1
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/2
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/3
 switchport mode access
 switchport nonegotiate
interface Vlan1
 no ip address
 shutdown
router eigrp 1
 redistribute ospf 10 metric 1000 33 255 1 1500
 network 10.10.10.28 0.0.0.3
 auto-summary
router ospf 10
 router-id 8.8.8.8
 log-adjacency-changes
 area 3 nssa
 redistribute eigrp 1 metric 100 subnets
 redistribute static subnets
 redistribute connected
 network 10.10.10.24 0.0.0.3 area 3
ip classless
ip flow-export version 9
line con 0
line aux 0
```

```
line vty 0 4
 login
end
```

```
R8#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 10 subnets, 2 masks
O IA    10.10.10.0/30 [110/320] via 10.10.10.25, 01:13:37, Serial0/1/1
O IA    10.10.10.4/30 [110/256] via 10.10.10.25, 01:13:37, Serial0/1/1
O IA    10.10.10.8/30 [110/256] via 10.10.10.25, 01:13:37, Serial0/1/1
O IA    10.10.10.12/30 [110/320] via 10.10.10.25, 01:13:37, Serial0/1/1
O IA    10.10.10.16/30 [110/192] via 10.10.10.25, 01:13:37, Serial0/1/1
O       10.10.10.20/30 [110/128] via 10.10.10.25, 01:13:47, Serial0/1/1
C       10.10.10.24/30 is directly connected, Serial0/1/1
L       10.10.10.26/32 is directly connected, Serial0/1/1
C       10.10.10.28/30 is directly connected, Serial0/1/0
L       10.10.10.29/32 is directly connected, Serial0/1/0
```

Router 9

```
R9#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R9
no ip cef
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 no ip address
 clock rate 2000000
 shutdown
interface Serial0/1/1
 ip address 10.10.10.30 255.255.255.252
 clock rate 2000000
interface GigabitEthernet0/2/0
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/1
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/2
 switchport mode access
 switchport nonegotiate
interface GigabitEthernet0/2/3
```

```
 switchport mode access
 switchport nonegotiate
interface Vlan1
 no ip address
 shutdown
router eigrp 1
 network 10.10.10.28 0.0.0.3
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

R9#sh ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 9 subnets, 2 masks
D EX    10.10.10.0/30 [170/3080448] via 10.10.10.29, 01:15:29, Serial0/1/1
D EX    10.10.10.4/30 [170/3080448] via 10.10.10.29, 01:15:29, Serial0/1/1
D EX    10.10.10.8/30 [170/3080448] via 10.10.10.29, 01:15:29, Serial0/1/1
D EX    10.10.10.12/30 [170/3080448] via 10.10.10.29, 01:15:29, Serial0/1/1
D EX    10.10.10.16/30 [170/3080448] via 10.10.10.29, 01:15:29, Serial0/1/1
D EX    10.10.10.20/30 [170/3080448] via 10.10.10.29, 01:15:39, Serial0/1/1
D EX    10.10.10.24/30 [170/3080448] via 10.10.10.29, 01:15:54, Serial0/1/1
C       10.10.10.28/30 is directly connected, Serial0/1/1
L       10.10.10.30/32 is directly connected, Serial0/1/1
```

Pings

R1 to R5:

```
R1#ping 10.10.10.14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.10.10.14, timeout is 2 seconds:
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/20/25 ms
```

R1 to R9:

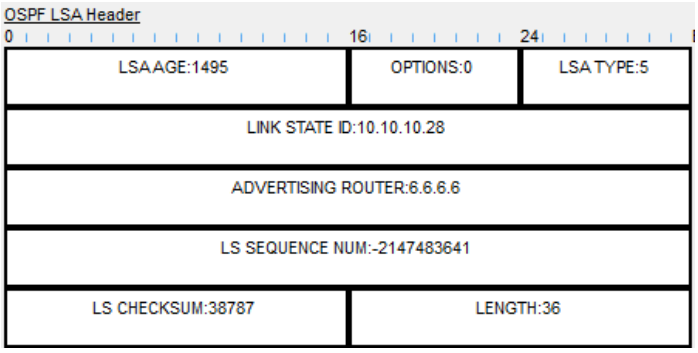
```
R1#ping 10.10.10.30
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.10.10.30, timeout is 2 seconds:
Success rate is 100 percent (5/5), round-trip min/avg/max = 30/35/41 ms
```

R5 to R9:

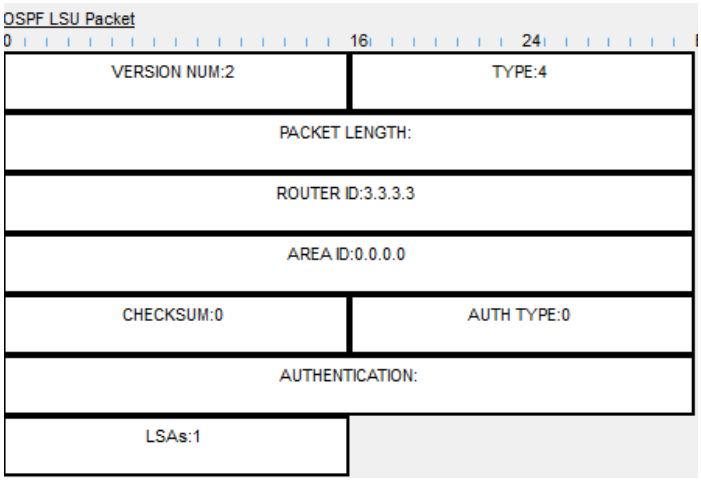
```
R5#ping 10.10.10.30
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.10.10.30, timeout is 2 seconds:
Success rate is 100 percent (5/5), round-trip min/avg/max = 31/34/40 ms
```

Specialized Area Packets

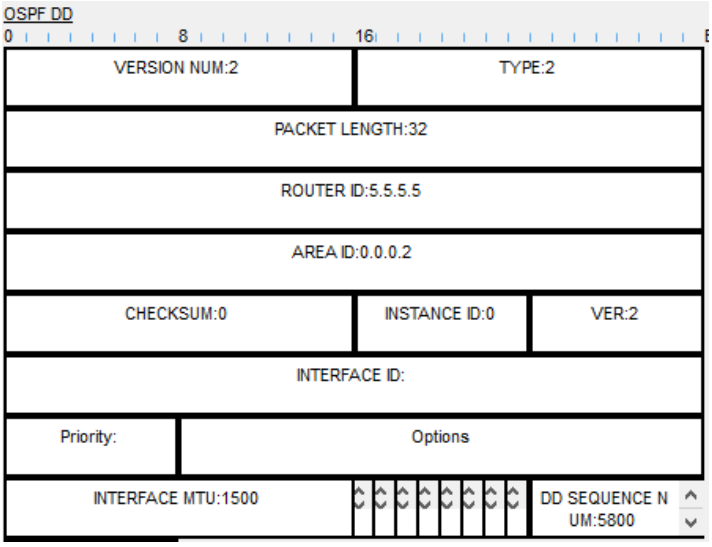
LSA type 5 being broadcasted into area 0 by Router 6



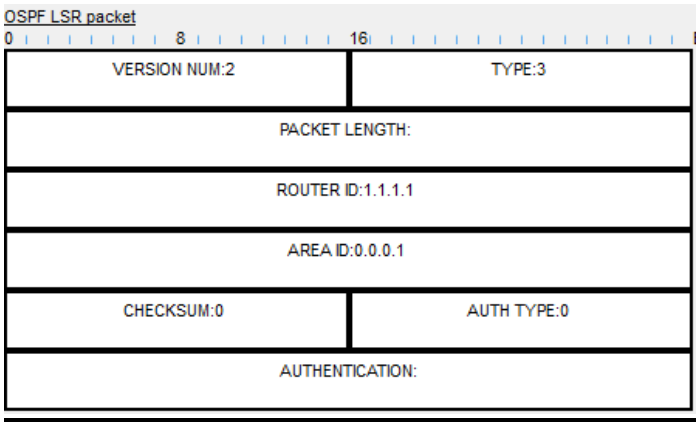
LSA type 4 being advertised by Router 3



LSA type 2 being sent in area 2



LSA type 3 being sent in area 1



Problems

Stubby Mismatch

After doing the initial configuration for a *Stubby* area in area 1, I encountered an OSPF adjacency error between R1 and R2: *"Hello from 10.10.10.2 with mismatched Stub/Transit area option bit"*. Adjacency debugging was on, and the log was filling with this message. Typically, when something is mismatched in networking, the admin did not configure the same command on both devices. Some researching in google helped me prove this theory: both adjacent interfaces needed to have the *area 1 stub* command for the relationship to be stubby. I only configured the command on R1, so I added it in R2.

Configuring a Not so Stubby Area

While configuring the *Not so Stubby* area, I ran into my second problem: I did not know the command to configure a NSSA. Luckily, there are many resources online, so I quickly came across a document containing a command that looked promising, *area [area number] nssa*, with various additional arguments. However, packet tracer didn't support the same additional options the document had. The issue here was figuring out what arguments and parameters I should use. Both the stubby and totally stubby areas had similar configurations statements. The difference was the *"no-summary"* line of the *area [area number] stub no-summary* command. The extra *"no-summary"* portion determined whether I wanted to exclude summary LSAs, added for totally stubby areas. I decided to omit the *"no-summary"* since it is likely used for a *totally not so stubby area*.

Proper OSPFv2 Configuration

The toughest problem I encountered occurred after I finished setting up the NSSA. The NSSA received all the routing information from the other areas but never advertised its area summary. In other words, the stub, totally stubby, and backbone areas all had routes to the

NSSA, but the NSSA had no routes back. I rechecked each IP and subnet mask, confirmed all the neighbor adjacencies and restarted every OSPF process. All the IPs were correct, the adjacencies were formed, and the interfaces were up. Every router should have been configured correctly in their respective area and they had the same process ID.

Finally, I decided to question the obscure way that I configured OSPFv2. To recap, there are two ways of advertising OSPFv2 subnetworks: using *network statements* in *router-configuration* mode or configuring the area in *interface-configuration* mode. For example, the command *10.10.10.0 0.0.0.255 area 0* in *router-configuration* mode and the command *ip ospf 10 area 0* on the *10.10.10.0/24* interface in *interface-configuration* mode both accomplish the same goal: advertising the *10.10.10.0/24* network. I configured OSPFv2 using the *interface-configuration* command but had not tried implementing the *network* statements. I wiped all OSPF configurations on each interface and replaced them with network statements. This may have been a bug in packet tracer, but I will personally be using network statements from this point fourth as I deem them more reliable.

Eigrp to Ospf Redistribution

The fourth problem I encountered was working out how to properly distribute EIGRP external routes into my OSPF network. The basic *redistribute* statement was not working. This meant I needed more than just a basic redistribute statement – I needed to come up with optional parameters that would satisfy OSPF. I tried many redistribution statements involving EIGRP metrics, but eventually noticed the following message that would pop up each time: *only classful networks will be redistributed*. My networks were subnetted, so they were *classless*. Now I narrowed down problem. I required a configuration that would redistribute *classless* networks.

After some searching, I found an additional argument that allowed classless addresses to be redistributed. My ever-growing command, *redistribute eigrp [instance] metric [#]*, solely needed the *subnets* argument appended to the end. After applying the changes then refreshing the OSPF process, the EIGRP routes began to distribute.

Ospf to Eigrp Redistribution

The fifth problem was the inverse of my last problem. This time, OSPF routes would not distribute into the EIGRP network. The base command, *redistribute ospf 10*, did not appear to redistribute my process 10 OSPF network. Perhaps this command only worked on classful networks. Thanks to my previous problem, I gained the insight that redistributing classless networks is slightly different to redistributing classful networks.

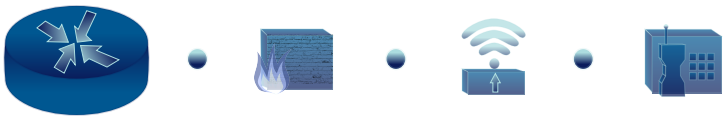
After refreshing the OSPF process ID a couple times for good measure, I decided to ask my peers. I was linked to a website that yielded a similar command with a unique metric: *redistribute ospf 10 metric 1000 33 255 1 1500*. Perhaps the metric somehow specified that these routes were classless. Whatever the issue was, redistributing OSPF with this specific metric solved the problem.

Conclusion

I set up stub, totally stubby, not so stubby and backbone areas using OSPFv2. This lab was riddled with more problems than my other ones, but it was nice to get some troubleshooting experience. Specialized areas might be a little overkill for a network with only ten routers, but the concepts are important to know for progressing further with OSPF. If I could give one piece of advice for configuring OSPFv2, it would be to use *network* statements rather than *interface-configuration* statements.

eBGP

An Intro to Routing Traffic with eBGP in Packet Tracer



Purpose

The networking infrastructure of the world is supported by BGP. If BGP was nonexistent, it would not be possible for us to google "sushi places near me" and get convenient responses out of nowhere in mere milliseconds. In this paper, I go over my process of configuring BGP and provide some background information along the way.

Background Information

Routing

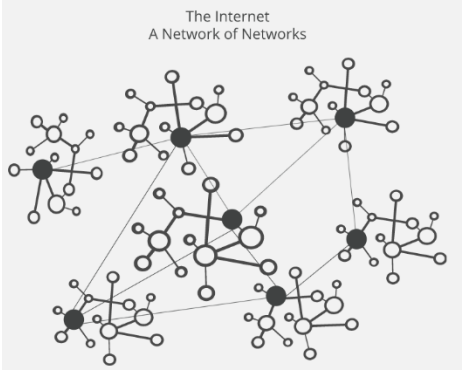
Routing is a significant process in networking as it allows hosts on different IP networks to connect to each other. *Border Gateway Protocol* (BGP) is a routing protocol simplifying the process of creating routes by using algorithms to figure out the directions automatically. In networking, routes are ultimately just *directions* for packets.

There are two options when dealing with traffic on a network; you can configure *static routes*, or you can set up a *routing protocol*. I like to think of static routes as absolute directions drawn onto a map, set in stone and unchangeable. The map can't be altered unless it is manually redrawn. If you were to follow the map, you might find some of the routes to be outdated.

It would be nice if routes were adaptable, if they could update based on the fastest paths available. This is the difference between *static routing* and *routing protocols*. Routing protocols update their routing directions automatically based on information sent from neighbors. This is the magic of routing protocols: automatic updates and directions – like google maps – for packets.

What is BGP?

Border Gateway Protocol is the most popular external routing protocol, commonly used by ISPs (Internet Service Providers) to route customer traffic. Without BGP, the internet would not function nearly as well, if at all. Think of BGP as the postal service that delivers a letter to the recipient in the fastest and most efficient manner possible. When someone submits data across the internet, BGP is responsible for choosing the best path out of all preexisting available paths, which usually means passing through autonomous systems.



An example of Autonomous systems and their local networks

So, what are autonomous systems? *Autonomous Systems* (AS) are a collection of routers, each with their own lesser hierarchy of routers that eventually connects to local networks. Each autonomous system is aware of other autonomous system(s) and can broadly determine where to route traffic based on which autonomous system holds the desired destination. ASes typically belong to ISPs (Internet service providers) or other large high-tech organizations, such as tech companies, universities, or scientific institutions. The internet is run under a collection of autonomous systems.

Kingdom Analogy

I suppose one could think of an autonomous system as a form of kingdom. Each kingdom has a ruler that dictates certain policies that the underlying citizens and infrastructure abide to. For example, if a kingdom is landlocked, it likely has a high demand for fish and salt. Therefore, a *policy* is implemented where all traders from the nearest port town have free access to and from the kingdom. Different autonomous systems often have these unique routing *policies*.

There are many paths and roads in the kingdom internally, so much so that if one goes down, alternate routes are readily available. Some kingdoms have routes bridging them, but often a traveler (packet) will have to journey through multiple kingdoms to reach their desired destination. In other words, a packet may have to pass through multiple ASes to reach its destination.

Each AS is assigned a unique, 32-bit number, the *Autonomous System Number* (ASN). These numbers differentiate what "kingdom" a router falls under. Routers with the same AS are part of the same kingdom.

Internal and External BGP

As I vaguely covered in my analogy, there are two types of BGP: *internal BGP* (within kingdoms) and *external BGP* (between kingdoms).

External BGP (eBGP) is the bridge that connects autonomous systems, where neighbors can broadly exchange network prefixes to learn more about each other's networks.

Internal BGP (iBGP) is a TCP based protocol to help advertise and support eBGP routes. The kicker: iBGP alone does not do any routing. To route, one needs an IP based protocol. So why bother with iBGP at all?

Consider an old, flimsy wooden bridge. Driving a cargo truck across would collapse the bridge. But now, with iBGP, that bridge is reinforced with a concrete foundation, metal bearings, and arches to brace the heavy loads. BGP is the only protocol designed to support the hundreds of thousands of routes that make up the internet. As of writing this, the size of the full IPv4 BGP routing table is around 800,000 prefixes without even accounting for IPv6. For reference, the average OSPF router would suffer at around 6000 prefixes. Therefore, iBGP is often used in conjunction with an IGP; the IGP does the local routing whilst iBGP contains the major routing table.

Both internal and external BGP sessions establish neighbors based on a peering system. You define a peer with a neighbor statement: for example, *neighbor 10.0.0.1 remote-as 100* states that there is a router connected, *10.0.0.1* running under ASN *100*. The neighbor *10.0.0.1* would need to define this router as a neighbor for a complete peer adjacency to form. Once both routers point to each other, they are peered. Networks are advertised with network statements: for example, *network 10.0.0.0 mask 255.255.255.0* will add the prefix *10.0.0.0/24* to the routing table. Other routers will direct traffic for *10.0.0.0/24* towards the router with the network statement.

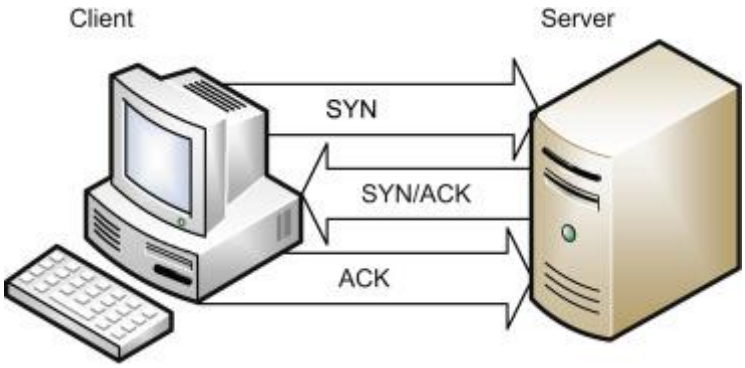
How does BGP function?

The main purpose BGP serves is forwarding traffic to an external network in the most efficient manner possible. Some factors that determine the best path are:

1. The path with the highest *weight*. This is a user defined variable.
2. The path with the highest *LOCAL_PREF*. Local preference determines which path is preferred when leaving a local AS.
3. The path with the highest *AS_PATH*. The main purpose of *AS_PATH* is to prevent infinite routing table updates. It is rather complicated, but essentially if a router goes down in a network, then this might cause the other routers to falsely change their paths, resulting in an infinite loop of changing paths. This can only happen in a distance-vector routing protocol such as BGP or RIP.
4. Favoring *eBGP* paths over iBGP paths.

BGP is a *distance-vector* routing protocol. Distance-vector routing protocols work by advertising routing tables to their neighbors. If the routes from the neighbor are better than the ones they currently have, the router will update its routing table to the preferable routes. Like all other routing protocols, BGP must first establish a neighbor adjacency with another BGP router to be able to exchange routing information. Unlike other routing protocols, BGP does not broadcast or multicast to discover other BGP neighbors. Neighbor relationships must be established manually and BGP uses TCP port 179 for the connection. There are a couple of different states BGP routers may encounter when becoming neighbors:

1. Idle. In Idle, BGP waits for a "start event". This could be when a new BGP neighbor is configured or when a reset occurs between peers that already had a connection. After the start event, BGP will initialize a TCP connection with the remote neighbor and initialize some functions. In success, BGP moves to the *Connect* state, while in failure, BGP remains in the *Idle* state.
2. Connect. In *Connect*, BGP waits for the TCP three-way handshake to complete. Both sides need to *synchronize* (SYN) and *acknowledge* (ACK) each other in a TCP three-way handshake. If the results are successful, BGP advances to the *OpenSent* state. If the results are unsuccessful, BGP begins the *Active* state.



3. Active. BGP will try another TCP three-way handshake to establish a connection with the remote neighbor. On success, BGP will transition to the *OpenSent* state. After a certain amount of time has passed with no success, BGP will revert to the *Connect* state.
4. OpenSent. BGP will wait for an "open message" from the remote neighbor. Open messages contain information about the BGP router, such as version, ASN, BGP router ID, and hold time. If the versions or hold times mismatch, BGP reverts to the *Idle* state. The ASN determines whether the BGP session will be running iBGP or eBGP. If the TCP session ever fails, BGP will revert to the *Active* state. If all passes, BGP will start sending keepalive messages to maintain the TCP session.
5. OpenConfirm. BGP waits for a keepalive message from the remote BGP neighbor. When keepalive messages are consistently received, BGP moves to the *Established* state. In any other case, BGP falls back to the *Idle* state.
6. Established. The neighbor adjacency has been formed. As long as keepalive messages are sent, the neighborship remains up. Otherwise, BGP resets back to Idle state.

Adjacencies are often formed by defining the *directly connected* interface as a neighbor, a common trait in most routing protocols. However, a technique when working with BGP is to use loopback interfaces as neighbors. Using loopbacks is common for iBGP but it also works with eBGP. Loopbacks are preferred because of redundancy: if the physical interface goes down, perhaps due to hardware, loopback interfaces will stay up since they are *virtual*.

Brief History of BGP

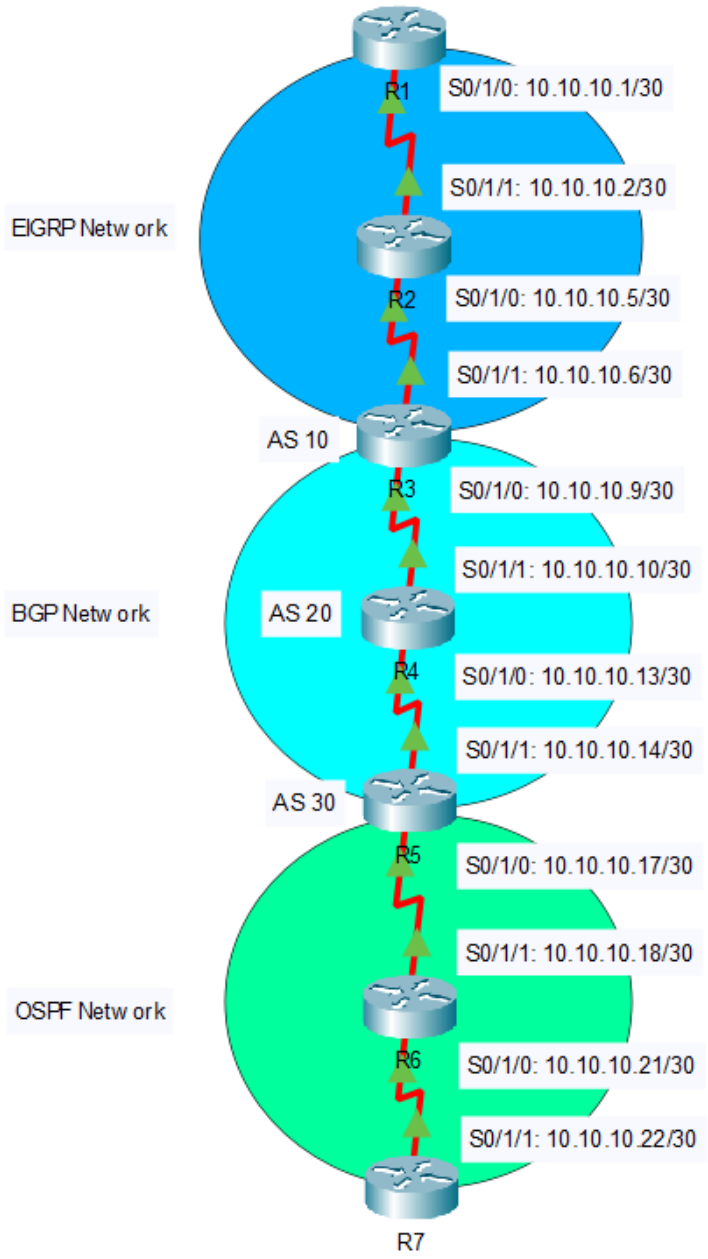
In the beginnings of the internet there was no BGP. Instead, there was *Gateway Gateway Protocol* (GGP), a brilliantly named protocol and a slightly more advanced version of an outdated protocol we have today – *Router Information Protocol* (RIP). Back then, routers were known as gateways, explaining the redundancy in the name.

Gateway Gateway Protocol was replaced by *Exterior Gateway Protocol* (EGP) in 1984, created to support the growing network infrastructure around the world. EGP introduced the concept of *autonomous systems,* which later became a big part of BGP.

However, engineers foresaw a fundamental problem with EGP: the inability to detect routing loops. If a topology was configured incorrectly and a router failed, an error could cause packets could circulate endlessly. Among other problems, it needed an update. In 1989 an early version of BGP emerged, BGP-1, closely followed by BGP-2, then BGP-3 in 1991. However, there was yet another problem that all three BGPs shared: they only supported *classful* addressing. This meant that each network supported one of these three prefix sizes: [/24], [/16], [/8]; or 255, 65535, 16777215 hosts, respectively. If I wanted a network that could support roughly 500 hosts back in BGP-3, I'd have to opt with a prefix of [/16] (65,535 hosts). I'd be wasting over 65,000 remaining addresses.

Throughout the 1990s, BGP adopted some innovations, including IPv6, but it wasn't until 2006 that the current version of BGP (BGP-4) was released with the support of *classless* inter-domain routing (CIDR). Classless addressing gave engineers much more flexibility with prefixes – the major downside of BGP-3 – and engineers gained access to *subnets*. Even today, the internet is still running the very same BGP-4.

Network Diagram



Summary

In this lab, I configured three routing protocols: eBGP, OSPF, and EIGRP. I began by designing a topology composed of seven routers. They were to be placed in a straight line, sectioned off into three networks by the different routing protocols. I created an IP scheme composed of [*/30*] subnets, each subnet implemented on neighbor serial interfaces. These subnets ranged from [*10.10.10.0/30*] to [*10.10.10.20/30*].

A good place to start on these types of configurations is assigning the IPs to the interfaces of the routers. When the stable green port lights indicated linked ethernet, I moved to OSPF configs. As I'd learnt from my previous encounters with OSPF, network statements are the most reliable way to enable an OSPF interface. OSPF-enabled interfaces peer with neighbor OSPF interfaces and share routing information. My topology placed OSPF on the bottom half of the network, met by BGP then EIGRP on top. Since I felt familiar with EIGRP, I moved on to the top section of routers first.

EIGRP is a lot like OSPF in the configuration. Both use network statements with near indistinguishable syntax, which is all I really needed to configure for them to work. While OSPF is entirely a link-state routing protocol, EIGRP is a hybrid of link-state and distance-vector. They are used interchangeably in enterprise networks, though each have their own more specific benefits.

After the IGPs were routing within their small domains, it was time to set up BGP. Initially learning BGP was a little confusing, navigating around the *address-families* and such, but I've grown to like it more than OSPF or EIGRP. If I just wanted to configure OSPF on an interface but not broadcast that network to the routing tables, I wouldn't know where to start or if that's even possible. However, BGP has separate commands for both adjacencies and network advertisement, giving the admin more control over their network. Eventually, I got eBGP running with full redistribution between OSPF and EIGRP.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

Router(config)# **interface [*interface*] [*id*]**

- Enables configuration on a specific interface

// BGP

Router(config)# **router bgp [*autonomous system number*]**

- Activates a BGP router and enters BGP router configuration mode

The autonomous system number (ASN) is a number that identifies a large collection of routers on the internet. Typically, there are networks run under an ASN by a technical administration. eBGP connects different autonomous systems while iBGP is run within each ASN.

Router(config-router)# **address-family [*protocol*]**

- Enters configuration mode for a BGP address family

I like to think of address-families as workspaces for certain IP protocols. For example, one might enter the "ipv4" or "ipv6" address-families to configure IP routing. This is where redistribution, network statements or activation commands occur.

Router(config-router)# **neighbor [*ip address*] remote-as [*neighbor's asn*]**

- Used in forming BGP neighbor adjacencies

This command takes an IP address of a neighbor's interface and the ASN of the neighbor. For a BGP neighborship to be established, each router must have routes to the neighbor's IP *and the* correct IP and ASN of their neighbor.

Router(config-router)# **network [*network address*] mask [*subnet mask*]**

- Advertises a directly connected network to the BGP routing table

BGP's network statements are not to be confused with OSPF or EIGRPs; they aren't used to form adjacencies between BGP routers. A BGP network statement is configured alongside a neighbor statement, the former advertising the network and the latter the neighbor establishment.

// OSPF

Router(config)# **router ospf [*process id*]**

- Enables the OSPF routing protocol and enters OSPF router configuration mode

Generally, OSPF process ids should be the same, though OSPF should still form adjacencies with different process ids. Each OSPF process retains a different routing table; depending on the configuration, process ID could determine what routes are redistributed. A router can run multiple OSPF processes but will contain a separate OSPF database per process.

Router(config-router)# **router-id [*router id*]**

- Uniquely determines an OSPF router within a domain

Router ids are automatically determined by the highest loopback interface if they are not manually defined. Router ids can play a part in DR/BDR elections.

Router(config-router)# **network [*network address*] [*wildcard mask*] area [*area number*]**

- Advertises the specified subnet to neighbor routers

The defined subnet should match an interface on the router. Routers in an area share a complete topological database and have route summaries of external areas.

// EIGRP

Router(config)# **router eigrp [*instance*]**

- Enables EIGRP of a particular instance and enters router configuration mode.

There can be multiple instances of EIGRP running on a router, however, adjacent routers will only communicate if they are using the same instance.

Router(config-router)# **network [*network address*] [*wildcard mask*]**

- Advertises the specified subnet to neighbor routers

Other EIGRP routers will gain knowledge of this network and form routes to it.

// Redistribution

Router(config-router)# **redistribute [*routing protocol*] [*protocol instance*]** *<metric <value>> subnets*

- Redistributes routes from a routing protocol into another local routing protocol

The routing protocol defined will be distributed in the local router that the user is in. There are many different additional options when redistributing routes, but I've found the metric and

subnets to be the most useful. Each routing protocol has a different metric, so when redistributing be sure to use the right one. Subnets usually refers to redistributing classless networks.

Configurations

Router 1

```
R1#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R1
no ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.1 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 1::1/64
 ipv6 eigrp 10
interface Serial0/1/1
 no ip address
 clock rate 2000000
 shutdown
interface Vlan1
 no ip address
 shutdown
router eigrp 10
 eigrp router-id 1.1.1.1
 network 10.10.10.0 0.0.0.3
ipv6 router eigrp 10
 eigrp router-id 1.1.1.0
 no shutdown
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end

R1#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
```

```
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 7 subnets, 2 masks
C       10.10.10.0/30 is directly connected, Serial0/1/0
L       10.10.10.1/32 is directly connected, Serial0/1/0
D       10.10.10.4/30 [90/2681856] via 10.10.10.2, 00:03:35, Serial0/1/0
D EX    10.10.10.8/30 [170/3668480] via 10.10.10.2, 00:03:33, Serial0/1/0
D EX    10.10.10.12/30 [170/3668480] via 10.10.10.2, 00:03:33, Serial0/1/0
D EX    10.10.10.16/30 [170/3668480] via 10.10.10.2, 00:03:33, Serial0/1/0
D EX    10.10.10.20/30 [170/3668480] via 10.10.10.2, 00:03:21, Serial0/1/0
```

Router 2

```
R2#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R2
ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.5 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 2::1/64
 ipv6 eigrp 10
interface Serial0/1/1
 ip address 10.10.10.2 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 1::2/64
 ipv6 eigrp 10
 clock rate 2000000
interface Vlan1
 no ip address
 shutdown
router eigrp 10
 eigrp router-id 2.2.2.2
 network 10.10.10.4 0.0.0.3
 network 10.10.10.0 0.0.0.3
ipv6 router eigrp 10
 eigrp router-id 2.2.2.0
 no shutdown
ip classless
ip flow-export version 9
line con 0
line aux 0
```

```
line vty 0 4
 login
end
```

```
R2#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 8 subnets, 2 masks
C       10.10.10.0/30 is directly connected, Serial0/1/1
L       10.10.10.2/32 is directly connected, Serial0/1/1
C       10.10.10.4/30 is directly connected, Serial0/1/0
L       10.10.10.5/32 is directly connected, Serial0/1/0
D EX    10.10.10.8/30 [170/3156480] via 10.10.10.6, 00:07:19, Serial0/1/0
D EX    10.10.10.12/30 [170/3156480] via 10.10.10.6, 00:07:19, Serial0/1/0
D EX    10.10.10.16/30 [170/3156480] via 10.10.10.6, 00:07:19, Serial0/1/0
D EX    10.10.10.20/30 [170/3156480] via 10.10.10.6, 00:07:07, Serial0/1/0
```

Router 3

```
R3#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R3
ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.9 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 3::1/64
interface Serial0/1/1
 ip address 10.10.10.6 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 2::2/64
 ipv6 eigrp 10
 clock rate 2000000
interface Vlan1
 no ip address
 shutdown
router eigrp 10
 eigrp router-id 3.3.3.3
 redistribute bgp 10 metric 1000 33 255 1 1500
 network 10.10.10.4 0.0.0.3
router bgp 10
```

```
 bgp log-neighbor-changes
 no synchronization
 neighbor 10.10.10.10 remote-as 20
 network 10.10.10.4 mask 255.255.255.252
 redistribute eigrp 10
ipv6 router eigrp 10
 eigrp router-id 3.3.3.0
 no shutdown
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

R3#sh ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 8 subnets, 2 masks
D       10.10.10.0/30 [90/2681856] via 10.10.10.5, 00:09:01, Serial0/1/1
C       10.10.10.4/30 is directly connected, Serial0/1/1
L       10.10.10.6/32 is directly connected, Serial0/1/1
C       10.10.10.8/30 is directly connected, Serial0/1/0
L       10.10.10.9/32 is directly connected, Serial0/1/0
B       10.10.10.12/30 [20/0] via 10.10.10.10, 00:00:00
B       10.10.10.16/30 [20/0] via 10.10.10.10, 00:00:00
B       10.10.10.20/30 [20/0] via 10.10.10.10, 00:00:00
```

Router 4

R4#sh run

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R4
ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.13 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 4::1/64
interface Serial0/1/1
 ip address 10.10.10.10 255.255.255.252
```

```
 ipv6 address FE80::2 link-local
 ipv6 address 3::2/64
 clock rate 2000000
interface Vlan1
 no ip address
 shutdown
router bgp 20
 bgp log-neighbor-changes
 no synchronization
 neighbor 10.10.10.9 remote-as 10
 neighbor 10.10.10.14 remote-as 30
 network 10.10.10.8 mask 255.255.255.252
 network 10.10.10.12 mask 255.255.255.252
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

R4#sh ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 8 subnets, 2 masks
B       10.10.10.0/30 [20/2681856] via 10.10.10.9, 00:00:00
B       10.10.10.4/30 [20/0] via 10.10.10.9, 00:00:00
C       10.10.10.8/30 is directly connected, Serial0/1/1
L       10.10.10.10/32 is directly connected, Serial0/1/1
C       10.10.10.12/30 is directly connected, Serial0/1/0
L       10.10.10.13/32 is directly connected, Serial0/1/0
B       10.10.10.16/30 [20/0] via 10.10.10.14, 00:00:00
B       10.10.10.20/30 [20/128] via 10.10.10.14, 00:00:00
```

Router 5

R5#sh run

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R5
ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
```

```
 ip address 10.10.10.17 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 5::1/64
 ipv6 ospf 10 area 0
interface Serial0/1/1
 ip address 10.10.10.14 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 4::2/64
 clock rate 2000000
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 5.5.5.5
 log-adjacency-changes
 redistribute bgp 30 subnets
 network 10.10.10.16 0.0.0.3 area 0
router bgp 30
 bgp log-neighbor-changes
 no synchronization
 neighbor 10.10.10.13 remote-as 20
 network 10.10.10.16 mask 255.255.255.252
 redistribute ospf 10
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

R5#sh ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 8 subnets, 2 masks
B       10.10.10.0/30 [20/0] via 10.10.10.13, 00:00:00
B       10.10.10.4/30 [20/0] via 10.10.10.13, 00:00:00
B       10.10.10.8/30 [20/0] via 10.10.10.13, 00:00:00
C       10.10.10.12/30 is directly connected, Serial0/1/1
L       10.10.10.14/32 is directly connected, Serial0/1/1
C       10.10.10.16/30 is directly connected, Serial0/1/0
L       10.10.10.17/32 is directly connected, Serial0/1/0
O       10.10.10.20/30 [110/128] via 10.10.10.18, 00:14:33, Serial0/1/0
```

Router 6

R6#sh run

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R6
ip cef
ipv6 unicast-routing
```

```
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 ip address 10.10.10.21 255.255.255.252
 ipv6 address FE80::1 link-local
 ipv6 address 6::1/64
 ipv6 ospf 10 area 0
interface Serial0/1/1
 ip address 10.10.10.18 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 5::2/64
 ipv6 ospf 10 area 0
 clock rate 2000000
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 6.6.6.6
 log-adjacency-changes
 network 10.10.10.16 0.0.0.3 area 0
 network 10.10.10.20 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end
```

```
R6#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 8 subnets, 2 masks
O E2    10.10.10.0/30 [110/20] via 10.10.10.17, 00:19:45, Serial0/1/1
O E2    10.10.10.4/30 [110/20] via 10.10.10.17, 00:19:45, Serial0/1/1
O E2    10.10.10.8/30 [110/20] via 10.10.10.17, 00:19:45, Serial0/1/1
O E2    10.10.10.12/30 [110/20] via 10.10.10.17, 00:19:45, Serial0/1/1
C       10.10.10.16/30 is directly connected, Serial0/1/1
L       10.10.10.18/32 is directly connected, Serial0/1/1
C       10.10.10.20/30 is directly connected, Serial0/1/0
L       10.10.10.21/32 is directly connected, Serial0/1/0
```

Router 7

```
R7#sh run
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
hostname R7
ip cef
ipv6 unicast-routing
no ipv6 cef
spanning-tree mode pvst
interface GigabitEthernet0/0/0
 no ip address
 duplex auto
 speed auto
 shutdown
interface GigabitEthernet0/0/1
 no ip address
 duplex auto
 speed auto
 shutdown
interface Serial0/1/0
 no ip address
 clock rate 2000000
 shutdown
interface Serial0/1/1
 ip address 10.10.10.22 255.255.255.252
 ipv6 address FE80::2 link-local
 ipv6 address 6::2/64
 ipv6 ospf 10 area 0
 clock rate 2000000
interface Vlan1
 no ip address
 shutdown
router ospf 10
 router-id 7.7.7.7
 log-adjacency-changes
 network 10.10.10.20 0.0.0.3 area 0
ipv6 router ospf 10
 log-adjacency-changes
ip classless
ip flow-export version 9
line con 0
line aux 0
line vty 0 4
 login
end

R7#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 7 subnets, 2 masks
O E2    10.10.10.0/30 [110/20] via 10.10.10.21, 00:21:27, Serial0/1/1
O E2    10.10.10.4/30 [110/20] via 10.10.10.21, 00:21:27, Serial0/1/1
O E2    10.10.10.8/30 [110/20] via 10.10.10.21, 00:21:27, Serial0/1/1
O E2    10.10.10.12/30 [110/20] via 10.10.10.21, 00:21:27, Serial0/1/1
O       10.10.10.16/30 [110/128] via 10.10.10.21, 00:21:27, Serial0/1/1
C       10.10.10.20/30 is directly connected, Serial0/1/1
L       10.10.10.22/32 is directly connected, Serial0/1/1
```

Problems

As BGP was the primary focus, there was the classic problem of deciphering its functionality and how to implement it. Due to my lack of experience, I left BGP to configure last since I was more confident with OSPF and EIGRP. After I set up the Interior Gateway Protocols and established connectivity within both networks, my last step was to bridge them with a BGP network. So how do you configure BGP?

The first useful article I found, *configure eBGP in Cisco IOS Router by Arranda Saputra*, gave valuable insight on the main commands needed to set up BGP along with some example configurations of eBGP. From this article, I recognized my network required three different autonomous systems running on each of my BGP routers. Each autonomous system needed a unique ASN because the network was connected through eBGP.

As I may have mentioned before, the difference between BGP and IGPs is that BGP requires *neighbor* statements to form adjacencies. These neighbor statements are often accompanied by *network* statements that tell the BGP network to add a specific subnet to the routing table. Afterwards, BGP exchanges these new routes through the adjacent links to build a complete routing table.

For a BGP adjacency to be established, each router must have *routes to the IP of the desired neighbor* and the *ASN of that neighbor*. Having proper routes to each neighbor's IP is critical to forming adjacencies and is already guaranteed for directly connected routers, but this also means two BGP neighbors could lie anywhere. For example, routers *A* and *C* are connected via router *B*. Theoretically, you could establish a BGP neighbor relationship between routers *A* and *C* if they both have routes to each other's IPs.

Thankfully, after assigning the correct network and neighbor statements, my BGP network was fully functional. Once I redistributed all the routes, every router gained a full table.

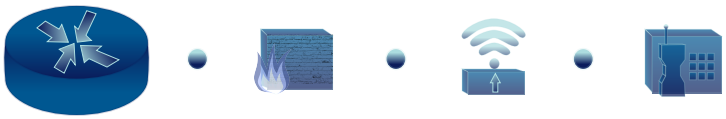
Conclusion

It always confused me how the vast internet could possibly run. If you asked me before this lab, I would have said something along the lines of "the internet is a mesh of different routing protocols like EIGRP, OSPF, RIP and BGP". I didn't have the greatest concept of interior vs exterior routing protocols, so I sort of lumped them all together.

While it isn't necessarily wrong, a more accurate answer would be that BGP is the main contributor to the internet. Like a metropolitan city, BGP is the infrastructure that supports each district, building and street of the internet. Then, within these smaller areas, you might find OSPF or EIGRP.

IPv6 eBGP

Routing IPv6 Traffic with BGP in GNS3



Purpose

The networking infrastructure of the world is supported by BGP. If it were not for BGP, none of us would be able to google "sushi places near me" and get convenient responses out of nowhere in mere milliseconds. In this paper, I go over my process of configuring BGP and provide some background information along the way. Since IPv6 is becoming more prominent, it seemed reasonable to try and configure BGP in IPv6. This lab was also a great introduction to GNS3— freelance software capable of emulating real cisco device operating systems.

Background Information

Routing

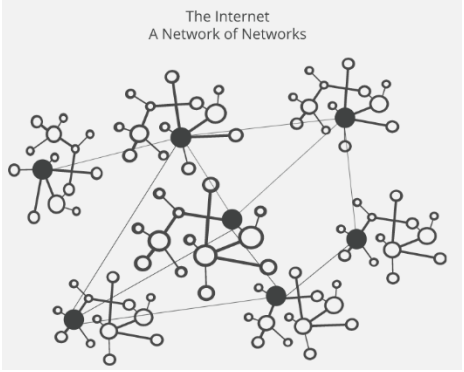
Routing is a significant process in networking as it allows hosts on different IP networks to connect to each other. *Border Gateway Protocol* (BGP) is a routing protocol simplifying the process of creating routes by using algorithms to figure out the directions automatically. In networking, routes are ultimately just *directions* for packets.

There are two options when dealing with traffic on a network; you can configure *static routes*, or you can set up a *routing protocol*. I like to think of static routes as absolute directions drawn onto a map, set in stone and unchangeable. The map can't be altered unless it is manually redrawn. If you were to follow the map, you might find some of the routes to be outdated.

It would be nice if routes were adaptable, if they could update based on the fastest paths available. This is the difference between *static routing* and *routing protocols*. Routing protocols update their routing directions automatically based on information sent from neighbors. This is the magic of routing protocols: automatic updates and directions – like google maps – for packets.

What is BGP?

Border Gateway Protocol is the most popular external routing protocol, commonly used by ISPs (Internet Service Providers) to route customer traffic. Without BGP, the internet would not function nearly as well, if at all. Think of BGP as the postal service that delivers a letter to the recipient in the fastest and most efficient manner possible. When someone submits data across the internet, BGP is responsible for choosing the best path out of all preexisting available paths, which usually means passing through autonomous systems.



An example of Autonomous systems and their local networks

So, what are autonomous systems? *Autonomous Systems* (AS) are a collection of routers, each with their own lesser hierarchy of routers that eventually connects to local networks. Each autonomous system is aware of other autonomous system(s) and can broadly determine where to route traffic based on which autonomous system holds the desired destination. ASes typically belong to ISPs (Internet service providers) or other large high-tech organizations, such as tech companies, universities, or scientific institutions. The internet is run under a collection of autonomous systems.

Kingdom Analogy

I suppose one could think of an autonomous system as a form of kingdom. Each kingdom has a ruler that dictates certain policies that the underlying citizens and infrastructure abide to. For example, if a kingdom is landlocked, it likely has a high demand for fish and salt. Therefore, a *policy* is implemented where all traders from the nearest port town have free access to and from the kingdom. Different autonomous systems often have these unique routing *policies*.

There are many paths and roads in the kingdom internally, so much so that if one goes down, alternate routes are readily available. Some kingdoms have routes bridging them, but often a traveler (packet) will have to journey through multiple kingdoms to reach their desired destination. In other words, a packet may have to pass through multiple ASes to reach its destination.

Each AS is assigned a unique, 32-bit number, the *Autonomous System Number* (ASN). These numbers differentiate what "kingdom" a router falls under. Routers with the same AS are part of the same kingdom.

Internal and External BGP

As I vaguely covered in my analogy, there are two types of BGP: *internal BGP* (within kingdoms) and *external BGP* (between kingdoms).

External BGP (eBGP) is the bridge that connects autonomous systems, where neighbors can broadly exchange network prefixes to learn more about each other's networks.

Internal BGP (iBGP) is a TCP based protocol to help advertise and support eBGP routes. The kicker: iBGP alone does not do any routing. To route, one needs an IP based protocol. So why bother with iBGP at all?

Consider an old, flimsy wooden bridge. Driving a cargo truck across would collapse the bridge. But now, with iBGP, that bridge is reinforced with a concrete foundation, metal bearings, and arches to brace the heavy loads. BGP is the only protocol designed to support the hundreds of thousands of routes that make up the internet. As of writing this, the size of the full IPv4 BGP routing table is around 800,000 prefixes without even accounting for IPv6. For reference, the average OSPF router would suffer at around 6000 prefixes. Therefore, iBGP is often used in conjunction with an IGP; the IGP does the local routing whilst iBGP contains the major routing table.

Both internal and external BGP sessions establish neighbors based on a peering system. You define a peer with a neighbor statement: for example, *neighbor 10.0.0.1 remote-as 100* states that there is a router connected, *10.0.0.1* running under ASN *100*. The neighbor *10.0.0.1* would need to define this router as a neighbor for a complete peer adjacency to form. Once both routers point to each other, they are peered. Networks are advertised with network statements: for example, *network 10.0.0.0 mask 255.255.255.0* will add the prefix *10.0.0.0/24* to the routing table. Other routers will direct traffic for *10.0.0.0/24* towards the router with the network statement.

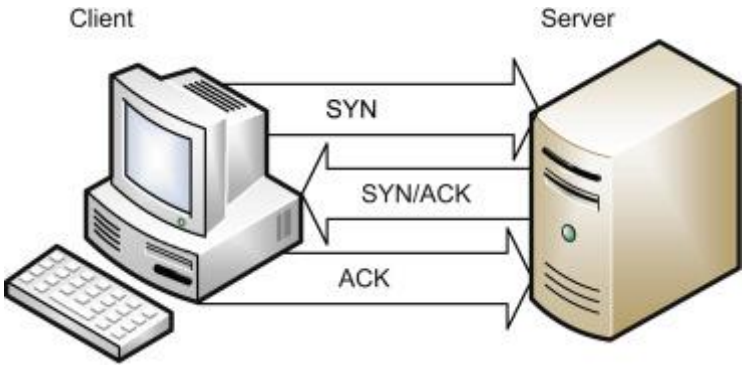
How does BGP function?

The main purpose BGP serves is forwarding traffic to an external network in the most efficient manner possible. Some factors that determine the best path are:

1. The path with the highest *weight*. This is a user defined variable.
2. The path with the highest *LOCAL_PREF*. Local preference determines which path is preferred when leaving a local AS.
3. The path with the highest *AS_PATH*. The main purpose of *AS_PATH* is to prevent infinite routing table updates. It is rather complicated, but essentially if a router goes down in a network, then this might cause the other routers to falsely change their paths, resulting in an infinite loop of changing paths. This can only happen in a distance-vector routing protocol such as BGP or RIP.
4. Favoring *eBGP* paths over iBGP paths.

BGP is a *distance-vector* routing protocol. Distance-vector routing protocols work by advertising routing tables to their neighbors. If the routes from the neighbor are better than the ones they currently have, the router will update its routing table to the preferable routes. Like all other routing protocols, BGP must first establish a neighbor adjacency with another BGP router to be able to exchange routing information. Unlike other routing protocols, BGP does not broadcast or multicast to discover other BGP neighbors. Neighbor relationships must be established manually and BGP uses TCP port 179 for the connection. There are a couple of different states BGP routers may encounter when becoming neighbors:

1. Idle. In Idle, BGP waits for a "start event". This could be when a new BGP neighbor is configured or when a reset occurs between peers that already had a connection. After the start event, BGP will initialize a TCP connection with the remote neighbor and initialize some functions. In success, BGP moves to the *Connect* state, while in failure, BGP remains in the *Idle* state.
2. Connect. In *Connect*, BGP waits for the TCP three-way handshake to complete. Both sides need to *synchronize* (SYN) and *acknowledge* (ACK) each other in a TCP three-way handshake. If the results are successful, BGP advances to the *OpenSent* state. If the results are unsuccessful, BGP begins the *Active* state.



3. Active. BGP will try another TCP three-way handshake to establish a connection with the remote neighbor. On success, BGP will transition to the *OpenSent* state. After a certain amount of time has passed with no success, BGP will revert to the *Connect* state.
4. OpenSent. BGP will wait for an "open message" from the remote neighbor. Open messages contain information about the BGP router, such as version, ASN, BGP router ID, and hold time. If the versions or hold times mismatch, BGP reverts to the *Idle* state. The ASN determines whether the BGP session will be running iBGP or eBGP. If the TCP session ever fails, BGP will revert to the *Active* state. If all passes, BGP will start sending keepalive messages to maintain the TCP session.
5. OpenConfirm. BGP waits for a keepalive message from the remote BGP neighbor. When keepalive messages are consistently received, BGP moves to the *Established* state. In any other case, BGP falls back to the *Idle* state.
6. Established. The neighbor adjacency has been formed. As long as keepalive messages are sent, the neighborship remains up. Otherwise, BGP resets back to Idle state.

Adjacencies are often formed by defining the *directly connected* interface as a neighbor, a common trait in most routing protocols. However, a technique when working with BGP is to use loopback interfaces as neighbors. Using loopbacks is common for iBGP but it also works with eBGP. Loopbacks are preferred because of redundancy: if the physical interface goes down, perhaps due to hardware, loopback interfaces will stay up since they are *virtual*.

Brief History of BGP

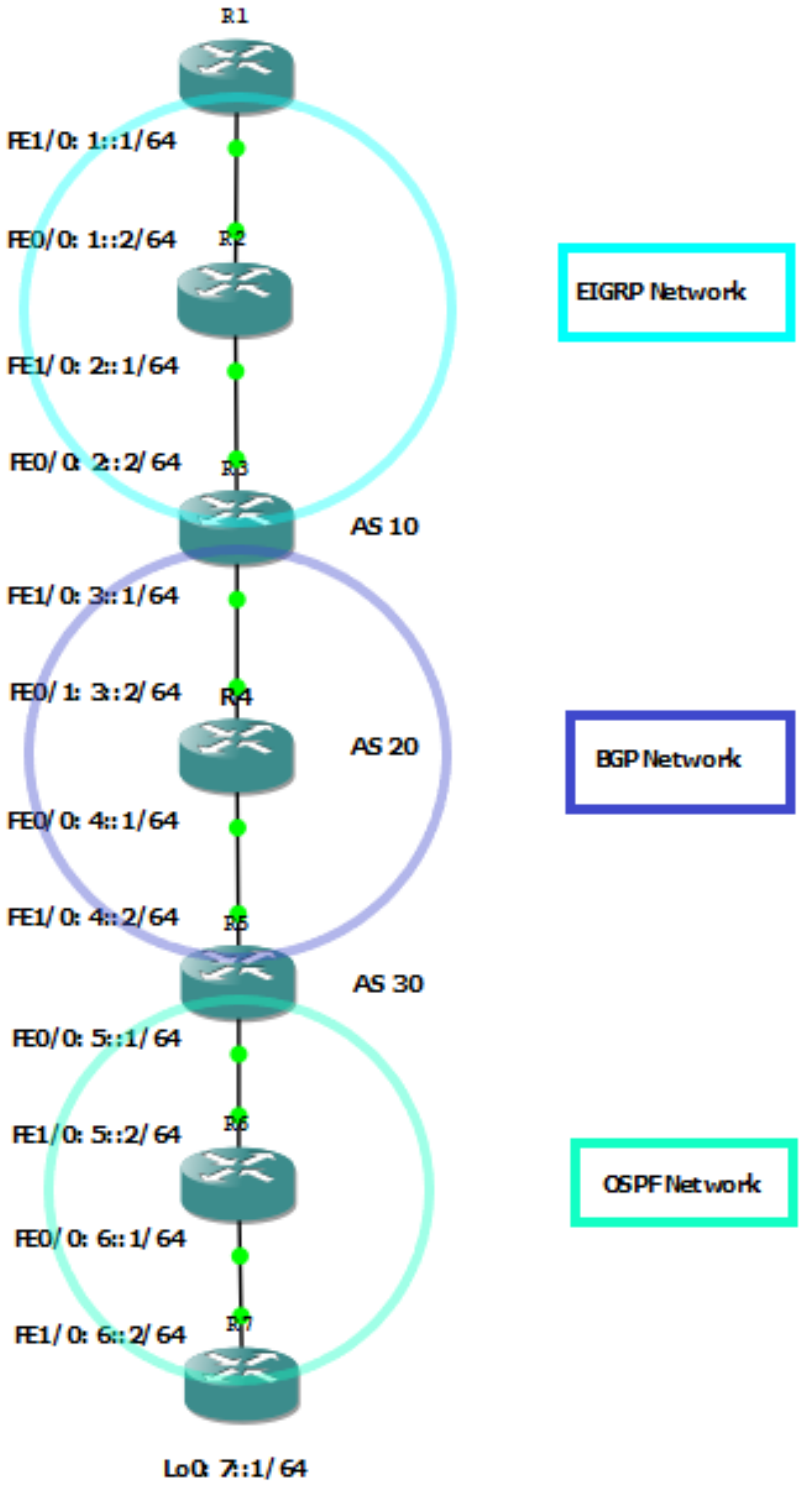
In the beginnings of the internet there was no BGP. Instead, there was *Gateway Gateway Protocol* (GGP), a brilliantly named protocol and a slightly more advanced version of an outdated protocol we have today – *Router Information Protocol* (RIP). Back then, routers were known as gateways, explaining the redundancy in the name.

Gateway Gateway Protocol was replaced by *Exterior Gateway Protocol* (EGP) in 1984, created to support the growing network infrastructure around the world. EGP introduced the concept of *autonomous systems,* which later became a big part of BGP.

However, engineers foresaw a fundamental problem with EGP: the inability to detect routing loops. If a topology was configured incorrectly and a router failed, an error could cause packets could circulate endlessly. Among other problems, it needed an update. In 1989 an early version of BGP emerged, BGP-1, closely followed by BGP-2, then BGP-3 in 1991. However, there was yet another problem that all three BGPs shared: they only supported *classful* addressing. This meant that each network supported one of these three prefix sizes: [/24], [/16], [/8]; or 255, 65535, 16777215 hosts, respectively. If I wanted a network that could support roughly 500 hosts back in BGP-3, I'd have to opt with a prefix of [/16] (65,535 hosts). I'd be wasting over 65,000 remaining addresses.

Throughout the 1990s, BGP adopted some innovations, including IPv6, but it wasn't until 2006 that the current version of BGP (BGP-4) was released with the support of *classless* inter-domain routing (CIDR). Classless addressing gave engineers much more flexibility with prefixes – the major downside of BGP-3 – and engineers gained access to *subnets*. Even today, the internet is still running the very same BGP-4.

Network Diagram



Summary

In this lab, I configured three routing protocols: eBGP, OSPF, and EIGRP. I began by designing a topology composed of seven routers. They were to be placed in a straight line, sectioned off into three networks by the different routing protocols. For those who have read my previous writeup, you may recognize this topology because it is the same. The difference here is the support of IPv6 routing configured in GNS3.

A good place to start on these types of configurations is assigning the IPs to the interfaces of the routers. When the stable green port lights indicated linked ethernet, I moved to OSPF configs. As I'd learnt from my previous encounters with OSPF, network statements are the most reliable way to enable an OSPF interface. OSPF-enabled interfaces peer with neighbor OSPF interfaces and share routing information. However, IPv6 is different. In OSPFv3 (OSPF, but with support for IPv6), network statements do not exist. They are replaced by extra configurations on an interface. For example, if I wanted my *gigabitethernet* interface to run OSPFv3, I'd need *ipv6 ospf [#] area [#]* written in that interface, instead of the *router* interface. My topology placed OSPF on the bottom half of the network, met by BGP in the middle, then EIGRP on top. Though I configured it before, I ran into some problems with forming OSPFv3 adjacencies. After finishing up OSPFv3, I moved to EIGRP.

EIGRPv6 is a lot like OSPFv3 regarding the configuration. Both use what I like to call *interface commands,* commands written in interface-configuration mode. While OSPF is entirely a link-state routing protocol, EIGRP is a hybrid of link-state and distance-vector. They are used interchangeably in enterprise networks, though each have their specific benefits.

After the *Interior Gateway Protocols* were routing within their small domains, it was time to set up IPv6 BGP. Navigating around BGP's the *address-families* was a little confusing, but I grew to appreciate the versatility that BGP brought compared to OSPF or EIGRP at the cost of more complexity. For example, enabling OSPF on an interface, but not broadcasting the network to the routing tables is not possible in OSPF because OSPF uses the same command for both features. However, BGP has separate commands for both adjacencies and network advertisement, giving the admin more control over their network. Eventually, I got IPv6 running in BGP with full redistribution between OSPF and EIGRP networks.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.

Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

Router(config)# **interface [*interface*] [*id*]**

- Enables configuration on a specific interface

// BGP

Router(config)# **router bgp [*autonomous system number*]**

- Activates a BGP router and enters BGP router configuration mode

The autonomous system number (ASN) is a number that identifies a large collection of routers on the internet. Typically, there are networks run under an ASN by a technical administration. eBGP connects different autonomous systems while iBGP is run within each ASN.

Router(config-router)# **address-family [*protocol*]**

- Enters configuration mode for a BGP address family

I like to think of address-families as workspaces for certain IP protocols. For example, one might enter the "ipv4" or "ipv6" address-families to configure IP routing. This is where redistribution, network statements or activation commands occur.

Router(config-router)# **neighbor [*ipv6 address*] remote-as [*neighbor's asn*]**

- Used in forming BGP neighbor adjacencies

This command takes an IP address of a neighbor's interface and the ASN of the neighbor. For a BGP neighborship to be established, each router must have routes to the neighbor's IP *and the* ASN of their neighbor.

Router(config-router-af)# **network [*ipv6 network address*] mask [*subnet mask*]**

- Advertises a directly connected network to the BGP routing table

BGP's network statements are not to be confused with OSPF or EIGRPs; they aren't used to form adjacencies between BGP routers. A BGP network statement is configured alongside a neighbor statement, the former advertising the network and the latter the neighbor establishment. Example: network 1::/64.

Router(config-router-af)# **neighbor [*ipv6 address*] activate**

- Enables a BGP neighbor

// OSPFv3

Router(config)# **ipv6 router ospf [*process id*]**

- Enables the OSPFv3 routing protocol and enters OSPFv3 router configuration mode

Generally, OSPF process ids should be the same, though OSPF should still form adjacencies with different process ids. Each OSPF process retains a different routing table; depending on the configuration, process ID could determine what routes are redistributed. A router can run multiple OSPF processes but will contain a separate OSPF database per process.

Router(config-router)# **router-id [*router id*]**

- Uniquely determines an OSPF router within a domain

Router ids are automatically determined by the highest loopback interface if they are not manually defined. Router ids can play a part in DR/BDR elections.

Router(config-if)# **ipv6 ospf [*process id*] area [*area number*]**

- Advertises the specified subnet to neighbor routers

Configured in interface-configuration mode. The defined subnet should match an interface on the router. Routers in an area share a complete topological database and have route summaries of external areas.

// EIGRPv6

Router(config)# **ipv6 router eigrp [*instance*]**

- Enables EIGRP of a particular instance and enters router configuration mode.

There can be multiple instances of EIGRP running on a router, however, adjacent routers will only communicate if they are using the same instance.

Router(config-if)# **ipv6 eigrp [*instance*]**

- Enables EIGRPv6 on the current interface

The network on the interface will be broadcast to adjacent EIGRPv6 routers.

// Redistribution

Router(config-router)# **redistribute [*routing protocol*] [*protocol instance*]** <*metric <value>>* *subnets*

- Redistributes routes from a routing protocol into another local routing protocol

The routing protocol defined will be distributed in the local router that the user is in. There are many different additional options when redistributing routes, but I've found the metric and subnets to be the most useful. Each routing protocol has a different metric, so when redistributing be sure to use the right one. Subnets usually refers to redistributing classless networks.

Configurations

Router 1

```
R1#show running-config
no service timestamps debug uptime
no service timestamps log uptime
hostname R1
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
no ip cef
no ip domain lookup
ipv6 unicast-routing
no ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface FastEthernet0/0
 no ip address
 shutdown
 duplex full
interface FastEthernet1/0
 ip address 10.10.10.1 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::1 link-local
 ipv6 address 1::1/64
 ipv6 eigrp 10
interface FastEthernet1/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/0
 no ip address
 shutdown
```

```
 speed auto
 duplex auto
interface FastEthernet2/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet3/0
 no ip address
 shutdown
 duplex full
interface FastEthernet4/0
 no ip address
 shutdown
 duplex full
router eigrp 10
 network 10.10.10.0 0.0.0.3
 eigrp router-id 1.1.1.1
ip forward-protocol nd
ip flow-export version 9
no ip http server
no ip http secure-server
ipv6 router eigrp 10
 eigrp router-id 1.1.1.0
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

```
R1#show ipv6 route
IPv6 Routing Table - default - 9 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
C   1::/64 [0/0]
     via FastEthernet1/0, directly connected
L   1::1/128 [0/0]
     via FastEthernet1/0, receive
D   2::/64 [90/30720]
     via FE80::2, FastEthernet1/0
EX  3::/64 [170/33280]
     via FE80::2, FastEthernet1/0
EX  4::/64 [170/2573568]
     via FE80::2, FastEthernet1/0
EX  5::/64 [170/2573568]
     via FE80::2, FastEthernet1/0
EX  6::/64 [170/2573568]
     via FE80::2, FastEthernet1/0
EX  7::1/128 [170/2573568]
     via FE80::2, FastEthernet1/0
```

```
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 2

```
R2#show running-config
no service timestamps debug uptime
no service timestamps log uptime
hostname R2
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
no ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface FastEthernet0/0
 ip address 10.10.10.2 255.255.255.252
 duplex full
 ipv6 address FE80::2 link-local
 ipv6 address 1::2/64
 ipv6 eigrp 10

interface FastEthernet1/0
 ip address 10.10.10.5 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::1 link-local
 ipv6 address 2::1/64
 ipv6 eigrp 10
interface FastEthernet1/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/0
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet3/0
 no ip address
 shutdown
 duplex full
interface FastEthernet4/0
 no ip address
 shutdown
 duplex full
router eigrp 10
 network 10.10.10.0 0.0.0.3
 network 10.10.10.4 0.0.0.3
 eigrp router-id 2.2.2.2
ip forward-protocol nd
```

```
ip flow-export version 9
no ip http server
no ip http secure-server
ipv6 router eigrp 10
 eigrp router-id 2.2.2.0
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

R2#show ipv6 route

```
IPv6 Routing Table - default - 10 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
C   1::/64 [0/0]
     via FastEthernet0/0, directly connected
L   1::2/128 [0/0]
     via FastEthernet0/0, receive
C   2::/64 [0/0]
     via FastEthernet1/0, directly connected
L   2::1/128 [0/0]
     via FastEthernet1/0, receive
EX  3::/64 [170/30720]
     via FE80::2, FastEthernet1/0
EX  4::/64 [170/2571008]
     via FE80::2, FastEthernet1/0
EX  5::/64 [170/2571008]
     via FE80::2, FastEthernet1/0
EX  6::/64 [170/2571008]
     via FE80::2, FastEthernet1/0
EX  7::1/128 [170/2571008]
     via FE80::2, FastEthernet1/0
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 3

R3#show running-config

```
no service timestamps debug uptime
no service timestamps log uptime
hostname R3
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
```

```
ipv6 unicast-routing
no ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface FastEthernet0/0
 ip address 10.10.10.6 255.255.255.252
 duplex full
 ipv6 address FE80::2 link-local
 ipv6 address 2::2/64
 ipv6 enable
 ipv6 eigrp 10

interface FastEthernet1/0
 ip address 10.10.10.9 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::1 link-local
 ipv6 address 3::1/64
 ipv6 enable
interface FastEthernet1/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/0
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet3/0
 no ip address
 shutdown
 duplex full
interface FastEthernet4/0
 no ip address
 shutdown
 duplex full
router eigrp 10
 network 10.10.10.4 0.0.0.3
 redistribute bgp 10 metric 1000 33 255 1 1500
 eigrp router-id 3.3.3.3
router bgp 10
 bgp router-id 10.10.10.10
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 neighbor 3::2 remote-as 20
 neighbor 3::2 ebgp-multihop 255
 neighbor 10.10.10.10 remote-as 20
 address-family ipv4
  network 10.10.10.4 mask 255.255.255.252
  network 10.10.10.8 mask 255.255.255.252
  redistribute eigrp 10
  neighbor 10.10.10.10 activate
 exit-address-family
 address-family ipv6
  redistribute connected
  redistribute eigrp 10
  network 2::/64
```

```
  network 3::/64
  neighbor 3::2 activate
 exit-address-family
ip forward-protocol nd
ip flow-export version 9
no ip http server
no ip http secure-server
ipv6 router eigrp 10
 eigrp router-id 3.3.3.0
 redistribute bgp 10 metric 1000 33 255 1 1500
 redistribute connected
control-plane
no parser cache
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

R3#show ipv6 route

```
IPv6 Routing Table - default - 10 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
D   1::/64 [90/30720]
     via FE80::1, FastEthernet0/0
C   2::/64 [0/0]
     via FastEthernet0/0, directly connected
L   2::2/128 [0/0]
     via FastEthernet0/0, receive
C   3::/64 [0/0]
     via FastEthernet1/0, directly connected
L   3::1/128 [0/0]
     via FastEthernet1/0, receive
B   4::/64 [20/0]
     via 3::2
B   5::/64 [20/0]
     via 3::2
B   6::/64 [20/0]
     via 3::2
B   7::1/128 [20/0]
     via 3::2
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 4

R4#show running-config

```
no service timestamps debug uptime
no service timestamps log uptime
```

```
no service password-encryption
hostname R4
boot-start-marker
boot-end-marker
no aaa new-model
memory-size iomem 5
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ip auth-proxy max-nodata-conns 3
ip admission max-nodata-conns 3
ipv6 unicast-routing
ip tcp synwait-time 5
interface FastEthernet0/0
 ip address 10.10.10.13 255.255.255.252
 duplex auto
 speed auto
 ipv6 address 4::1/64
 ipv6 address FE80::1 link-local
 ipv6 enable
interface FastEthernet0/1
 ip address 10.10.10.10 255.255.255.252
 duplex auto
 speed auto
 ipv6 address 3::2/64
 ipv6 address FE80::2 link-local
interface FastEthernet1/0
 no ip address
 shutdown
 duplex auto
 speed auto
 ipv6 enable
interface FastEthernet2/0
 no ip address
 shutdown
 duplex auto
 speed auto
interface FastEthernet3/0
 no ip address
 shutdown
 duplex auto
 speed auto
interface FastEthernet4/0
 no ip address
 shutdown
 duplex auto
 speed auto
router bgp 20
 bgp router-id 20.20.20.20
 no bgp default ipv4-unicast
 bgp log-neighbor-changes
 neighbor 3::1 remote-as 10
 neighbor 3::1 ebgp-multihop 255
 neighbor 4::2 remote-as 30
 neighbor 4::2 ebgp-multihop 255
 neighbor 10.10.10.9 remote-as 10
 neighbor 10.10.10.14 remote-as 30
 address-family ipv4
  neighbor 3::1 activate
  neighbor 4::2 activate
  neighbor 10.10.10.9 activate
  neighbor 10.10.10.14 activate
  no auto-summary
```

```
  no synchronization
  network 10.10.10.8 mask 255.255.255.252
  network 10.10.10.12 mask 255.255.255.252
 exit-address-family
 address-family ipv6
  neighbor 3::1 activate
  neighbor 4::2 activate
  network 3::/64
  network 4::/64
  redistribute connected
  no synchronization
 exit-address-family
no ip http server
no ip http secure-server
ip forward-protocol nd
ip flow-export version 9
no cdp log mismatch duplex
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
line vty 0 4
 login
end
```

R4#show ipv6 route

```
IPv6 Routing Table - 11 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route
       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
B   1::/64 [20/30720]
     via 3::1
B   2::/64 [20/0]
     via 3::1
C   3::/64 [0/0]
     via ::, FastEthernet0/1
L   3::2/128 [0/0]
     via ::, FastEthernet0/1
C   4::/64 [0/0]
     via ::, FastEthernet0/0
L   4::1/128 [0/0]
     via ::, FastEthernet0/0
B   5::/64 [20/0]
     via 4::2
B   6::/64 [20/2]
     via 4::2
B   7::1/128 [20/2]
     via 4::2
L   FE80::/10 [0/0]
     via ::, Null0
L   FF00::/8 [0/0]
     via ::, Null0
```

Router 5

```
R5#show running-config
no service timestamps debug uptime
no service timestamps log uptime
hostname R5
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface FastEthernet0/0
 ip address 10.10.10.17 255.255.255.252
 duplex full
 ipv6 address FE80::1 link-local
 ipv6 address 5::1/64
 ipv6 ospf 10 area 0
interface FastEthernet1/0
 ip address 10.10.10.14 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::2 link-local
 ipv6 address 4::2/64
interface FastEthernet1/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/0
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet3/0
 no ip address
 shutdown
 duplex full
interface FastEthernet4/0
 no ip address
 shutdown
 duplex full
router ospf 10
 router-id 5.5.5.5
 redistribute bgp 30 subnets
 network 10.10.10.16 0.0.0.3 area 0
router bgp 30
 bgp router-id 30.30.30.30
 bgp log-neighbor-changes
```

```
 no bgp default ipv4-unicast
 neighbor 4::1 remote-as 20
 neighbor 4::1 ebgp-multihop 255
 neighbor 10.10.10.13 remote-as 20
 address-family ipv4
  network 10.10.10.16 mask 255.255.255.252
  redistribute ospf 10
  neighbor 10.10.10.13 activate
 exit-address-family
 address-family ipv6
  redistribute connected
  redistribute ospf 10 include-connected
  network 4::/64
  network 5::/64
  neighbor 4::1 activate
 exit-address-family
ip forward-protocol nd
ip flow-export version 9
no ip http server
no ip http secure-server
ipv6 router ospf 10
 router-id 5.5.5.0
 redistribute connected
 redistribute bgp 30
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

```
R5#show ipv6 route
IPv6 Routing Table - default - 10 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
B   1::/64 [20/0]
     via 4::1
B   2::/64 [20/0]
     via 4::1
B   3::/64 [20/0]
     via 4::1
C   4::/64 [0/0]
     via FastEthernet1/0, directly connected
L   4::2/128 [0/0]
     via FastEthernet1/0, receive
C   5::/64 [0/0]
```

```
     via FastEthernet0/0, directly connected
L    5::1/128 [0/0]
     via FastEthernet0/0, receive
O    6::/64 [110/2]
     via FE80::2, FastEthernet0/0
O    7::1/128 [110/2]
     via FE80::2, FastEthernet0/0
L    FF00::/8 [0/0]
     via Null0, receive
```

Router 6

R6#show running-config

```
no service timestamps debug uptime
no service timestamps log uptime
hostname R6
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface FastEthernet0/0
 ip address 10.10.10.21 255.255.255.252
 duplex full
 ipv6 address FE80::1 link-local
 ipv6 address 6::1/64
 ipv6 enable
 ipv6 ospf 10 area 0
interface FastEthernet1/0
 ip address 10.10.10.18 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::2 link-local
 ipv6 address 5::2/64
 ipv6 enable
 ipv6 ospf 10 area 0
interface FastEthernet1/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/0
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet3/0
 no ip address
 shutdown
 duplex full
interface FastEthernet4/0
```

```
 no ip address
 shutdown
 duplex full
router ospf 10
 router-id 6.6.6.6
 network 10.10.10.16 0.0.0.3 area 0
 network 10.10.10.20 0.0.0.3 area 0
ip forward-protocol nd
ip flow-export version 9
no ip http server
no ip http secure-server
ipv6 router ospf 10
 router-id 6.6.6.0
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

R6#show ipv6 route

```
IPv6 Routing Table - default - 10 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
OE2 1::/64 [110/1]
     via FE80::1, FastEthernet1/0
OE2 2::/64 [110/1]
     via FE80::1, FastEthernet1/0
OE2 3::/64 [110/1]
     via FE80::1, FastEthernet1/0
OE2 4::/64 [110/20]
     via FE80::1, FastEthernet1/0
C   5::/64 [0/0]
     via FastEthernet1/0, directly connected
L   5::2/128 [0/0]
     via FastEthernet1/0, receive
C   6::/64 [0/0]
     via FastEthernet0/0, directly connected
L   6::1/128 [0/0]
     via FastEthernet0/0, receive
O   7::1/128 [110/1]
     via FE80::2, FastEthernet0/0
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 7

R7#show running-config

```
no service timestamps debug uptime
```

```
no service timestamps log uptime
hostname R7
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface Loopback0
 no ip address
 ipv6 address 7::1/64
 ipv6 ospf 10 area 0
interface FastEthernet0/0
 no ip address
 shutdown
 duplex full
interface FastEthernet1/0
 ip address 10.10.10.22 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::2 link-local
 ipv6 address 6::2/64
 ipv6 ospf 10 area 0
interface FastEthernet1/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/0
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet2/1
 no ip address
 shutdown
 speed auto
 duplex auto
interface FastEthernet3/0
 no ip address
 shutdown
 duplex full
interface FastEthernet4/0
 no ip address
 shutdown
 duplex full
router ospf 10
 router-id 7.7.7.7
 network 10.10.10.20 0.0.0.3 area 0
ip forward-protocol nd
ip flow-export version 9
no ip http server
no ip http secure-server
ipv6 router ospf 10
 router-id 7.7.7.0
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
```

```
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

```
R7#show ipv6 route
IPv6 Routing Table - default - 10 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
OE2 1::/64 [110/1]
     via FE80::1, FastEthernet1/0
OE2 2::/64 [110/1]
     via FE80::1, FastEthernet1/0
OE2 3::/64 [110/1]
     via FE80::1, FastEthernet1/0
OE2 4::/64 [110/20]
     via FE80::1, FastEthernet1/0
O   5::/64 [110/2]
     via FE80::1, FastEthernet1/0
C   6::/64 [0/0]
     via FastEthernet1/0, directly connected
L   6::2/128 [0/0]
     via FastEthernet1/0, receive
C   7::/64 [0/0]
     via Loopback0, directly connected
L   7::1/128 [0/0]
     via Loopback0, receive
L   FF00::/8 [0/0]
     via Null0, receive
```

Problems

The first problem I encountered was setting up EIGRPv6. I assigned all the IPs on the correct interfaces and did a basic setup on every router. I assumed EIGRPv6 would be configured like OSPFv3 (something I've done in the past) by means of advertising a network in *interface-configuration mode*. Sure enough, there was a command strikingly just like in OSPFv3: *ipv6 eigrp [instance]*. After confirming with some cisco documentation, I enabled EIGRPv6 in interface-configuration mode and defined the respective EIGRP routers in IPv6. My EIGRPv6 network was active.

Next came OSPFv3 – which I configured incorrectly but did not realize it at the time – and finally BGPv6. BGPv6 was something new to me, so I had to find some resources that explained the basics. I learnt about *address-families*, which are spaces to write protocol-related configurations. For example, BGP requires the admin to enter *the IPv4 address-family* to configure IPv4 content.

There were two issues I had while configuring BGPv6: establishing neighbors and advertising IPv6 networks. I wanted to use *link-local* addresses as my neighbor interfaces. The documentation claimed it was possible by adding a "%" sign at the end of an address, for example, *neighbor fe80::2% remote-as 20*, but I got the error *"create a peer-group first"* instead. I couldn't find anything regarding peer groups, so I settled on using the global address of the interface.

Even with the proper network and neighbor statements, the routes would still not advertise. No neighbor adjacencies would form, and so no packets could travel across my BGPv6 network. Finally, I stumbled on a command that could solve my issue: *no BGP default ipv4-unicast*. By default, only IPv4 is advertised, regardless of other BGP configuration. I implemented the command on my BGP routers.

Now everything should work. BGP was working, and I was sure I had configured OSPF and EIGRP correctly. But part of my network was down; something was wrong with OSPFv3. I narrowed down the problem to reveal that OSPFv3 neighbor adjacencies were not forming. The link state databases weren't updating and my debug logs displayed *"area BACKBONE(0) (Inactive)"*. I troubleshooted the *"area BACKBONE(0) (Inactive)"* error and found others with similar issues, except these articles dated back a decade and were in IPv4. There was no solution in any on the old forums.

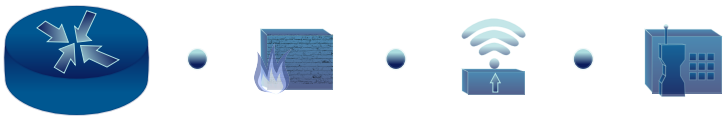
I double checked the interfaces and addresses of my routers; all were configured correctly according to the topology. I added IPv6 router ids to no avail and power cycled all the routers. There was one more trick up my sleeve: I had another cisco router IOS, the 7200 series. I was currently using the 3600s. My final attempt to fix the adjacencies would be to switch to the 7200s. I disliked this solution, as it seemed unrealistic in reality: you would have to upgrade the hardware which meant spending a lot of money. Luckily, I'm using an emulator, so I did not have to worry about the cost. I configured OSPFv3 on the 7200 series routers the same as it was on the 3600 series IOS, and it worked.

Conclusion

In this lab, I configured BGP, OSPF and EIGRP with IPv6. While it was not the main focus, I also got to learn GNS3 and virtual networking. This lab produced far more problems than I expected, but that was only natural due to the transition from packet tracer to GNS3. Many of these problems are ones I would have faced in the real world since GNS3 is an emulation program. If there is one piece of advice for anyone routing IPv6 in BGP, it would be remembering to add the *no bgp default ipv4-unicast* statement, which is basically BGP's equivalent of *ipv6 unicast-routing*.

iBGP

Implementing iBGP with Loopbacks as Neighbor Links



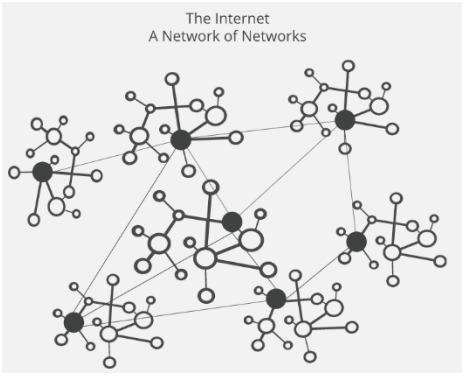
Purpose

The purpose of this lab was to configure internal BGP using the neighbor's loopback addresses as the interface to form adjacencies. Since BGP is the main contributor in the miracle that is the internet, it was useful to learn each aspect of BGP and how it played a role in contributing to the internet.

Background Information

What is BGP?

BGP (Border Gateway Protocol) is the most popular routing protocol, commonly used by ISPs (Internet Service Providers) to route customer traffic. Without BGP, the internet would not function nearly as well, if at all. Think of BGP as the postal service that delivers a letter to the recipient in the fastest and most efficient manner possible. When someone submits data across the internet, BGP is responsible for choosing the best path out of all preexisting available paths, which usually means passing through autonomous systems.



An example of Autonomous systems and their local networks

So, what are autonomous systems? Autonomous systems (AS) are a collection of routers, each with their own lesser hierarchy of routers that eventually connects to local networks. Each autonomous system is aware of other autonomous system(s) and can broadly determine where to route traffic based on which autonomous system holds the desired destination. ASes typically belong to ISPs (Internet service providers) or other large high-tech organizations, such as tech companies, universities, or scientific institutions. The internet is run under a collection of autonomous systems.

Kingdom Analogy

I suppose one could think of an autonomous system as a form of kingdom. Each kingdom has a ruler that dictates certain policies that the underlying citizens and infrastructure abide to.

For example, if a kingdom is landlocked, it likely has a high demand for fish and salt. Therefore, a *policy* is implemented where all traders from the nearest port town have free access to and from the kingdom. Different autonomous systems often have these unique routing *policies*.

There are many paths and roads in the kingdom internally, so much so that if one goes down, alternate routes are readily available. Some kingdoms have routes bridging them, but often a traveler (packet) will have to journey through multiple kingdoms to reach their desired destination. In other words, a packet may have to pass through multiple ASes to reach its destination.

Each AS is assigned a unique, 32-bit number, the *Autonomous System Number* (ASN). These numbers differentiate what "kingdom" a router falls under. Routers with the same AS are part of the same kingdom.

Internal and External BGP

As I vaguely covered in my analogy, there are two types of BGP: *internal BGP* (within kingdoms) and *external BGP* (between kingdoms).

External BGP (eBGP) is the bridge that connects autonomous systems, where neighbors can broadly exchange network prefixes to learn more about each other's networks.

Internal BGP (iBGP) is a TCP based protocol to help advertise and support eBGP routes. The kicker: iBGP alone does not do any routing. To route, one needs an IP based protocol. So why bother with iBGP at all?

Consider an old, flimsy wooden bridge. Driving a cargo truck across would collapse the bridge. But now, with iBGP, that bridge is reinforced with a concrete foundation, metal bearings, and arches to brace the heavy loads. BGP is the only protocol designed to support the hundreds of thousands of routes that make up the internet. As of writing this, the size of the full IPv4 BGP routing table is around 800,000 prefixes without even accounting for IPv6. For reference, the average OSPF router would suffer at around 6000 prefixes. Therefore, iBGP is often used in conjunction with an IGP; the IGP does the local routing whilst iBGP contains the major routing table.

Both internal and external BGP sessions establish neighbors based on a peering system. You define a peer with a neighbor statement: for example, *neighbor 10.0.0.1 remote-as 100* states that there is a router connected, *10.0.0.1* running under ASN *100*. The neighbor *10.0.0.1* would need to define this router as a neighbor for a complete peer adjacency to form. Once both routers point to each other, they are peered. Networks are advertised with network statements: for example, *network 10.0.0.0 mask 255.255.255.0* will add the prefix *10.0.0.0/24* to the routing table. Other routers will direct traffic for *10.0.0.0/24* towards the router with the network statement.

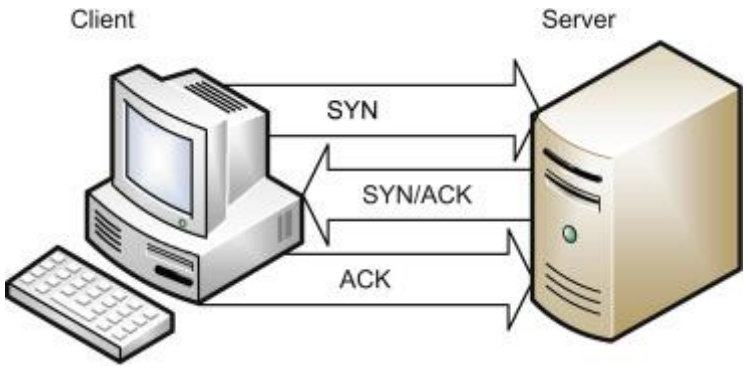
How does BGP function?

The main purpose BGP serves is forwarding traffic to an external network in the most efficient manner possible. Some factors that determine the best path are:

1. The path with the highest *weight*. This is a user defined variable.
2. The path with the highest *LOCAL_PREF*. Local preference determines which path is preferred when leaving a local AS.
3. The path with the highest *AS_PATH*. The main purpose of *AS_PATH* is to prevent infinite routing table updates. It is rather complicated, but essentially if a router goes down in a network, then this might cause the other routers to falsely change their paths, resulting in an infinite loop of changing paths. This can only happen in a distance-vector routing protocol such as BGP or RIP.
4. Favoring *eBGP* paths over iBGP paths.

BGP is a *distance-vector* routing protocol. Distance-vector routing protocols work by advertising routing tables to their neighbors. If the routes from the neighbor are better than the ones they currently have, the router will update its routing table to the preferable routes. Like all other routing protocols, BGP must first establish a neighbor adjacency with another BGP router to be able to exchange routing information. Unlike other routing protocols, BGP does not broadcast or multicast to discover other BGP neighbors. Neighbor relationships must be established manually and BGP uses TCP port 179 for the connection. There are a couple of different states BGP routers may encounter when becoming neighbors:

1. Idle. In Idle, BGP waits for a "start event". This could be when a new BGP neighbor is configured or when a reset occurs between peers that already had a connection. After the start event, BGP will initialize a TCP connection with the remote neighbor and initialize some functions. In success, BGP moves to the *Connect* state, while in failure, BGP remains in the *Idle* state.
2. Connect. In *Connect*, BGP waits for the TCP three-way handshake to complete. Both sides need to *synchronize* (SYN) and *acknowledge* (ACK) each other in a TCP three-way handshake. If the results are successful, BGP advances to the *OpenSent* state. If the results are unsuccessful, BGP begins the *Active* state.



3. Active. BGP will try another TCP three-way handshake to establish a connection with the remote neighbor. On success, BGP will transition to the *OpenSent* state. After a

certain amount of time has passed with no success, BGP will revert to the *Connect* state.

4. OpenSent. BGP will wait for an "open message" from the remote neighbor. Open messages contain information about the BGP router, such as version, ASN, BGP router ID, and hold time. If the versions or hold times mismatch, BGP reverts to the *Idle* state. The ASN determines whether the BGP session will be running iBGP or eBGP. If the TCP session ever fails, BGP will revert to the *Active* state. If all passes, BGP will start sending keepalive messages to maintain the TCP session.
5. OpenConfirm. BGP waits for a keepalive message from the remote BGP neighbor. When keepalive messages are consistently received, BGP moves to the *Established* state. In any other case, BGP falls back to the *Idle* state.
6. Established. The neighbor adjacency has been formed. As long as keepalive messages are sent, the neighborship remains up. Otherwise, BGP resets back to Idle state.

Adjacencies are often formed by defining the *directly connected* interface as a neighbor, a common trait in most routing protocols. However, a technique when working with BGP is to use loopback interfaces as neighbors. Using loopbacks is common for iBGP but it also works with eBGP. Loopbacks are preferred because of redundancy: if the physical interface goes down, perhaps due to hardware, loopback interfaces will stay up since they are *virtual*.

Brief History of BGP

In the beginnings of the internet there was no BGP. Instead, there was *Gateway Gateway Protocol* (GGP), a brilliantly named protocol and a slightly more advanced version of an outdated protocol we have today – *Router Information Protocol* (RIP). Back then, routers were known as gateways, explaining the redundancy in the name.

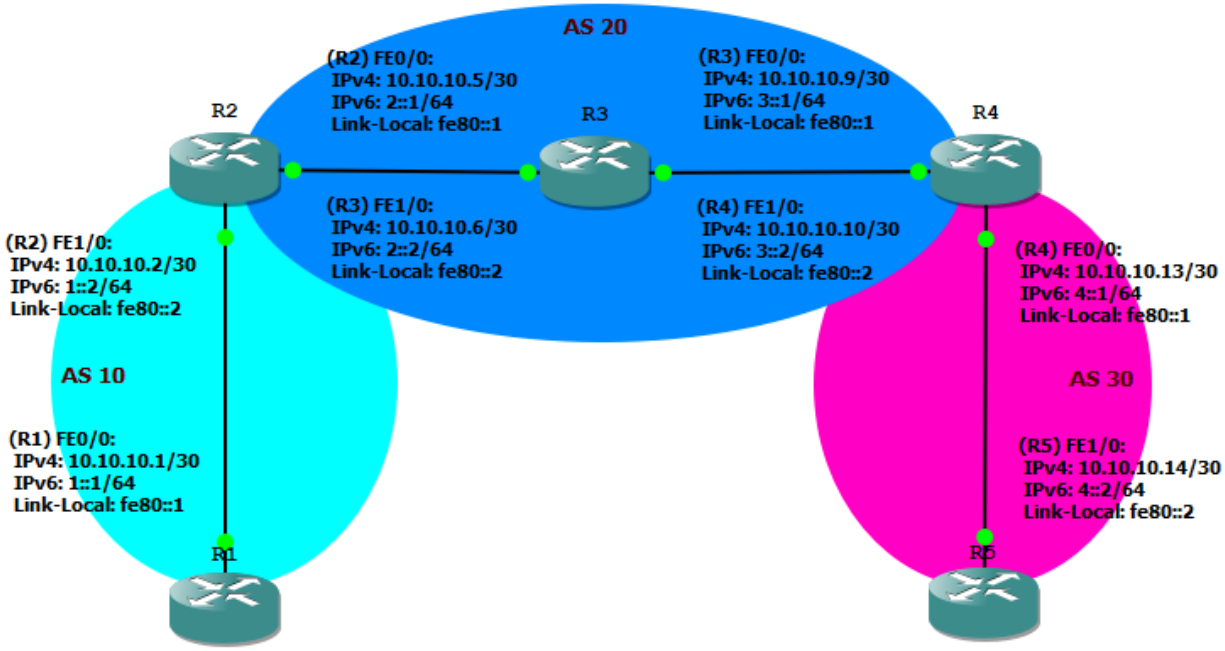
Gateway Gateway Protocol was replaced by *Exterior Gateway Protocol* (EGP) in 1984, created to support the growing network infrastructure around the world. EGP introduced the concept of *autonomous systems*, which later became a big part of BGP.

However, engineers foresaw a fundamental problem with EGP: the inability to detect routing loops. If a topology was configured incorrectly and a router failed, an error could cause packets could circulate endlessly. Among other problems, it needed an update. In 1989 an early version of BGP emerged, BGP-1, closely followed by BGP-2, then BGP-3 in 1991. However, there was yet another problem that all three BGPs shared: they only supported *classful* addressing. This meant that each network supported one of these three prefix sizes: [/24], [/16], [/8]; or 255, 65535, 16777215 hosts, respectively. If I wanted a network that could support roughly 500 hosts back in BGP-3, I'd have to opt with a prefix of [/16] (65,535 hosts). I'd be wasting over 65,000 remaining addresses.

Throughout the 1990s, BGP adopted some innovations, including IPv6, but it wasn't until 2006 that the current version of BGP (BGP-4) was released with the support of *classless* inter-domain routing (CIDR). Classless addressing gave engineers much more flexibility with prefixes – the

major downside of BGP-3 – and engineers gained access to *subnets*. Even today, the internet is still running the very same BGP-4.

Network Diagram



Summary

I began by creating a topology that would house my five virtual Cisco 7200 series routers. For each serial interface of every router, I devised an appropriate IP scheme mainly consisting of */30* subnets, chaining together the five routers. Since the purpose of this project was to use *loopback* interfaces to establish connections between the BGP peers, I configured one loopback per router. The loopbacks act as neighbor interfaces for adjacent routers. After *IPv4* was complete, I did the same for *IPv6*.

The next step was configuring the routing protocol, BGP. I decided to take a plunge in the deep end and started implementing iBGP first; having configured eBGP in previous projects, I had a fair understanding of the concepts, but had yet to touch iBGP. iBGP needs to be configured in a mesh-like topology, where every router is a neighbor with the others on the network. The problem arises when neighbors are not directly connected, for example, like router 3 and 5 in my topology. These routers are not directly connected, so, ironically, I would need to implement an *underlaying* routing protocol for iBGP to "become aware" of every router. Kind of

funny how iBGP, a routing protocol, needs the support of another routing protocol (or just static routes) to operate.

I chose to use OSPF as my "intermediary" routing protocol for iBGP. Luckily, the actual configuration for iBGP was much like eBGP, so I did not have many problems once OSPF was set up. Before I enabled the loopbacks to act as the neighbors, I wanted to test my iBGP network with the physical interfaces. I configured iBGP routing on the internal portion of the network using physical interfaces as my neighbor interfaces. I could ping throughout the iBGP network, inspiring confidence that the loopbacks would work once I made the transition.

After fixing a conflicting addressing problem caused by my loopback addresses and reworking the IP scheme, I began switching the iBGP neighbors to the loopback interfaces. Once iBGP was configured with the loopback interfaces, I moved to enabling the eBGP part of the network. I used static routing so the eBGP loopback interfaces were "aware" of each other instead of a routing protocol like with the iBGP network. With iBGP and eBGP pinging correctly in *IPv4*, I did the same for *IPv6*.

Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

Router(config)# **interface [*type*] [*interface number*]**

- Enter interface configuration mode

Router(config)# **router bgp [*autonomous system number*]**

- Enter router configuration mode for BGP

The ASN specifies the autonomous system of the router. This will determine whether the router is part of an iBGP or eBGP network.

Router(config-router)# **no bgp default ipv4-unicast**

- Enables advertising for IPv6 routes

By default, BGP only enables the IPv4 address-family.

Router(config-router)# **neighbor [*ip address*] remote-as [*neighbor's asn*]**

- Used in forming BGP neighbor adjacencies.

This command takes an IP address of a neighbor's interface and the ASN of the neighbor. For a BGP neighborship to be established, each router must have routes to the neighbor's ip.

Router(config-router)# **neighbor [*ip address*] update-source [*interface id*]**

- Exchanges BGP information with the specified interface on the neighbor router

This command works the same for ipv4 and ipv6. Loopback interfaces are often used as the update source because they are virtual and cannot drop down.

Router(config-router)# **neighbor [*ip address*] ebgp-multihop [*hop count*]**

- Roughly defines the number of hops needed to reach an ipv4 or ipv6 BGP neighbor

Increasing the hop count increases the time-to-live (TTL) of a BGP update packet. If BGP packets exceed their time-to-live, then they will be discarded. This may be the case for a packet attempting to reach a loopback neighbor interface.

Router(config-router)# **address-family [*protocol*]** *vrf <vrf name>*

- Enters configuration mode for a BGP address family

I like to think of address-families as workspaces for certain IP protocols. For example, IPv6 is configured in the IPv6 address family. This is where redistribution, network statements or activation commands occur.

Router(config-router-af)# **network [*network address*] mask [*subnet mask*]**

- Advertises a directly connected network to the BGP routing table

BGP's network statements are not to be confused with OSPF or EIGRPs network statements; they aren't used to form adjacencies between BGP routers. A BGP network statement is configured alongside a neighbor statement, the former advertising the network and the latter establishing the neighbor relationship.

Router(config-router-af)# **neighbor [*ip address*] activate**

- Brings up a neighbor previously defined in a neighbor statement

This command is typically configured by default for each neighbor.

Configurations

Router 1

```
R1#show running-config
hostname R1
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface Loopback0
 ip address 100.10.10.10 255.255.255.255
 ipv6 address 10::1/64
interface FastEthernet0/0
 ip address 10.10.10.1 255.255.255.252
 duplex full
 ipv6 address FE80::1 link-local
 ipv6 address 1::1/64
router bgp 10
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 neighbor 20::1 remote-as 20
 neighbor 20::1 ebgp-multihop 2
 neighbor 20::1 update-source Loopback0
 neighbor 20.20.20.20 remote-as 20
 neighbor 20.20.20.20 ebgp-multihop 2
 neighbor 20.20.20.20 update-source Loopback0
 address-family ipv4
  network 10.10.10.0 mask 255.255.255.252
  network 100.10.10.10 mask 255.255.255.255
  neighbor 20.20.20.20 activate
 exit-address-family
 address-family ipv6
  redistribute static
  network 1::/64
  network 10::1/128
  neighbor 20::1 activate
 exit-address-family
ip forward-protocol nd
no ip http server
no ip http secure-server
ip route 20.20.20.20 255.255.255.255 FastEthernet0/0
ipv6 route 20::1/128 FastEthernet0/0 1::2
ipv6 route 20::/64 FastEthernet0/0
control-plane
```

```
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

R1#show ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
C        10.10.10.0/30 is directly connected, FastEthernet0/0
L        10.10.10.1/32 is directly connected, FastEthernet0/0
B        10.10.10.4/30 [20/0] via 20.20.20.20, 00:07:16
B        10.10.10.8/30 [20/2] via 20.20.20.20, 00:07:16
B        10.10.10.12/30 [20/0] via 20.20.20.20, 00:07:16
      20.0.0.0/32 is subnetted, 1 subnets
S        20.20.20.20 is directly connected, FastEthernet0/0
      30.0.0.0/32 is subnetted, 1 subnets
B        30.30.30.30 [20/2] via 20.20.20.20, 00:07:16
      40.0.0.0/32 is subnetted, 1 subnets
B        40.40.40.40 [20/3] via 20.20.20.20, 00:07:16
      50.0.0.0/32 is subnetted, 1 subnets
B        50.50.50.50 [20/0] via 20.20.20.20, 00:07:16
      100.0.0.0/32 is subnetted, 1 subnets
C        100.10.10.10 is directly connected, Loopback0
```

R1#show ipv6 route

```
IPv6 Routing Table - default - 13 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
C   1::/64 [0/0]
     via FastEthernet0/0, directly connected
L   1::1/128 [0/0]
     via FastEthernet0/0, receive
B   2::/64 [20/0]
     via 20::1
B   3::/64 [20/2]
     via 20::1
B   4::/64 [20/0]
     via 20::1
C   10::/64 [0/0]
     via Loopback0, directly connected
L   10::1/128 [0/0]
     via Loopback0, receive
```

```
S   20::/64 [1/0]
     via FastEthernet0/0, directly connected
S   20::1/128 [1/0]
     via 1::2, FastEthernet0/0
B   30::1/128 [20/1]
     via 20::1
B   40::1/128 [20/2]
     via 20::1
B   50::1/128 [20/0]
     via 20::1
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 2

```
R2#show running-config
hostname R2
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface Loopback0
 ip address 20.20.20.20 255.255.255.255
 ipv6 address 20::1/64
 ipv6 ospf 20 area 20
interface FastEthernet0/0
 ip address 10.10.10.5 255.255.255.252
 duplex full
 ipv6 address FE80::1 link-local
 ipv6 address 2::1/64
 ipv6 ospf 20 area 20
interface FastEthernet1/0
 ip address 10.10.10.2 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::2 link-local
 ipv6 address 1::2/64
router ospf 10
 router-id 2.2.2.2
 network 10.10.10.4 0.0.0.3 area 0
 network 20.20.20.20 0.0.0.0 area 0
router bgp 20
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 neighbor 10::1 remote-as 10
 neighbor 10::1 ebgp-multihop 2
 neighbor 10::1 update-source Loopback0
 neighbor 30::1 remote-as 20
 neighbor 30::1 update-source Loopback0
 neighbor 40::1 remote-as 20
 neighbor 40::1 update-source Loopback0
 neighbor 30.30.30.30 remote-as 20
 neighbor 30.30.30.30 update-source Loopback0
 neighbor 40.40.40.40 remote-as 20
 neighbor 40.40.40.40 update-source Loopback0
```

```
 neighbor 100.10.10.10 remote-as 10
 neighbor 100.10.10.10 ebgp-multihop 2
 neighbor 100.10.10.10 update-source Loopback0
 address-family ipv4
  network 10.10.10.0 mask 255.255.255.252
  network 20.20.20.20 mask 255.255.255.255
  network 100.10.10.10 mask 255.255.255.255
  redistribute ospf 10 match internal external 1 external 2
  neighbor 30.30.30.30 activate
  neighbor 40.40.40.40 activate
  neighbor 100.10.10.10 activate
 exit-address-family
 address-family ipv6
  redistribute ospf 20 match internal external 1 external 2
  redistribute static
  network 1::/64
  network 2::/64
  network 10::1/128
  network 20::1/128
  neighbor 10::1 activate
  neighbor 30::1 activate
  neighbor 40::1 activate
 exit-address-family
ip forward-protocol nd
no ip http server
no ip http secure-server
ip route 100.10.10.10 255.255.255.255 FastEthernet1/0
ipv6 route 10::1/128 FastEthernet1/0 1::1
ipv6 route 10::/64 FastEthernet1/0
ipv6 router ospf 20
 router-id 20.20.20.20
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

```
R2#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 6 subnets, 2 masks
C        10.10.10.0/30 is directly connected, FastEthernet1/0
L        10.10.10.2/32 is directly connected, FastEthernet1/0
C        10.10.10.4/30 is directly connected, FastEthernet0/0
L        10.10.10.5/32 is directly connected, FastEthernet0/0
O        10.10.10.8/30 [110/2] via 10.10.10.6, 00:08:00, FastEthernet0/0
B        10.10.10.12/30 [200/0] via 40.40.40.40, 00:07:29
```

```
      20.0.0.0/32 is subnetted, 1 subnets
C        20.20.20.20 is directly connected, Loopback0
      30.0.0.0/32 is subnetted, 1 subnets
O        30.30.30.30 [110/2] via 10.10.10.6, 00:08:00, FastEthernet0/0
      40.0.0.0/32 is subnetted, 1 subnets
O        40.40.40.40 [110/3] via 10.10.10.6, 00:08:00, FastEthernet0/0
      50.0.0.0/32 is subnetted, 1 subnets
B        50.50.50.50 [200/0] via 40.40.40.40, 00:07:29
      100.0.0.0/32 is subnetted, 1 subnets
S        100.10.10.10 is directly connected, FastEthernet1/0
```

R2#show ipv6 route

```
IPv6 Routing Table - default - 14 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
C   1::/64 [0/0]
     via FastEthernet1/0, directly connected
L   1::2/128 [0/0]
     via FastEthernet1/0, receive
C   2::/64 [0/0]
     via FastEthernet0/0, directly connected
L   2::1/128 [0/0]
     via FastEthernet0/0, receive
O   3::/64 [110/2]
     via FE80::2, FastEthernet0/0
B   4::/64 [200/0]
     via 40::1
S   10::/64 [1/0]
     via FastEthernet1/0, directly connected
S   10::1/128 [1/0]
     via 1::1, FastEthernet1/0
C   20::/64 [0/0]
     via Loopback0, directly connected
L   20::1/128 [0/0]
     via Loopback0, receive
O   30::1/128 [110/1]
     via FE80::2, FastEthernet0/0
O   40::1/128 [110/2]
     via FE80::2, FastEthernet0/0
B   50::1/128 [200/0]
     via 4::2
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 3

R3#show running-config

```
hostname R3
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
multilink bundle-name authenticated
```

```
ip tcp synwait-time 5
interface Loopback0
 ip address 30.30.30.30 255.255.255.255
 ipv6 address 30::1/64
 ipv6 ospf 20 area 20
interface FastEthernet0/0
 ip address 10.10.10.9 255.255.255.252
 duplex full
 ipv6 address FE80::1 link-local
 ipv6 address 3::1/64
 ipv6 ospf 20 area 20
interface FastEthernet1/0
 ip address 10.10.10.6 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::2 link-local
 ipv6 address 2::2/64
 ipv6 ospf 20 area 20
router ospf 10
 router-id 3.3.3.3
 network 10.10.10.4 0.0.0.3 area 0
 network 10.10.10.8 0.0.0.3 area 0
 network 30.30.30.30 0.0.0.0 area 0
router bgp 20
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 neighbor 20::1 remote-as 20
 neighbor 20::1 update-source Loopback0
 neighbor 40::1 remote-as 20
 neighbor 40::1 update-source Loopback0
 neighbor 20.20.20.20 remote-as 20
 neighbor 20.20.20.20 update-source Loopback0
 neighbor 40.40.40.40 remote-as 20
 neighbor 40.40.40.40 update-source Loopback0
 address-family ipv4
  neighbor 20.20.20.20 activate
  neighbor 40.40.40.40 activate
 exit-address-family
 address-family ipv6
  redistribute ospf 20 match internal external 1 external 2
  network 2::/64
  network 3::/64
  neighbor 20::1 activate
  neighbor 40::1 activate
 exit-address-family
ip forward-protocol nd
no ip http server
no ip http secure-server
ipv6 router ospf 20
 router-id 30.30.30.30
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
```

```
end

R3#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 6 subnets, 2 masks
B        10.10.10.0/30 [200/0] via 20.20.20.20, 00:07:31
C        10.10.10.4/30 is directly connected, FastEthernet1/0
L        10.10.10.6/32 is directly connected, FastEthernet1/0
C        10.10.10.8/30 is directly connected, FastEthernet0/0
L        10.10.10.9/32 is directly connected, FastEthernet0/0
B        10.10.10.12/30 [200/0] via 40.40.40.40, 00:07:31
      20.0.0.0/32 is subnetted, 1 subnets
O        20.20.20.20 [110/2] via 10.10.10.5, 00:07:57, FastEthernet1/0
      30.0.0.0/32 is subnetted, 1 subnets
C        30.30.30.30 is directly connected, Loopback0
      40.0.0.0/32 is subnetted, 1 subnets
O        40.40.40.40 [110/2] via 10.10.10.10, 00:08:07, FastEthernet0/0
      50.0.0.0/32 is subnetted, 1 subnets
B        50.50.50.50 [200/0] via 40.40.40.40, 00:07:31
      100.0.0.0/32 is subnetted, 1 subnets
B        100.10.10.10 [200/0] via 20.20.20.20, 00:07:31

R3#show ipv6 route
IPv6 Routing Table - default - 15 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
B   1::/64 [200/0]
     via 20::1
C   2::/64 [0/0]
     via FastEthernet1/0, directly connected
L   2::2/128 [0/0]
     via FastEthernet1/0, receive
C   3::/64 [0/0]
     via FastEthernet0/0, directly connected
L   3::1/128 [0/0]
     via FastEthernet0/0, receive
B   4::/64 [200/0]
     via 40::1
B   10::/64 [200/0]
     via 20::1
B   10::1/128 [200/0]
     via 1::1
B   20::/64 [200/0]
     via 10::1
O   20::1/128 [110/1]
     via FE80::1, FastEthernet1/0
C   30::/64 [0/0]
     via Loopback0, directly connected
L   30::1/128 [0/0]
     via Loopback0, receive
O   40::1/128 [110/1]
```

```
     via FE80::2, FastEthernet0/0
B    50::1/128 [200/0]
     via 4::2
L    FF00::/8 [0/0]
     via Null0, receive
```

Router 4

```
R4#show running-config
hostname R4
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface Loopback0
 ip address 40.40.40.40 255.255.255.255
 ipv6 address 40::1/64
 ipv6 ospf 20 area 20
interface FastEthernet0/0
 ip address 10.10.10.13 255.255.255.252
 duplex full
 ipv6 address FE80::1 link-local
 ipv6 address 4::1/64
interface FastEthernet1/0
 ip address 10.10.10.10 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::2 link-local
 ipv6 address 3::2/64
 ipv6 ospf 20 area 20
router ospf 10
 router-id 4.4.4.4
 network 10.10.10.8 0.0.0.3 area 0
 network 40.40.40.40 0.0.0.0 area 0
router bgp 20
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 neighbor 20::1 remote-as 20
 neighbor 20::1 update-source Loopback0
 neighbor 30::1 remote-as 20
 neighbor 30::1 update-source Loopback0
 neighbor 50::1 remote-as 30
 neighbor 50::1 ebgp-multihop 2
 neighbor 50::1 update-source Loopback0
 neighbor 20.20.20.20 remote-as 20
 neighbor 20.20.20.20 update-source Loopback0
 neighbor 30.30.30.30 remote-as 20
 neighbor 30.30.30.30 update-source Loopback0
 neighbor 50.50.50.50 remote-as 30
 neighbor 50.50.50.50 ebgp-multihop 2
 neighbor 50.50.50.50 update-source Loopback0
 address-family ipv4
  network 10.10.10.12 mask 255.255.255.252
  network 50.50.50.50 mask 255.255.255.255
  redistribute ospf 10 match internal external 1 external 2
```

```
  neighbor 20.20.20.20 activate
  neighbor 30.30.30.30 activate
  neighbor 50.50.50.50 activate
 exit-address-family
 address-family ipv6
  redistribute ospf 20 match internal external 1 external 2
  network 3::/64
  network 4::/64
  network 50::1/128
  neighbor 20::1 activate
  neighbor 30::1 activate
  neighbor 50::1 activate
 exit-address-family
ip forward-protocol nd
no ip http server
no ip http secure-server
ip route 50.50.50.50 255.255.255.255 FastEthernet0/0
ipv6 route 50::1/128 FastEthernet0/0 4::2
ipv6 router ospf 20
 router-id 40.40.40.40
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

R4#show ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-ISinter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 6 subnets, 2 masks
B        10.10.10.0/30 [200/0] via 20.20.20.20, 00:07:46
O        10.10.10.4/30 [110/2] via 10.10.10.9, 00:08:22, FastEthernet1/0
C        10.10.10.8/30 is directly connected, FastEthernet1/0
L        10.10.10.10/32 is directly connected, FastEthernet1/0
C        10.10.10.12/30 is directly connected, FastEthernet0/0
L        10.10.10.13/32 is directly connected, FastEthernet0/0
      20.0.0.0/32 is subnetted, 1 subnets
O        20.20.20.20 [110/3] via 10.10.10.9, 00:08:13, FastEthernet1/0
      30.0.0.0/32 is subnetted, 1 subnets
O        30.30.30.30 [110/2] via 10.10.10.9, 00:08:23, FastEthernet1/0
      40.0.0.0/32 is subnetted, 1 subnets
C        40.40.40.40 is directly connected, Loopback0
      50.0.0.0/32 is subnetted, 1 subnets
S        50.50.50.50 is directly connected, FastEthernet0/0
      100.0.0.0/32 is subnetted, 1 subnets
B        100.10.10.10 [200/0] via 20.20.20.20, 00:07:47
```

```
R4#show ipv6 route
IPv6 Routing Table - default - 15 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
B   1::/64 [200/0]
     via 20::1
O   2::/64 [110/2]
     via FE80::1, FastEthernet1/0
C   3::/64 [0/0]
     via FastEthernet1/0, directly connected
L   3::2/128 [0/0]
     via FastEthernet1/0, receive
C   4::/64 [0/0]
     via FastEthernet0/0, directly connected
L   4::1/128 [0/0]
     via FastEthernet0/0, receive
B   10::/64 [200/0]
     via 20::1
B   10::1/128 [200/0]
     via 1::1
B   20::/64 [200/0]
     via 10::1
O   20::1/128 [110/2]
     via FE80::1, FastEthernet1/0
O   30::1/128 [110/1]
     via FE80::1, FastEthernet1/0
C   40::/64 [0/0]
     via Loopback0, directly connected
L   40::1/128 [0/0]
     via Loopback0, receive
S   50::1/128 [1/0]
     via 4::2, FastEthernet0/0
L   FF00::/8 [0/0]
     via Null0, receive
```

Router 5

```
R5#show running-config
hostname R5
boot-start-marker
boot-end-marker
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
multilink bundle-name authenticated
ip tcp synwait-time 5
interface Loopback0
 ip address 50.50.50.50 255.255.255.255
 ipv6 address 50::1/64
interface FastEthernet1/0
 ip address 10.10.10.14 255.255.255.252
 speed auto
 duplex auto
 ipv6 address FE80::2 link-local
```

```
 ipv6 address 4::2/64
router bgp 30
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 neighbor 40::1 remote-as 20
 neighbor 40::1 ebgp-multihop 2
 neighbor 40::1 update-source Loopback0
 neighbor 40.40.40.40 remote-as 20
 neighbor 40.40.40.40 ebgp-multihop 2
 neighbor 40.40.40.40 update-source Loopback0
 address-family ipv4
  network 10.10.10.12 mask 255.255.255.252
  network 50.50.50.50 mask 255.255.255.255
  neighbor 40.40.40.40 activate
 exit-address-family
 address-family ipv6
  network 4::/64
  network 50::1/128
  neighbor 40::1 activate
 exit-address-family
ip forward-protocol nd
no ip http server
no ip http secure-server
ip route 40.40.40.40 255.255.255.255 FastEthernet1/0
ipv6 route 40::1/128 FastEthernet1/0 4::1
control-plane
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
end
```

R5#show ip route

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
B        10.10.10.0/30 [20/0] via 40.40.40.40, 00:07:11
B        10.10.10.4/30 [20/2] via 40.40.40.40, 00:07:45
B        10.10.10.8/30 [20/0] via 40.40.40.40, 00:07:45
C        10.10.10.12/30 is directly connected, FastEthernet1/0
L        10.10.10.14/32 is directly connected, FastEthernet1/0
      20.0.0.0/32 is subnetted, 1 subnets
B        20.20.20.20 [20/3] via 40.40.40.40, 00:07:45
      30.0.0.0/32 is subnetted, 1 subnets
B        30.30.30.30 [20/2] via 40.40.40.40, 00:07:45
      40.0.0.0/32 is subnetted, 1 subnets
S        40.40.40.40 is directly connected, FastEthernet1/0
      50.0.0.0/32 is subnetted, 1 subnets
```

```
C        50.50.50.50 is directly connected, Loopback0
      100.0.0.0/32 is subnetted, 1 subnets
B        100.10.10.10 [20/0] via 40.40.40.40, 00:07:11
```

```
R5#show ipv6 route
IPv6 Routing Table - default - 14 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2, l - LISP
B   1::/64 [20/0]
     via 40::1
B   2::/64 [20/2]
     via 40::1
B   3::/64 [20/0]
     via 40::1
C   4::/64 [0/0]
     via FastEthernet1/0, directly connected
L   4::2/128 [0/0]
     via FastEthernet1/0, receive
B   10::/64 [20/0]
     via 40::1
B   10::1/128 [20/0]
     via 40::1
B   20::/64 [20/0]
     via 40::1
B   20::1/128 [20/2]
     via 40::1
B   30::1/128 [20/1]
     via 40::1
S   40::1/128 [1/0]
     via 4::1, FastEthernet1/0
C   50::/64 [0/0]
     via Loopback0, directly connected
L   50::1/128 [0/0]
     via Loopback0, receive
L   FF00::/8 [0/0]
     via Null0, receive
```

Problems

Loopback Conflict

My first problem occurred when I was assigning IPv4 loopback addresses: the loopbacks conflicted with my main interface addresses. I had a pattern for my loopback ip scheme that went as such: *10.10.10.10* for router one, *20.20.20.20* for router two, *30.30.30.30* for router three, and so forth. The IP scheme I used for my *FastEthernet* interfaces was various [/30] subnets of the *10.10.10.0* network. When I configured all my loopbacks and *FastEthernet* interfaces, I noticed that one router had a loopback address of *10.10.10.10/32* and another had a *FastEthernet* address of *10.10.10.10/30*. These addresses conflicted, so I changed the loopback on router one to *100.10.10.10*. If I were to do this project again, I would use the *10.0.0.0* network for my [/30] subnets to retain a pattern in the loopbacks.

Advertising the loopbacks in OSPF

This problem occurred during the configuration of iBGP, when I tried establishing neighbors using the loopback interfaces. After configuring the statements *neighbor [ip] remote-as [asn]* and *neighbor [ip] update-source loopback0* on each of my iBGP routers, the neighbor relationship failed to form. After some debugging, I learnt that the neighbors were not establishing because there were no routes to the loopback interfaces. Loopback interfaces are not directly connected. I had two options: use static routes to define each loopback interface or use an *Interior Gateway Protocol* (IGP) to route the loopbacks for me. I opted to advertise the loopback interfaces using OSPF.

Static routes for eBGP neighbors

My last problem was establishing eBGP neighbor relationships with loopback interfaces in IPv6. eBGP neighbors would form perfectly fine without loopbacks as the *update-source* but failed when I added the extra command. I tried altering of the *time-to-live* value for the eBGP packets, using the command *neighbor [ip] ebgp-multihop [hop count]*, but met little success.

However, I began to realize that this problem was very similar to my last: there were no routes to the loopback addresses. I decided to use IPv6 static routes to route the loopbacks because my eBGP networks only consisted of two routers. I configured the static routes, yet the neighbors failed to establish.

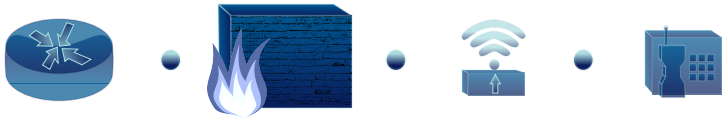
I scoured google for a means to solve this issue, but I found no useful responses; the static routes should have solved my problem. I tried editing the static routes. It is important to be aware of the types of static routes configurable. One type – known as a *directly connected* static route – uses the *exit interface* of the router to forward packets; another type – known as a *recursive* static route – uses the *next hop ip address*, the ip of the neighbor router; the third type – known as a *fully specified* static route – uses both *exit interface* and *next hop ip*. My first implementation of the static routes used *recursive* static routes. For whatever reason, changing the route to a *fully specified* static route fixed the problem.

Conclusion

Overall, I managed to get iBGP and eBGP up and running using loopbacks in both IPv4 and IPv6. Most of my problems stemmed from the routing loopbacks, using OSPF and static routing, but not from the actual BGP commands. For others pursuing the same lab, check connectivity and routes to neighbor's loopback addresses as the first means of troubleshooting. It is also advisable to write an entire IP scheme, ensuring there are no conflicting addresses, before attempting to configure a larger network such as this one.

Layer 2 Security

Network Penetration Testing and Mitigation with Kali Linux

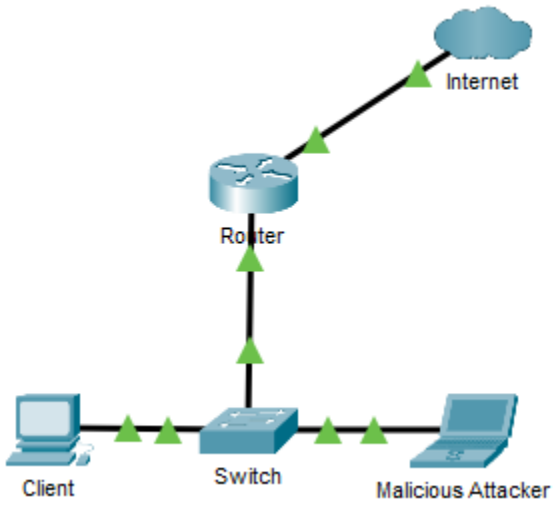


Purpose

With reliance on the internet being at an all-time high, proper security is critical to user safety. In this paper, I cover a couple popular attacks easily performed in Kali Linux, an open-source Linux distribution designed for digital forensics and network penetration testing. Afterwards, I'll demonstrate some techniques used to mitigate them.

Background Information

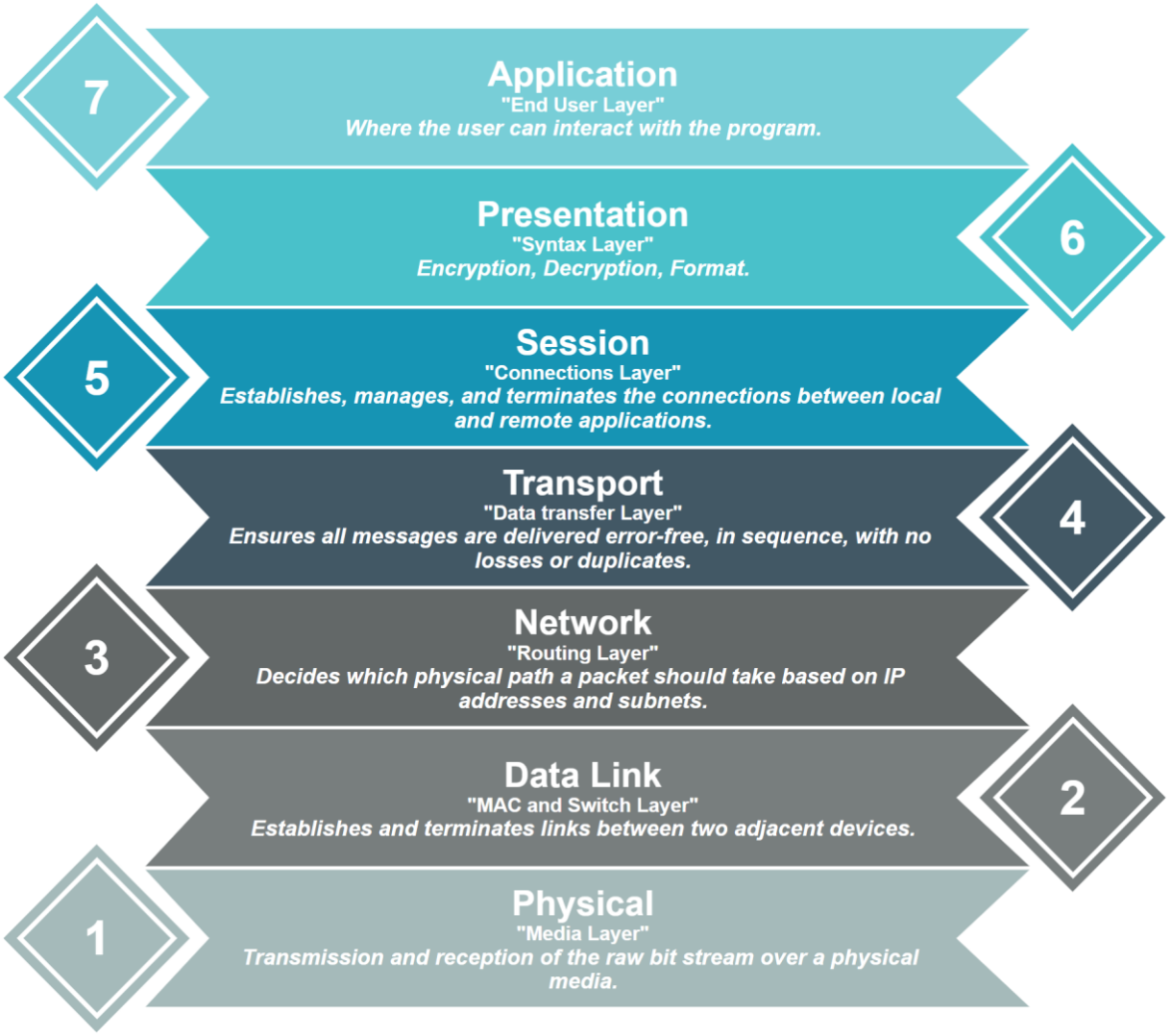
If you're responsible for the security of a company, even down to a small private network, you want to be aware of the vulnerabilities that attackers may exploit. All these attacks assume a switched ethernet network running *internet protocol* (IP). In other words, an attacker has physically infiltrated your building and has access to a specific network device: the *switch*.



A simple way to think of a switch is like a power strip: a single outlet in the wall can be split into multiple outlets if needs be. While a power strip extends the number of outlets, a switch extends the number of ethernet ports. A typical router only has two or three ethernet ports, so at times when more are needed, a network administrator would introduce a switch.

What is Layer 2?

Layer 2 refers to the second layer (Data Link Layer) of the Open Systems Interconnection (OSI) model. There are seven Layers in total:



Although the modern internet doesn't strictly follow the OSI model, it is very useful for troubleshooting and breaking down networking problems into smaller, bite-sized chunks. If a problem can be narrowed down to one specific Layer, a lot of extra work can be avoided.

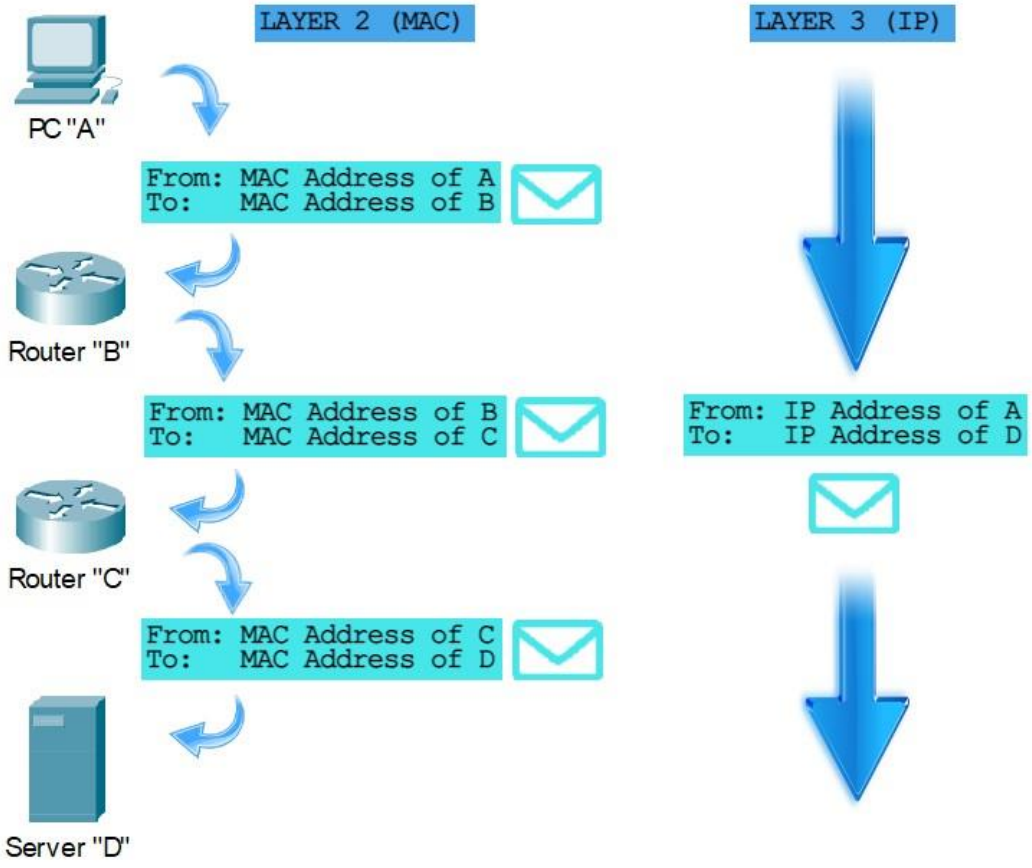
IP addresses vs. MAC addresses

Initially it may seem that IP and MAC addresses are redundant since they both uniquely identify a network device. However, they serve different purposes. MAC operates at Layer 2 of the OSI model – hence why many of these attacks include the stealing and forging of MAC addresses – while IP operates at Layer 3.

During transmission, MAC addresses are used to direct packets from *device to device* while IP addresses from *source to destination*, across multiple devices. The source and

destination MAC addresses will change based on where the packet is in transit, but the source and destination IPs won't.

For example, the average train running through the UK will stop at different stations, whether it's a station the passenger wants to get off at or not. The overhead display will show the previous and next stations, prompting the passenger of where they are in transit. While the passenger only has use for the specific stations they enter and exit, the train will still stop at each station regardless. Layer 2 is a bit like station to station and Layer 3 is more entry station to exit station.



How does a switch function?

Back in the old days, during the times of hubs and repeaters, packets were broadcasted through every port on the device. If ten computers were connected to a hub, and two were trying to communicate, every device would receive the packets then manually drop them unless they were the destined host. You can imagine how hackers took advantage of being dealt free data not destined for them. Eliminating security flaws and saving bandwidth in mind, network engineers created a better device: the switch.

Each port on a switch stores the MAC address of the connected device. Instead of relying on broadcasts, switches intelligently direct traffic to the correct destination based on MAC address. The *Content-Addressable Memory (CAM) table* maps each *port* to a *MAC address* so

the switch can easily direct data using the destination MAC address on a packet through the correct port.

If a host on the *FastEthernet 0/1* port requests a webpage, data will route to and from the webserver and eventually back to the switch (though it will probably take mere milliseconds). After reaching the switch, the packets forward through the port associated with their destination MAC address. Switches can be joined together to support even more ports. A switch-to-switch connection is known as a *trunk port*.

Spoofing and Sniffing

There are three ultimate goals of an attacker: capturing a host's data, altering a host's data, or denying a host access to a certain service (DOS). Many attacks incorporate *spoofed* MAC addresses to *sniff* a legitimate host's packets. Spoofing is where an attacker identifies themselves as different device by falsifying their data. Sniffing refers to capturing packets intended for a different device. Before we get into the attacks, I recommend familiarizing with the general commands.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

// IPv4 DHCP Configuration

Router(config)# **ip dhcp exclude-address [*start ip address*]** *<end ip address>*

- Set an IP or range of IPs to exclude from the pool

If the network administrator so chooses to exclude a range of IP addresses, the range would be from the Initial IP to the End IP, inclusive. The End IP argument is not necessary when excluding only one IP. Excludes are typically reserved for pre-configured static IP addresses, for example, interfaces on the router.

Router(config)# **ip dhcp pool [*pool name*]**

- Creates a pool for distributing routing information

Dynamic Host Configuration Protocol (DHCP) automates the assignment of IP addresses to devices on the local network. An IP pool is used to define the sequential range of IP addresses that the server will divvy out to clients.

Router(dhcp-config)# **network [*network address*] [*subnet mask*]**

- Configures the pool to distribute the set subnet

Router(dhcp-config)# **default-router [*ip address*]**

- Sets the client's default gateway to the specified IP address

Router(dhcp-config)# **dns-server [*ip address*]**

- Sets the client's DNS server to the specified IP address

Router# **show ip dhcp binding**

- Displays the current mappings of the IP pool to the clients

// VLAN Configuration

Switch(config)# **vlan [*id*]**

- Create a vlan with specified *id*

A Virtual Lan (VLAN) is used to partition groups of devices on a switched network.

Switch(config-if)# **switchport access vlan [*id*]**

- Configure an interface to be part of a specified vlan

*If a user desires to configure a vlan across multiple interfaces, use the command **interface range [interface] [start-end id]**.*

// Kali Linux commands

$ sudo su

- Enter root mode

$ ip a

- Display device interface settings, synonymous to *ipconfig* on Windows

$ sudo ip addr add [*IP address*]/[*subnet prefix*] dev [*interface*]

- Add an IP address to a (dev)ice interface

// Graphical Applications

$ sudo ettercap -G

- Opens the Ettercap Graphical interface

Ettercap is a program primarily focused on sniffing packets and man-in-the-middle (MITM) attacks. It is possible to sniff in four modes: IP based, MAC based, ARP based (full-duplex) and PublicARP based (half-duplex) across different interfaces.

$ sudo yersinia -G

- Opens the Yersinia Graphical interface

Yersina is a framework for performing Layer 2 attacks. It is designed to exploit weaknesses in certain networking protocols, such as STP, CDP, DTP, DHCP, HSRP, 802.1Q, 802.1X, ISL and VTP.

$ wireshark

- Opens the Wireshark Graphical interface

Wireshark is a packet analyzer, designed to pick up packets "on the wire" (a live interface).

// Creating VLAN interfaces on Kali (VLAN Hopping)

$ ip link add link [*interface*] name [*name*] type vlan id [*id*]

- Create a VLAN on a specified *interface*, typically *eth0*, with a specified *name* and *id*.

$ ip addr add [*IP address*]/[*subnet prefix*] dev [*link name*]

- Add the IP address of the specified link

$ ip link set dev [*link name*] up

- Set the link to up

// NMAP

$ nmap <*scan type(s)*> <*options*> **[*target*]**

- Probe at a network or target to find information about hosts, open ports, and software

Common NMAP Settings

- **Host Discovery**
  - **-*sL***: List targets only
  - **-*sn***: Scan a network but omit port scans
- **Service and Version Detection**
  - **-*sV***: Attempts to determine the version of the service running on port
  - **-*A***: Enables OS detection, version detection, script scanning, and traceroute
  - **-*O***: Remote OS detection using TCP/IP stack fingerprinting
- **Port Specification**
  - **-*p* [*port*]**: Scan a specific port (for range use ports *a*-*b*)

Attacks

ARP Spoofing

DHCP Starvation and MITM

VLAN Hopping

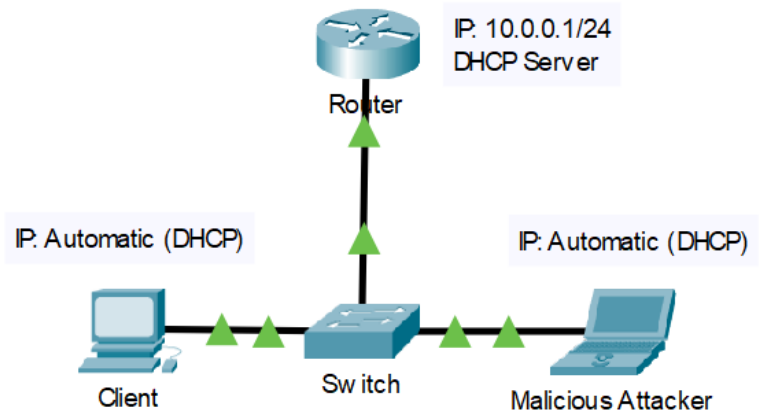
CDP Flooding

Introductory NMAP

Address Resolution Protocol (ARP) Spoofing

ARP spoofing is a type of attack where false ARP packets are sent over a Local Area Network (LAN), linking an attacker's MAC address with an IP of a device on the network. Once the attacker's MAC address is connected to an authentic IP, the attacker will receive data intended for that IP. ARP attacks are used to intercept, modify, or stop data in-transit. ARP spoofing attacks can only occur on LANs that use ARP.

Topology

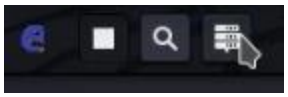


This attack assumes a pre-configured DHCP server. For information on creating a basic DHCP server on a CISCO router, please refer to the commands above.

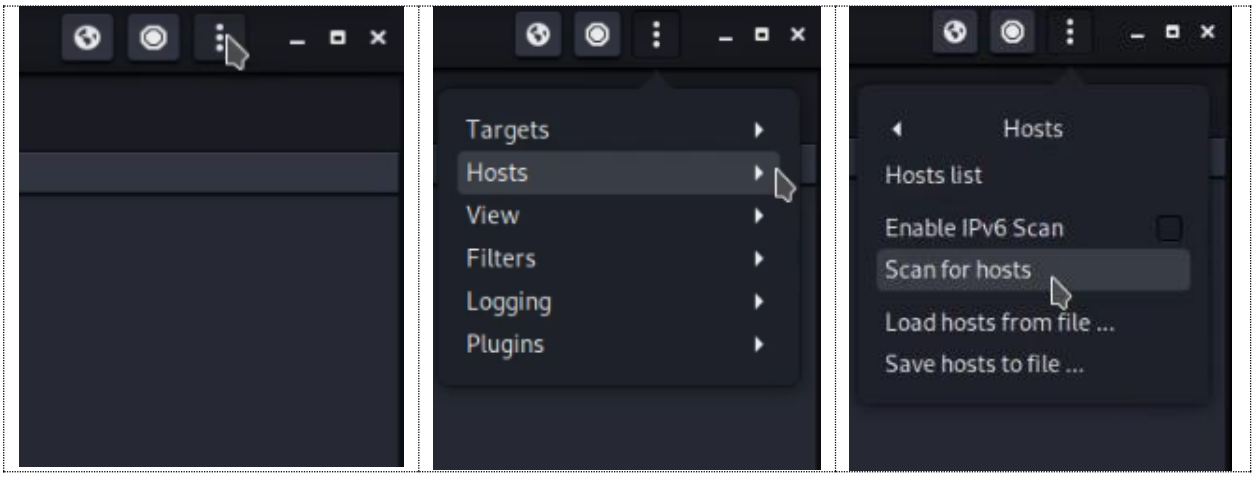
ARP Spoofing with Ettercap Graphical

Open Ettercap and proceed with default settings, making sure the primary interface is ethernet (typically *eth0*).

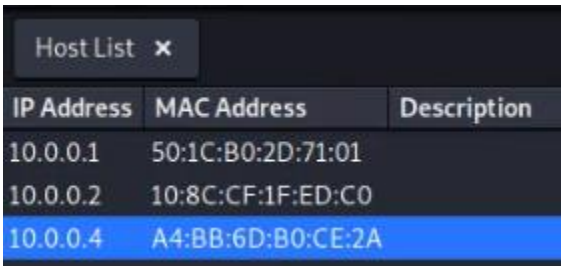
Open the host list (top left for v0.8.3.1).



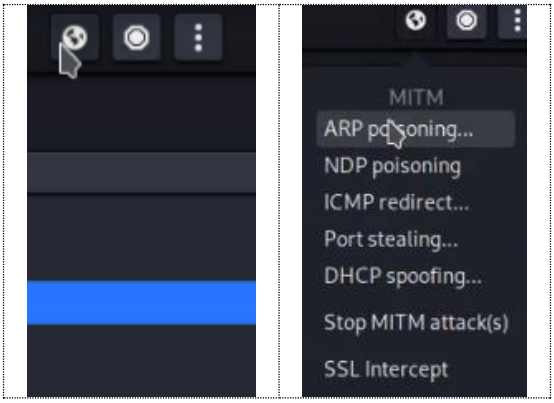
Navigate to the settings tab, go to “Hosts” and “Scan for hosts”



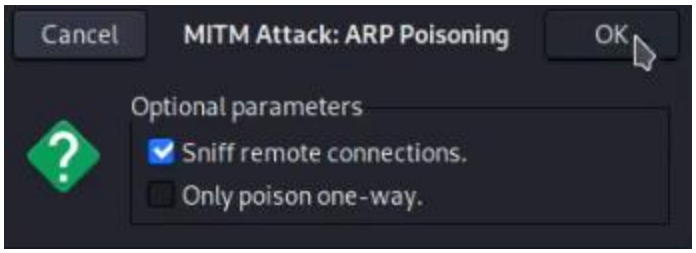
Add the router and a host to Target 1 and Target 2.



Navigate to the MITM tab and select the ARP poisoning attack.



Begin with default settings.



At this point, Ettercap is broadcasting false ARP messages in hope to link your MAC address with the IP of the target. You should be able to open a packet sniffer, such as *Wireshark*, and sniff up packets destined for the legitimate host.

Mitigation

To prevent attacks like ARP spoofing, network engineers must ensure that only valid ARP requests and responses are relayed. *Dynamic ARP Inspection* (DAI) intercepts every ARP message and verifies their MAC address to IP binding before the local ARP cache is updated. Only clients with the correct bindings are forwarded through, otherwise the packets are dropped. There are many unique configurations for DAI, for example, rate limiting, set MAC authentication, or access lists. However, the method I will cover is simply trusting specific ports to skip authentication.

Mitigation Commands

Switch(config)# **ip arp inspection vlan [*id*]**

- Enables DAI on the specified vlan

Switch(config-if)# **ip arp inspection trust**

- Skips DAI on the current interface

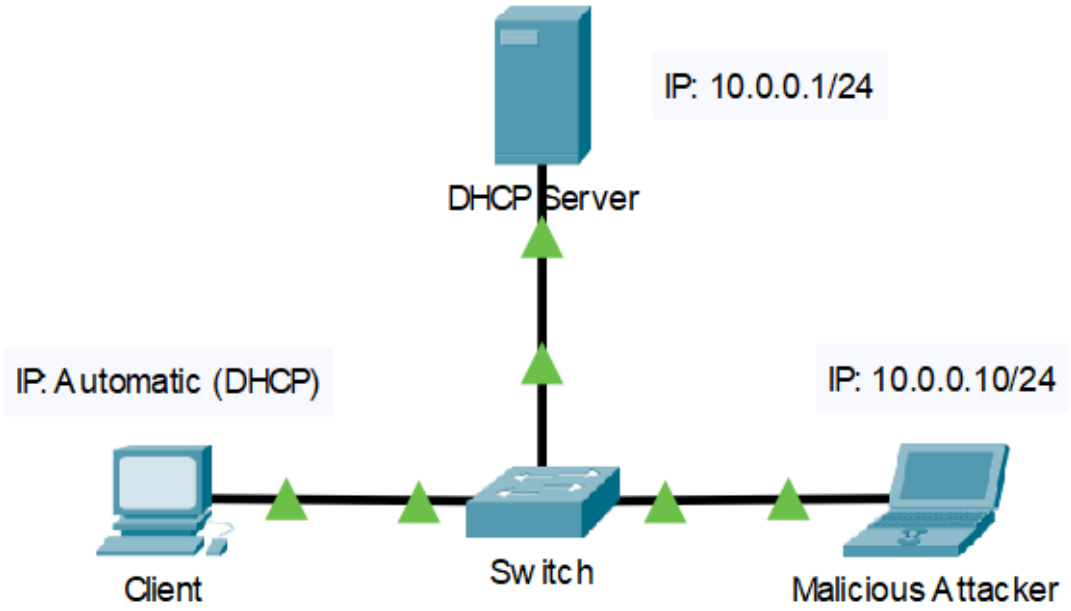
By trusting an interface, you are specifying that you do not want Dynamic ARP Inspection to run. Links to known devices are typically the interfaces you'd want to trust.

Dynamic Host Configuration Protocol (DHCP) Starvation & MITM

In a DHCP Starvation attack, an attacker broadcasts many DHCP requests with spoofed MAC addresses in order to drain the pool of a DHCP server. If the DHCP server responds, its available addresses will deplete within a short amount of time. Once there are no more available addresses, any legitimate client trying to connect to the network won't be allocated an IP address, denying them service to the internet. After "starving" the pool, attackers can try to assert themselves as the new DHCP server.

An attacker launches a rogue DHCP server on their machine. Without access to information from the genuine DHCP server, new clients receive false DHCP settings from the attacker on their requests. These settings might be designed to forward all the client's data to the attacker, then out to the router like nothing was wrong in the first place.

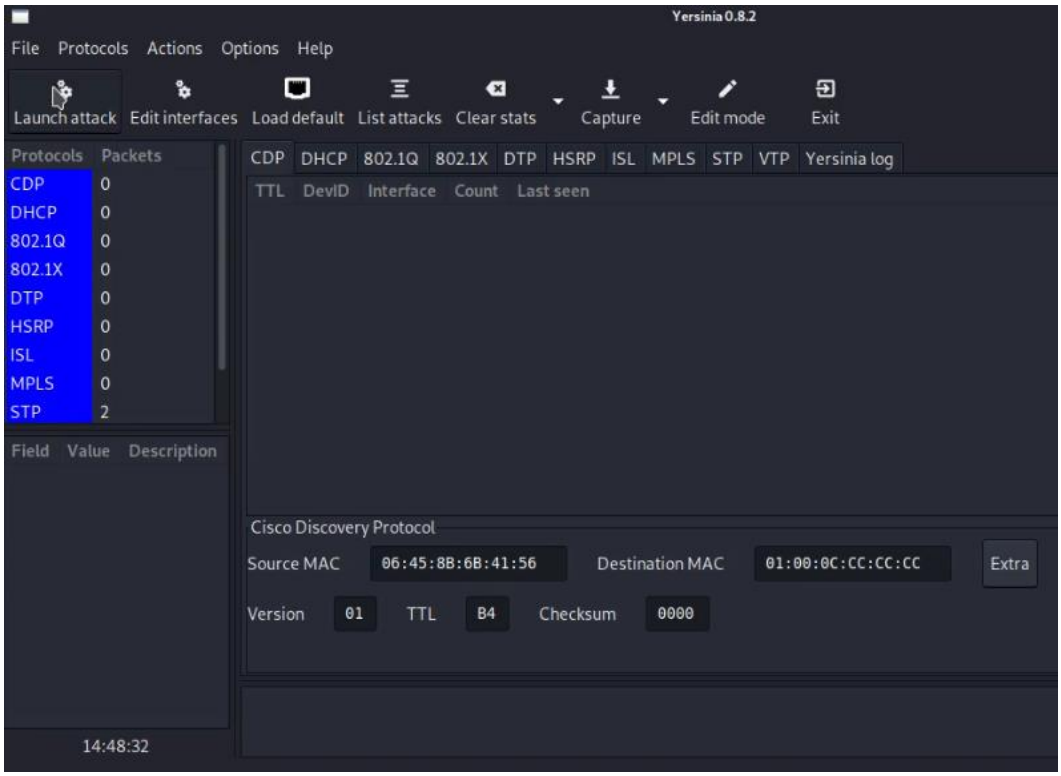
Topology



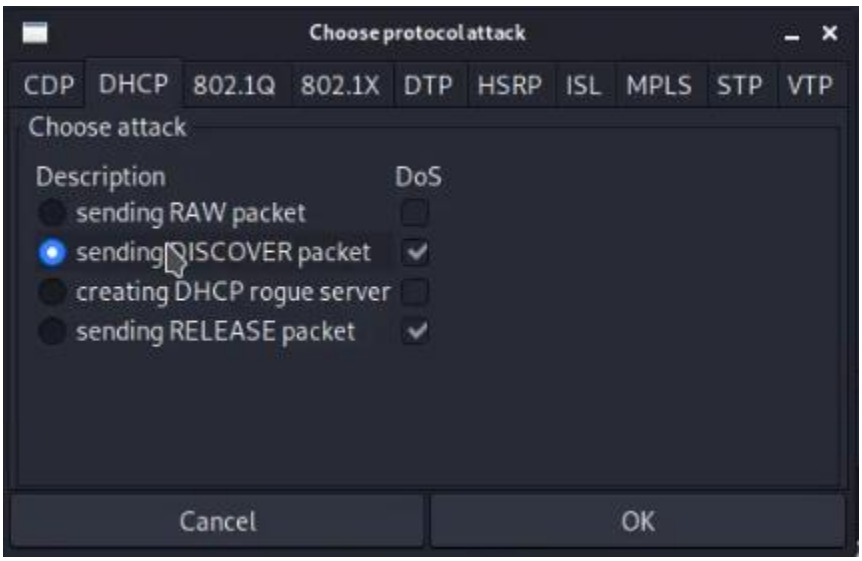
This attack assumes a pre-configured DHCP server. For information on creating a basic DHCP server on a CISCO router, please refer to the commands above.

DHCP Starvation with Yersinia

Upon starting Yersinia, find and hit “Launch attack”.



A popup should appear. Enter the DHCP tab and start sending DISCOVER packets.

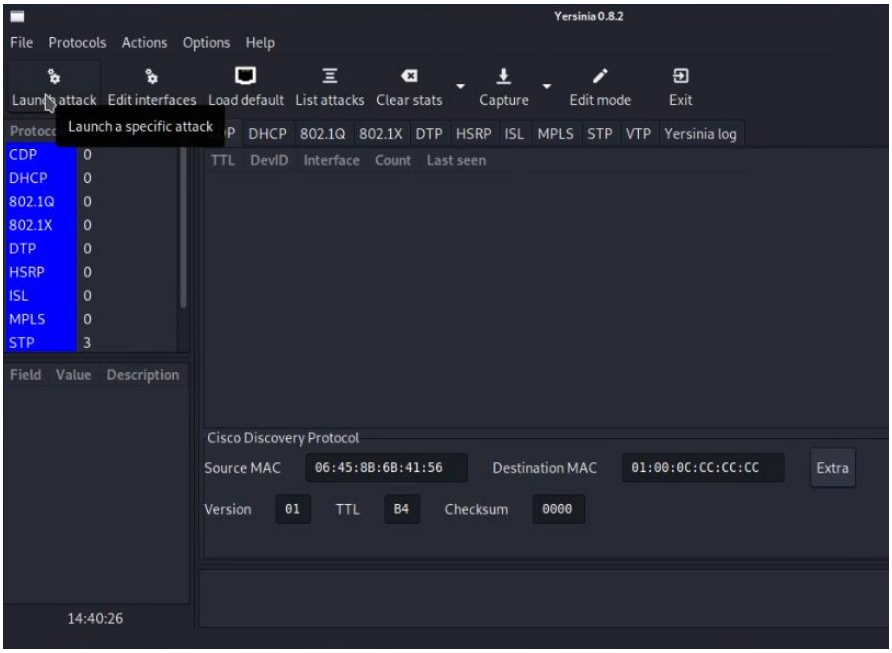


Yersinia should begin to flood the network with DHCP discover packets, devouring the DHCP server’s pool. Any client that requests DHCP information will not be able to obtain it from the legitimate DHCP server.

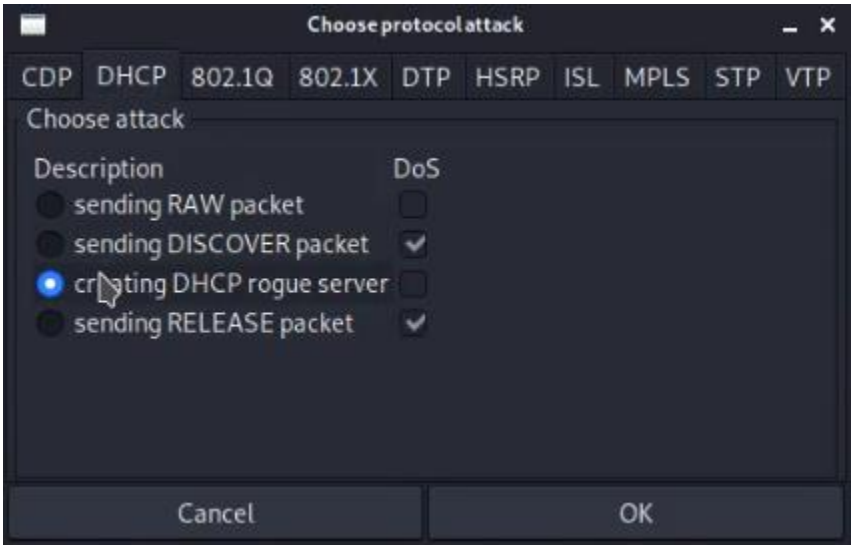
Setting up a Rogue DHCP server in Yersinia

Once the legitimate server's DHCP pool has been exhausted, an attacker can take the opportunity to set up a fake one. The attacker could set up a fake DHCP server without starving the pool, but that would put them in direct competition with the real server. To guarantee their position as the sole DHCP server, a competent attacker would exhaust the pool before imitating a server.

Enter Yersinia and hit "Launch attack"

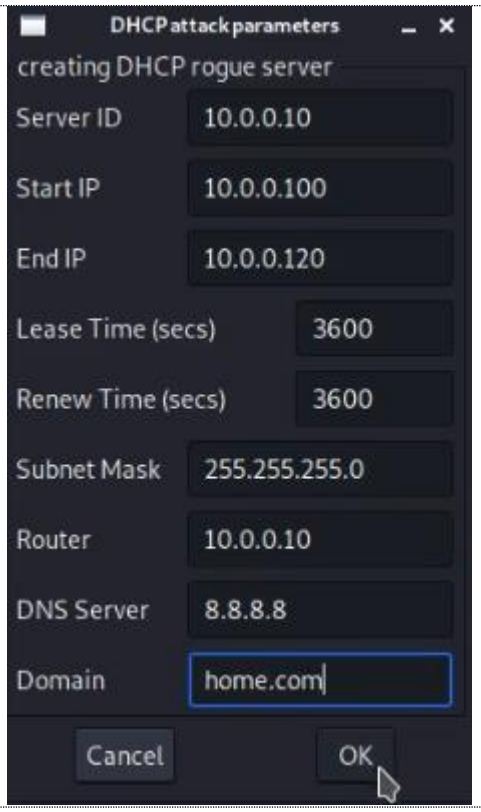


Choose the DHCP tab and create a rogue server

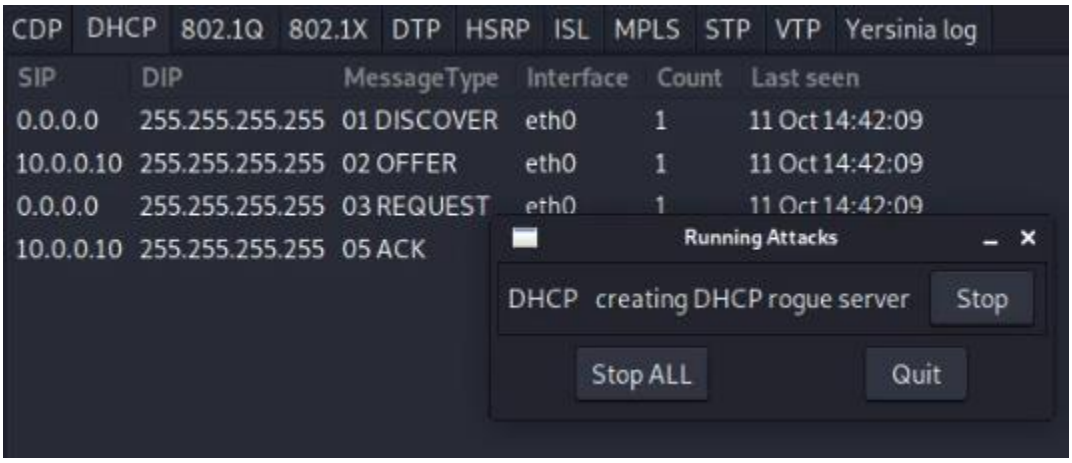


After creating a DHCP server, you should be prompted to configure some parameters. To the right are some examples I used.

- Server ID – the IP of your PC
- Start IP – the beginning address of the IP pool
- End IP – the end address of the IP pool
- Subnet Mask – the mask of the pool
- Router – set the default-router for the client
- DNS Server – set the DNS server of the client
- Domain – set the domain of the client



The rogue DHCP server should now create. Logs can be seen within the DHCP tab in Yersinia, notifying the user of user requests.



Mitigation

Enabling some form of DHCP snooping is a typical the way to counter a DHCP attack. A network engineer should configure DHCP snooping on a switch and trust the port they know to contain the legitimate DHCP server. By enabling DHCP snooping without trusting any ports, every DHCP response is denied, including responses from a genuine DHCP server. They should also restrict the number of DHCP requests able to be sent through a port so attackers can't flood the network with fake requests.

Mitigation Commands

// Preventing a DHCP starvation attack

Switch(config-if)# **ip dhcp snooping limit rate [*packets per second*]**

- Limits the maximum number of DHCP packets allowed through an interface in a given second

// Preventing a rogue DHCP server

Switch(config)# **ip dhcp snooping**

- Enables DHCP snooping on the switch globally

Switch(config)# **ip dhcp snooping vlan [*id*]**

- Enables DHCP snooping on the specified vlan

By design, every interface initially resides within *VLAN 1*, the default vlan. If a switch has no user-configured VLANs, enable snooping on *VLAN 1*.

Switch(config)# **no ip dhcp snooping information option**

- Disable DHCP Option 82

Option 82 can cause problems for certain switches so it's best to disable it.

Switch(config-if)# **ip dhcp snooping trust**

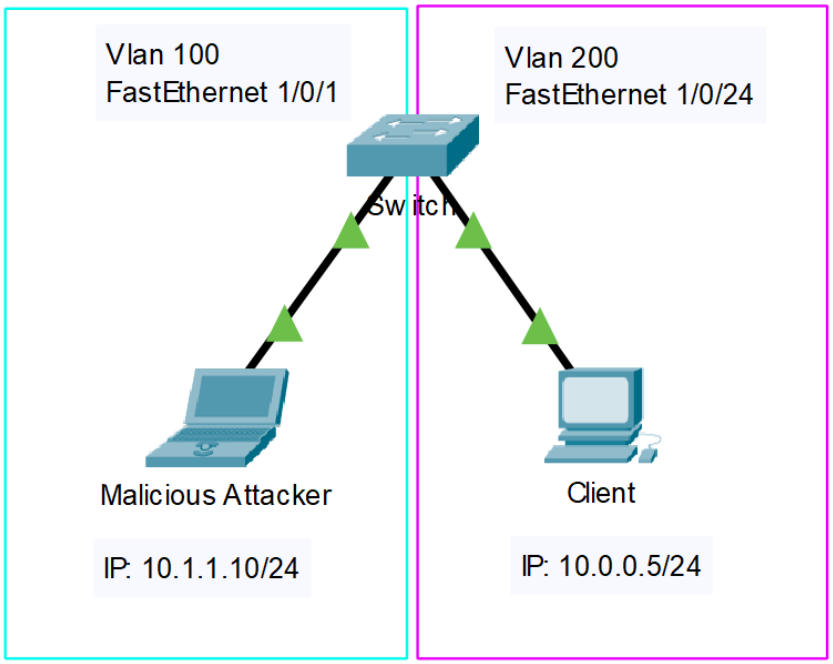
- Trust the interface of the legitimate DHCP server

Virtual Lan (VLAN) Hopping

Before we get into the attack, we need to understand a particular feature on switches – VLANs. Imagine a square room built entirely out of doors. Anyone can pass through a door and leave through another since there is no interference, no rules dictating otherwise. Now, consider a solid wall placed to divide that room into two sections. While the outer foundation is still all doors, the inside is completely split with no ability to communicate with the other section. In essence, this is what you can do to a switch. You can section ports on a switch into a certain VLAN and those ports will only be able to communicate with their respective VLAN. So even if two devices were plugged in directly next to each other, they could only relay traffic if they were on the same VLAN.

The exception to this rule – and what hackers like to exploit – is trunk ports. Switch-to-switch links. Unlike the typical access ports, trunk ports support multiple VLANs so network administrators can extend their VLAN configurations across multiple switches. However, a hacker might pretend to be a trunk link, enabling them access to every VLAN if the connected switch accepts the request. This attack manipulates *Dynamic Trunking Protocol* (DTP) to bypass or "hop" into a different VLAN than what a device is originally part of.

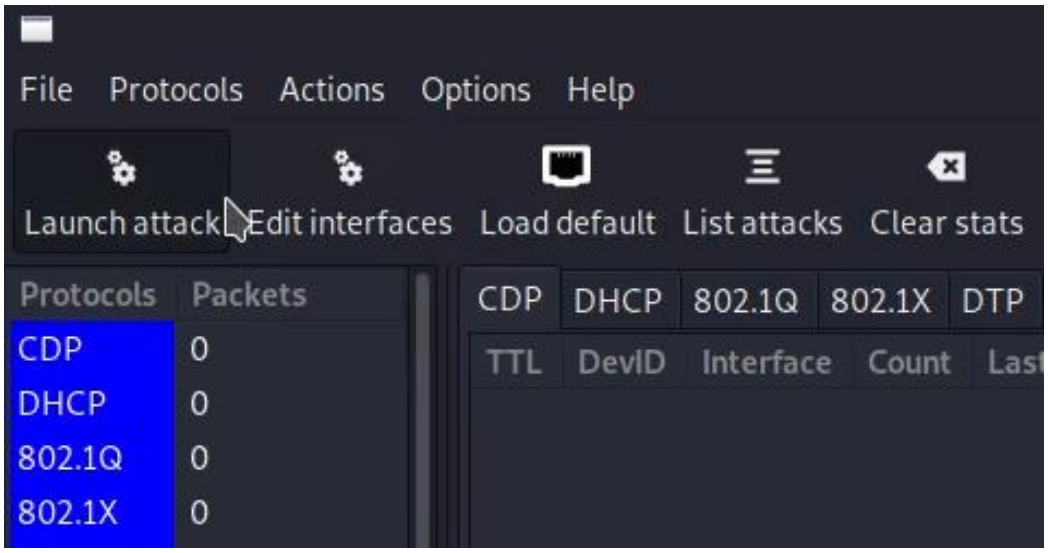
Topology



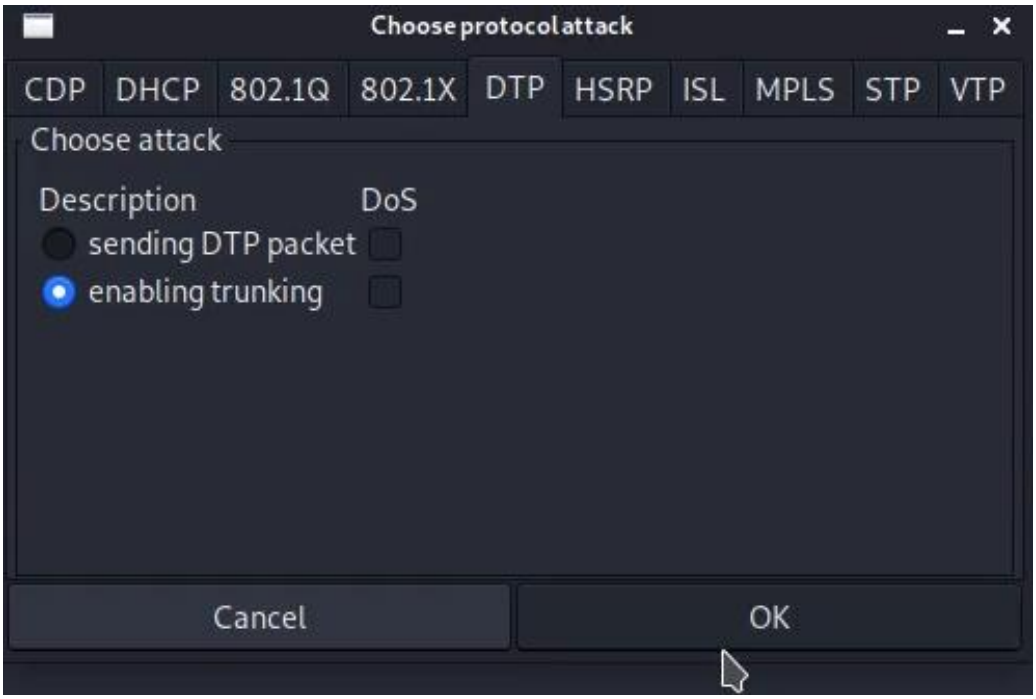
This attack assumes a network with configured VLANs such as this topology. For more information on configuring VLANs, please refer to the command section above.

Enabling a DTP Trunking Link

After starting Yersinia, find and hit "Launch attack".



Under the DTP Protocol tab, enable trunking.

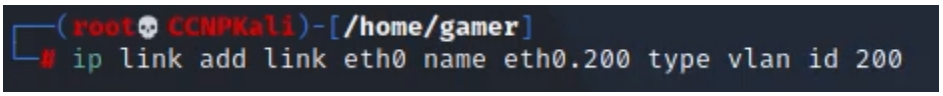


This is all we need from Yersinia – negotiating our link with the switch to be a trunk port – the rest is done in the Kali terminal. Open a new terminal and enter *root mode* by typing *sudo su*.

Creating a VLAN interface on Kali

Root mode grants us access to the highest privilege commands without us having to type *sudo* before every command we enter. Our goal now is to create a VLAN on Kali to match the one on the switch we wish to "hop". The one in this example is *VLAN 200*, which we do like this:

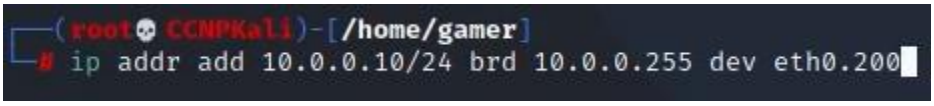
$ ip link add link *eth0* name *eth0.200* type vlan id *200*



We can verify this link has been added using the *ip link* command.

With the VLAN now added in Kali, we need to create an IP for this link. The IP should match the subnet of the victim's VLAN, which happens to be *10.0.0.0/24* and *VLAN 200* in this example.

$ ip addr add *10.0.0.10/24* brd *10.0.0.255* dev *eth0.200*



Finally, set the device link to up.

$ ip link set dev *eth0.200* up

If everything plays out correctly, we should be able to ping and reach devices in an external VLAN, in this case *VLAN 200*.

Mitigation

As I mentioned earlier, *switch-to-switch* connections have *trunk* links while *switch-to-device* connections have *access* links. A default, unconfigured port will negotiate the connection based on the neighboring device. In an ideal world where all devices are honest, this method works perfectly fine. But as we've just learnt, attackers can manipulate the negotiation process to become trunk ports.

Instead of going through the entire negotiation process, we can designate ports as either trunk or access. This is known as port-security, which encompasses many more features than just port types. A network engineer should only configure trunk ports when a switch-to-switch link is guaranteed in their topology, the rest being access ports.

Mitigation Commands

// Defining a Trunk port

Switch(config-if)# **switchport trunk encapsulation dot1q**

- Specify an encapsulation method

To define a trunk port, encapsulation must not be set as "auto". Dot1q is generally used.

Switch(config-if)# **switchport mode trunk**

- Set the mode to trunk.

// Defining an Access port

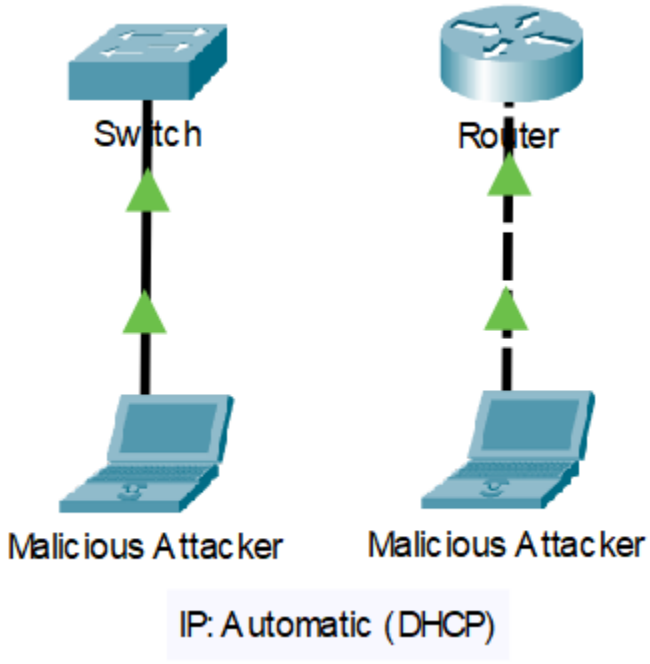
Switch(config-if)# **switchport mode access**

- Set the mode to access.

Cisco Discovery Protocol (CDP) Flooding

Cisco Discovery Protocol (CDP) is a CISCO proprietary protocol used to request information about directly connected devices, such as model, IOS, and IP. In order to exchange information, devices first need to establish a neighborship. Establishing a neighborship expends resources on the device. Not a lot of resources, but if many CDP requests are sent to a device in a short period of time, the processor runs out and stops managing common tasks. CDP Flooding is your run-of-the-mill *denial of service* (DOS) attack where an attacker overloads a device's processor by repeatedly sending CDP packets.

Topology



All you need for this attack is a connection to a CISCO device. In this example, I only attack a switch, but the same process can be done for a router, or practically any other CISCO device.

For reference, let's take a gander at how processes on a Switch look *before* a CDP Flood. Here are some of the commands I used:

Device# **show cdp traffic**

- Get the total packets output and input on the device

Device# **show processes cpu sorted | i CPU utilization | CDP Protocol**

- Check the amount of CPU usage expended by CDP

Device# **show processes cpu history**

- Graphically see the CPU percentage expended over time

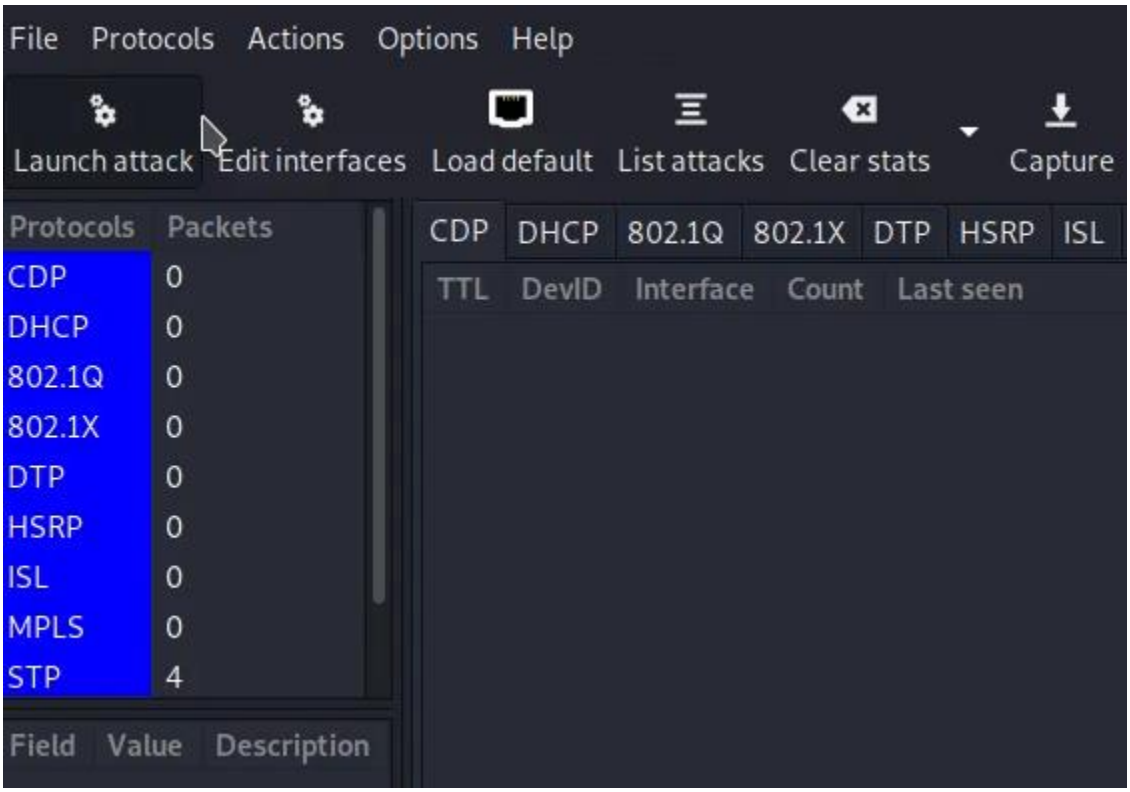
Outputs

```
Switch#show cdp traffic
CDP counters :
        Total packets output: 589, Input: 0
        Hdr syntax: 0, Chksum error: 0, Encaps failed: 0
        No memory: 0, Invalid packet: 0,
        CDP version 1 advertisements output: 0, Input: 0
        CDP version 2 advertisements output: 589, Input: 0

Switch#show processes cpu sorted | i CPU utilization | CDP Protocol
CPU utilization for five seconds: 4%/0%; one minute: 5%; five minutes: 5%
 156           9        3142          2  0.00%  0.00%  0.00%   0 CDP Protocol

Switch#show processes cpu history
      5544444444444444444445555544444444444444444444444444444666
  100
   90
   80
   70
   60
   50
   40
   30
   20
   10 **                    *****                               *
     0....5....1....1....2....2....3....3....4....4....5....5....6
               0    5    0    5    0    5    0    5    0    5    0
               CPU% per second (last 60 seconds)
              * = maximum CPU%   # = average CPU%
```

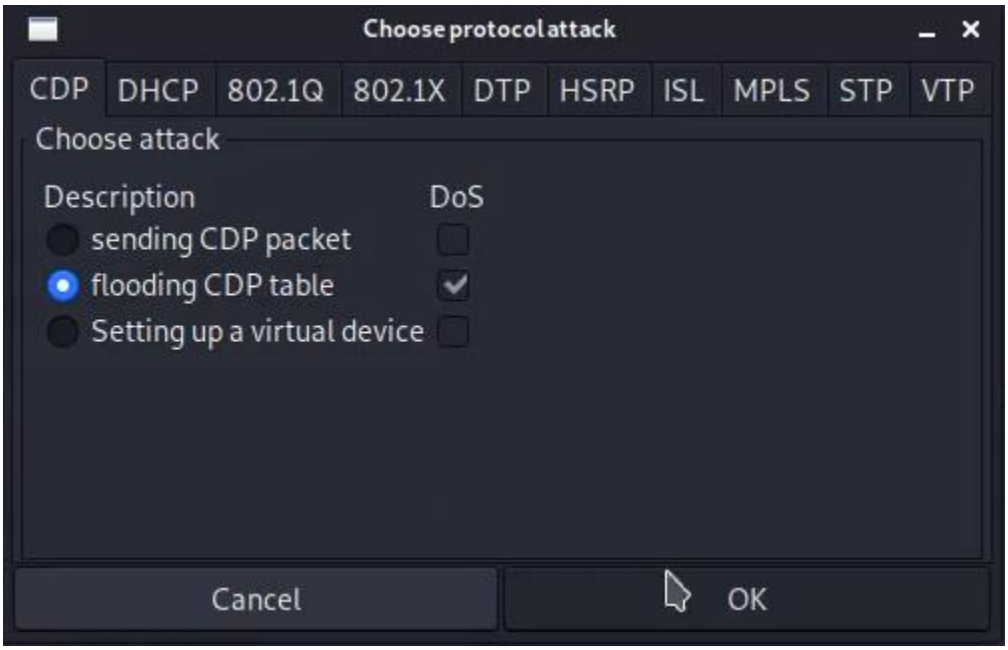
There are no notable processes hindering the CPU of the switch.

CDP Flooding with Yersinia

Begin by opening Yersinia, then click “Launch attack”.



When prompted by the dialogue box, enter the CDP tab and flood the CDP table.



Congratulations, this is all you need to do for a CDP Flooding attack.

Hopping back on the switch, I ran the commands from earlier. CDP annihilated the performance of the switch.

Outputs

```
Switch#show cdp traffic
CDP counters :
        Total packets output: 45, Input: 32509
        Hdr syntax: 0, Chksum error: 0, Encaps failed: 0
        No memory: 0, Invalid packet: 0,
        CDP version 1 advertisements output: 1, Input: 32509
        CDP version 2 advertisements output: 44, Input: 0

Switch#show processes cpu sorted | i CPU utilization | CDP Protocol
CPU utilization for five seconds: 99%/24%; one minute: 82%; five minutes: 32%
 156       51464        3412      15083 34.56% 29.34% 10.42%   0 CDP Protocol

Switch#show process cpu history
      9999999999999999999999999999999999999999999999999999999999
      8899999999998888899999888889999999999888889999988888899999999
  100 ************************************************************
   90 ************************************************************
   80 ************************************************************
   70 ************************************************************
   60 ************************************************************
   50 ************************************************************
   40 ************************************************************
   30 ************************************************************
   20 ************************************************************
   10 ************************************************************
     0....5....1....1....2....2....3....3....4....4....5....5....6
               0    5    0    5    0    5    0    5    0    5    0
              CPU% per second (last 60 seconds)
             * = maximum CPU%   # = average CPU%
```

Mitigation

CDP Flooding is one of the simplest attacks I've performed so far. With devastating effects, it should be clear CDP must not be overlooked. Luckily, the solution is rather straightforward – disable it. Unless you are an engineer in a constantly changing network, there is no desperate need for CDP.

Mitigation Commands

Device(config)# **no cdp run**

- Completely disable CDP on a device

Every CDP packet will be dropped, and the device will not run Cisco Discovery Protocol.

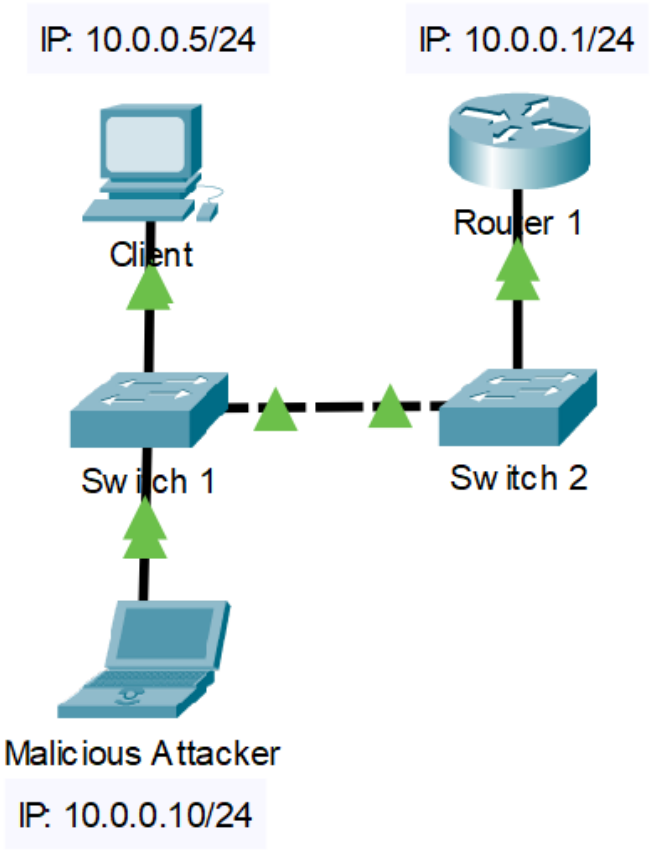
Device(config-if)# **no cdp enable**

- Rather than shutting down CDP, disable it on an interface

Mapping Networks with NMAP

Secure networks will likely have all the mitigations we've covered in place. Which is a great first step, but vulnerabilities still exist. Attackers can use *NMAP* (a network mapper) to find open, exploitable ports and software versions of devices on a network. In this section, I will try to analyze the data gained through running NMAP commands to figure out potential vulnerabilities in my network.

Topology



Let's start by running the command *nmap -O -sV 10.0.0.0/24* (see command section for reference).

```
$ nmap -O -sV 10.0.0.0/24
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-01 14:59 PDT
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try
using --system-dns or specify valid servers with --dns-servers

Nmap scan report for 10.0.0.1
Host is up (0.0011s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
23/tcp open  telnet  Cisco router telnetd (password required but not set)
MAC Address: 50:1C:B0:2D:71:01 (Cisco Systems)
OS details: Cisco 836, 890, 1751, 1841, 2800, or 2900 router (IOS 12.4 - 15.1),
Cisco Aironet 1141N (IOS 12.4) or 3602I (IOS 15.3) WAP, Cisco Aironet 2600-series
WAP (IOS 15.2(2))
Network Distance: 1 hop
Service Info: Device: router

Nmap scan report for 10.0.0.5
Host is up (0.00090s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE       VERSION
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds?
2179/tcp open  vmrdp?
5357/tcp open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
MAC Address: A4:BB:6D:B0:CE:2A (Dell)
Device type: general purpose
Running: Microsoft Windows 10
OS CPE: cpe:/o:microsoft:windows_10
OS details: Microsoft Windows 10 1709 - 1909
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Nmap scan report for 10.0.0.10
Host is up (0.000082s latency).
All 1000 scanned ports on 10.0.0.10 are closed
Too many fingerprints match this host to give specific OS details
Network Distance: 0 hops

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 256 IP addresses (3 hosts up) scanned in 29.04 seconds
```

Practically out of thin air, I'm able to view information on three devices NMAP found:

- A Router
    - IP: *10.0.0.1/24*, MAC: *50:1C:B0:2D:71:01*
    - IOS version likely in the range 12.4 to 15.1
    - 1 Open port
        - 23 – Telnet
    - 1 hop away
- A Host
    - IP: *10.0.0.5/24*, MAC: *A4:BB:6D:B0:CE:2A*
    - Dell running Windows 10

    - 5 Open ports
        - 135 – msrpc
        - 139 – netbios-ssn
        - 445
        - 2179
        - 5357 – http
    - 1 hop away
- The Kali Laptop running the attack

IP and MAC addresses are useful for countless reasons. Open ports give an attacker entryway into a device. By knowing the hop count, we roughly deduce what the topology looks like. A keen eye might spot port 445 (SMB). Port 445 was significantly venerable to an attack called *EternalBlue* back in SMB version 1.0 but is disabled on Windows 10 and above.

Mitigation

There is no straightforward counter to NMAP. Unlike the other attacks so far, NMAP is basically countered by NMAP. It should be used beforehand to find vulnerabilities (mainly open ports) that would be better closed. Close any unused ports.

Process

Installing Kali

As my only objective was to perform the attacks above, I decided to search for an attack tool. There are many applications online that can run various attacks, but the one with the most potential appeared to be a tool called Kali Linux. It came with a lot of base programs and those that weren't pre-installed could be via the command-line interface. At least that's what I thought.

I installed Kali Linux as a windows application, allowing me to run it natively from the Windows 10 operating system. *Ettercap* was a common name that popped up during my research, so I wanted to launch it on Kali. However, the Kali distribution for windows didn't automatically come with Ettercap, so I had to install it manually. That's where my first problem arose. I ran the install command for Ettercap, some information was released into the console, and just like magic Ettercap was now a program I could activate. Sadly, like magic, Ettercap didn't work. Or at least when it opened, it would immediately crash. I narrowed the problem down to a couple of missing dependencies. I installed those dependencies only to have new dependencies reveal their absences. This became an endless cycle with no clear end in sight.

When the pool of dependencies became too deep for me to see the bottom, I started looking into other options to run Kali. At this point in time, I didn't realize Kali was an actual operating system, like Windows or Mac. *Kali Linux* implied to me "a tool on Linux". Of my knowledge at the time, Ubuntu was the only Linux distribution. Therefore, if I got Ubuntu, Kali would come installed, right?

That left me with two options if I wanted to run Ubuntu: use a Windows Virtual Machine (VM) or image it on a spare laptop. Both were equally viable since I was in a position where nothing would be lost if something went wrong. I chose to try a VM, Windows Subsystem for Linux (WSL), which I got working after following a tutorial online. The VM was extremely slow and glitchy and I didn't have any strong feelings towards fixing the issues, so I settled on imaging the laptop. In hindsight, I should have done more research on Kali instead of blindly firing and hoping fixes would miraculously land.

The software I chose to flash Ubuntu on my USB drive was BalenaEtcher. Etcher's sleek, modern design made the interface clear and the instructions straightforward. Select image, select drive. With Ubuntu flashed on my USB, I opened the spare laptop and booted from the USB. A couple of questions and drive partitions later, I could launch Ubuntu. Which is great because Ubuntu was totally what I needed. Except it wasn't.

In the middle of booting Ubuntu, one of my peers entered the room and asked what I planned on doing with it. I replied, explaining how I was about to install Kali, when he told me Kali Linux was its own operating system. Luckily, I had just gotten the experience of flashing USBs and imaging drives, so I repeated the steps once more for Kali. With Kali now imaged, I opened the console and entered *sudo ettercap -G*, the command for launching Ettercap Graphical, as a test to see if programs worked. It ran pre-installed without any dependency issues.

Kali Features

A handful of problems I've run across in Kali have been solved, simply by adding *sudo* to the start of the command. *Sudo* specifies the highest-level root privileges a user can obtain and is often required for higher-level commands to run properly. For example, Ettercap will initially open without *sudo* privileges but then crashes when running major actions.

There are many working tutorials online for most of the attacks, however sometimes the commands were deprecated. Specifically, during my VLAN hopping attack, the deprecated command *vconfig* was used. *vconfig* used to be the way to create VLANs on Kali, but when run, would only produce an output telling the user to use *iproute2* instead. So, I searched up the *iproute2* documentation and found a command that claimed to set up a VLAN. I replaced deprecated sections of tutorials with synonymous *iproute2* commands, piecing together a working alternative.

Ettercap

Within the Ettercap interface, there is a list of hosts. But sometimes the hosts don't show up automatically. I'm particularly proud of figuring out this problem without searching online and instead by exploring the Ettercap's layout and options; one can scan for hosts manually in the hosts settings. I was also confused about the *targets* system Ettercap has. In an Ettercap attack, two IPs need to be specified. I believe this is because one is set as the router while the other is set as the target host. Generally, throughout references I've seen, the best practice is to set the router as Target 1 and the victim as Target 2.

Yersinia

After installing Yersinia and running a DHCP attack, I learnt the starvation part worked while the rogue server failed to create. At one point when I launched Yersinia, I noticed a warning in the console stating I didn't have *libvlc-bin* installed, a dependency for Yersinia. Considering this might be a reason as to why the rogue server failed, I set about installing the package. A website claimed the command *sudo apt-get install -y libvlc-bin* would work, so I ran it then re-installed Yersinia. I seemed to be able to create rogue servers afterwards, though I'm not sure whether to credit the above or just restarting the computer.

Conclusion

In a couple of weeks, someone with no knowledge on Kali Linux or penetration testing was able to perform attacks on a live network. Anyone can learn these tools, so it really is important to set up proper defenses on a network containing sensitive data. In learning the attacks, I feel I understand the protocols exploited to a higher depth.

Wireless Lan Controller

Managing Wireless Access Points with a WLC

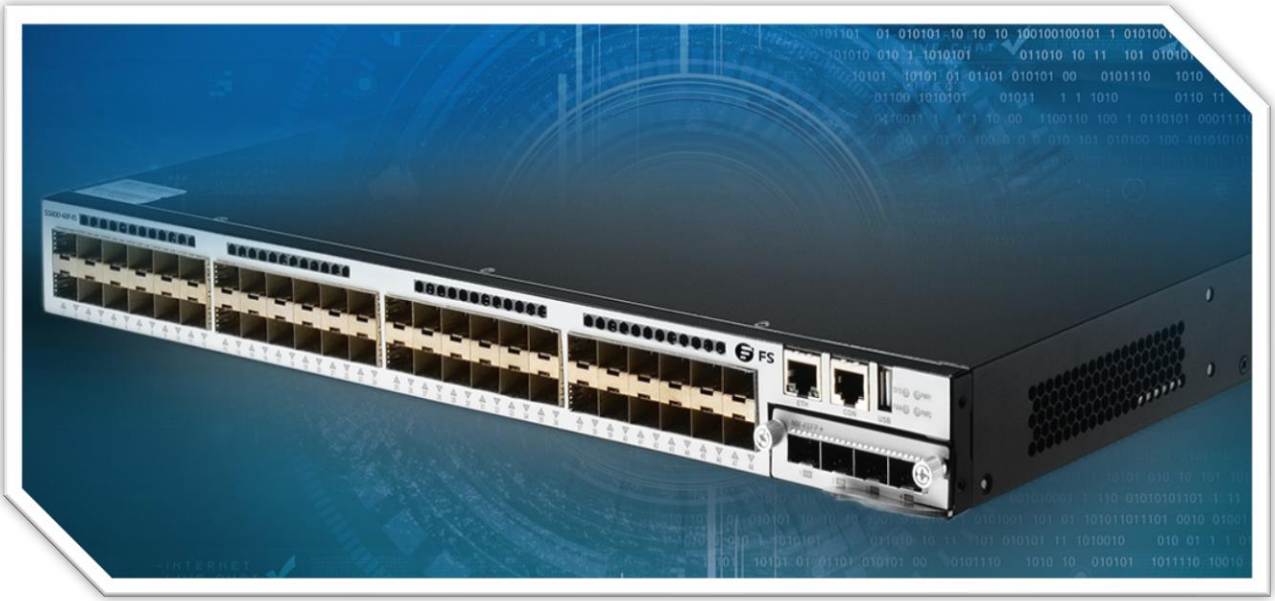
Purpose

As the demand for WiFi access grows, more Wireless Access Points are thrust into larger networks to ensure the signal extends the desired range. However, each additional access point brings more configuration an administrator must account for, becoming increasingly tedious as the number increases. Wireless LAN Controllers monitor and manage access points in bulk, removing a lot of extra hassle for network engineers. In this Lab, I attempt to host a wireless network with access to the internet using one of these controllers.

Background Information

It should be noted that my methods and understanding may not be conventional, as my only goal was to establish connection to the internet for wireless clients using a Wireless LAN Controller. I did not consider security, though that may be something I may implement in the future. That said, in order to understand how each part fits together, I will attempt to cover the following necessary concepts:

- *Switching*
- *VLANs*
- *DHCP*
- *Wireless*
- *The Cisco 5500 series Wireless LAN Controller*
- *NAT*

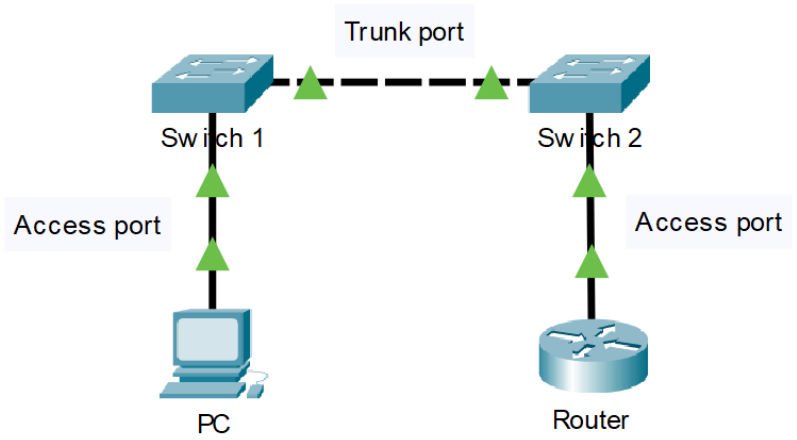


How does a Switch function?

A simple way to think of a switch is like a power strip: a single outlet in the wall can be split into multiple outlets if needs be. While a power strip extends the number of *outlets*, a switch extends the number of *ethernet ports*. A typical router only has two or three ethernet ports, so at times when more are needed, a network administrator would introduce a switch.

Each port on the switch stores the MAC address of the connected device. Instead of relying on broadcasts, switches intelligently direct traffic to the correct destination based on the MAC address. The *Content-Addressable Memory* (CAM) table maps each *port* to a *MAC address* so the switch can easily direct data using the destination MAC address of a packet through the correct port.

If a host connected to the *Fast Ethernet 0/1* port requests a webpage, data would route to and from the webserver and eventually back to the switch (though it will probably take mere milliseconds). After reaching the switch, the packets forward through the port associated with their destination MAC address. Switches can be joined together to support even more ports. A switch-to-switch connection is known as a *trunk port*. Otherwise, the connection is an *access port*. These port types will bring more significance when we look at VLANs.



What are VLANs?

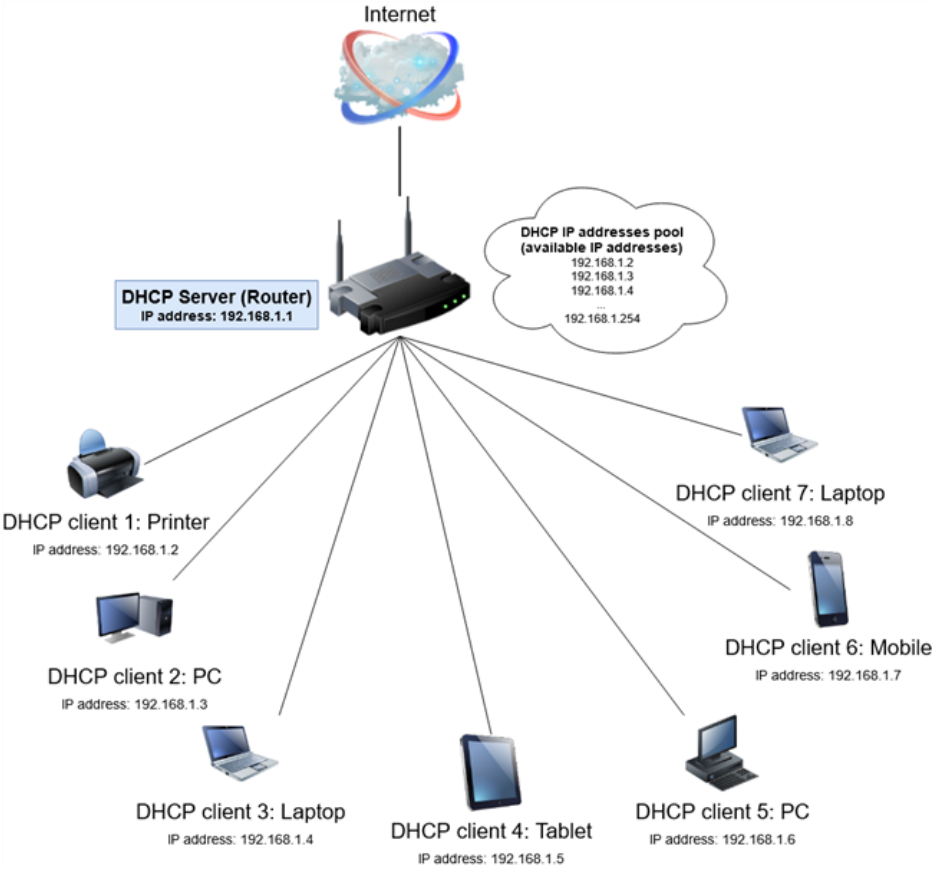
Imagine a square room built entirely out of doors. Anyone can pass through a door and leave through another since there is no interference, no rules dictating otherwise. Now, consider a solid wall placed to divide that room into two sections. While the outer foundation is still all doors, the inside is completely split with no ability to communicate with the other section. In essence, this is what you can do to a switch. You can section ports on a switch into a certain VLAN and those ports will only be able to communicate with their respective VLAN. So even if two devices were plugged in directly next to each other, they could only relay traffic if they were on the *same* VLAN. But as with every rule, there is the exception: *trunk* ports can relay traffic of *multiple* VLANs on the *same* port.

A major use case for VLANs is separating sensitive traffic, perhaps confidential data, on a network from the general traffic. A business might have multiple departments, each with little

reason to communicate with the other, and therefore separate them with different VLANs. Having more VLANs also reduces the size of the broadcast domain – the number of IPs on a subnet – granting less packet congestion.

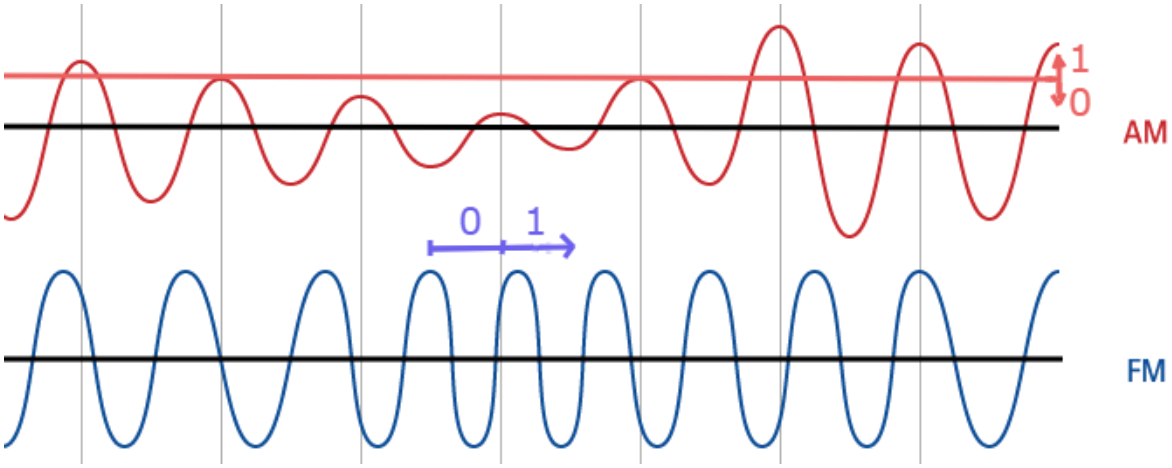
Dynamic Host Configuration Protocol

Dynamic Host Configuration Protocol (DHCP) is a protocol, configurable on a server, that leases IP addresses to clients automatically. For the average person with little networking experience, dynamically receiving addresses is much less confusing that statically assigning them. When many devices need addresses, DHCP provides them, leasing zero duplicates.



How does wireless work?

All wireless transmissions are made possible by radio waves. Radio signals can produce different amplitudes or frequencies of waves, which can be converted to binary 1s and 0s based on how large the amplitude or frequency of the wave is. Imagine you drop a stone into a still lake. Ripples begin to emanate from where the stone hit the water, some big and some small. Almost like a hidden code, we can deduce whether a wave represents a “0” or a “1”, depending on if that wave is above a certain size (Amplitude Modulation) or how compact the waves are together (Frequency Modulation).



Radio waves used for WiFi communication are very similar to those used for walkie-talkies, cell phones, even clinical MRI devices – the difference is the frequency range allocated for each protocol. WiFi transmits at frequencies of roughly 2.4 GHz or 5 GHz. The higher a frequency, the more data a signal can carry.

WiFi uses 802.11 networking standards. Here are some of the 802.11 protocols that have evolved over the decades (From worst to best).

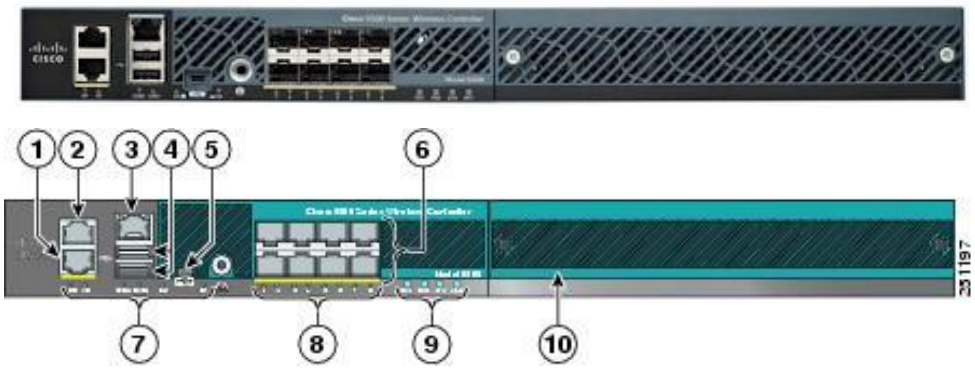
- 802.11b
  - Introduced in 1999
  - Transmits at 2.4 GHz
  - Can handle up to 11 Mbps
  - Cheap cost made it popular, but is less common now as faster standards became cheaper
- 802.11a
  - Introduced after 802.11b
  - Transmits at 5 GHz
  - Can handle up to 54 Mbps
  - Utilizes OFDM, a more efficient coding technique that spits a radio signal into several sub-signals, greatly reducing interference
- 802.11g
  - Released in 2003
  - Transmits at 2.4 GHz
  - Can handle up to 54 Mbps
  - Uses the same OFDM coding as 802.11a
- 802.11n
  - Introduced in 2009
  - Transmits at 2.4 or 5 GHz
  - Can handle up to 140 Mbps
  - Can transmit up to 4 streams of data
- 802.11ac
  - Introduced in 2014
  - Transmits at 5 GHz
  - Can handle up to 450 Mbps
  - Backwards compatible, much less prone to interference and far faster than its predecessors
- 802.11ax (WiFi 6)
  - Introduced in 2019
  - Transmits at 2.4 or 5 GHz
  - Can handle up to 9.2 Gbps

    - Allows manufacturers to install more antennas on a router, accepting more connections at once without worries of interference
- 802.11be (WiFi 7)
    - Projected to be the standard by 2024
    - Transmits at 2.4, 5, or 6 GHz
    - Rates as high as 40 Gbps

Cisco 5508 Wireless LAN Controller

As I mentioned in my introduction, Wireless LAN Controllers (WLCs) are used to remotely manage access points. For those unfamiliar, access points are devices that host wireless networks, allowing clients to join a network without any physical link. Access points usually connect to routers or switches, which handle all the routing and necessary steps to ensure connectivity to the internet. Access points will either run *lightweight* or *autonomous* operating systems. Lightweight access points are controlled by a WLC, and autonomous ones are standalone. Since I am using a WLC, the access points are lightweight.

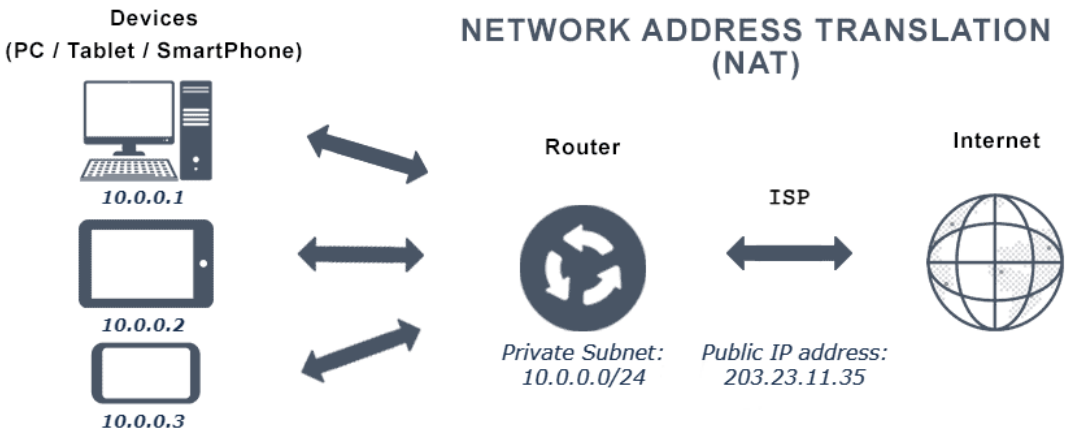
Below are the physical and logical topologies of the 5508 WLC. It is useful to understand the ports and interfaces on the device before configuring it. The WLC operating system is release 8.3.122.0.



1. Redundant port (RJ-45)
2. Service port (RJ-45)
    - *For remotely controlling and managing the WLC using a secure connection*
    - *The service port's IP must be on a different subnet from the management interface*
3. Console port (RJ-45)
    - *For serial connections*
4. USB ports 0 and 1 (Type A)
5. Console port (Mini USB Type B)
6. SFP distribution ports 1-8
    - *SFP (Small Form-factor Pluggable) ports allow fiber or copper connections depending on the module plugged in to the port*
7. Management port LEDs
8. SFP port LEDs
9. Alarm and Notification LEDs
10. Expansion module slot

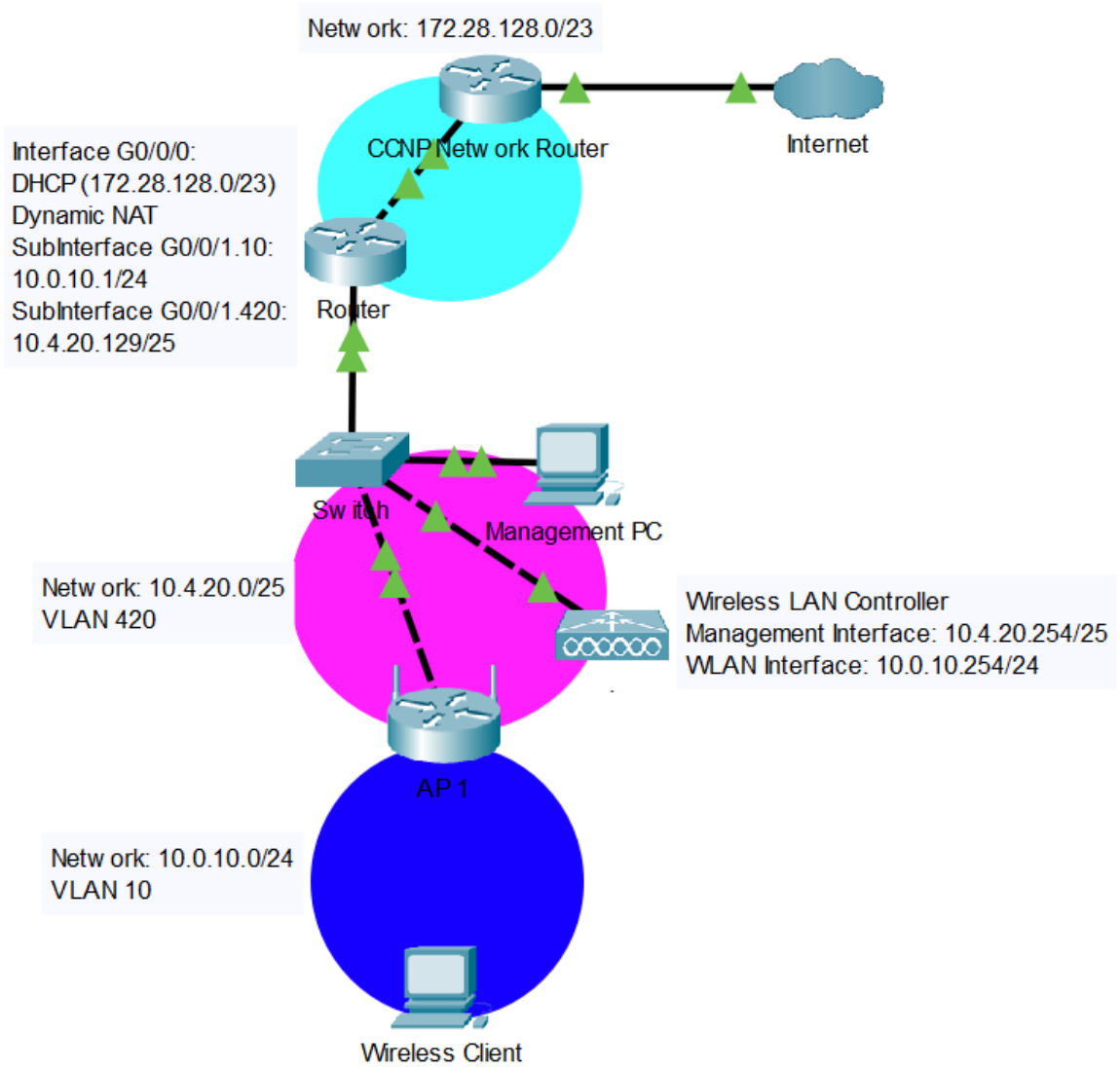
Network Address Translation

Network Address Translation (NAT) is a process that enables a unique IP address to represent an entire group of devices. Typically, this IP address is “Public”, administered by one’s ISP.



In this example, the public IP, *203.23.11.35*, is representing the *10.0.0.0/24* subnet. There can be multiple networks that use the same private subnet – what matters is that the public IP is unique. With so many devices nowadays, the total IPv4 address pool is running out. NAT helps conserve these addresses by translating multiple private IPs through one public IP. This is known as dynamic NAT. There are other kinds of NAT such as static or PAT, but for this Lab I use dynamic NAT.

Network Diagram



Process

In every lab that I have previously done, there were times when I was overwhelmed due to the complexity or size of what I had to do. Usually, this was my fault, as I always tried to brute-force my way through a problem, biting off more that I could chew, instead of breaking it down into practical chunks. Since I was diving into a realm where I had little experience – working with wireless and the WLC – I decided to create manageable goals for each day, to prevent becoming helplessly lost. While some goals spanned more than a day, they helped narrow my focus so I could be more productive.

Connecting to the WLC

I began with arguably the most important step in working with any device: connecting to it. All I wanted was to access an interface which lets me interact with the WLC. Since Cisco routers and Switches are mainly configured from console, I connected my computer to the WLC with a console cable. Almost immediately, I was met with a bunch of questions regarding the initial setup. What should the system name, username, or password be? Do I want to enable SNMP? Enter the service-port IP address. Overwhelmed by the options, I took a step back and searched for some documentation.

Initial Setup

The documentation wasn't very hard to find. It did, however, make up for that by being as overwhelming as the questions. Luckily, there was a section on initial setup. Except the steps were not using the console interface, they were using the web interface. I decided to primarily configure the WLC using the web interface.

The documentation provided me with a lot of options, though many of which were not expanded upon to my liking. To flush out the details that were not clear to me, I decided to compile my own list of the most important information, providing extra reference to areas that lacked explanation, so I could refer to that instead of the official documentation. The major steps that I compiled are as followed: the system name is the WLC's name, the username and passwords are for administrative login; leave SNMP default, it probably doesn't need to change; the service port is for remote management and must be on a different subnet from the management interface; enable Link-Aggregation (it apparently simplifies controller configurations); use *VLAN 0*, assign *port 1* and configure a separate DHCP server for the management interface; leave the RF group as default; use the country code that the network will be deployed in, multiple can be selected if the network extends across multiple countries; use a *192.0.2.0/24* address for the virtual interface since this is for testing purposes (*RFC 5727*); the wireless profile and SSIDs will show up as the WiFi networks; disable the RADIUS server and enable all *802.11* protocols. While I ended up changing some settings later on, taking the time in the beginning to create this configuration guide massively propelled me forward. Referencing my guide, I made a list of configurations ahead of time before finishing the setup wizard.

Connecting an Access Point to the WLC

After completing the wizard with my researched configurations, I finally had access to the main control panel of the WLC. Although the documentation was significantly helpful for the initial setup, I decided to use other sources for the rest; it was difficult picking the details that I needed out of a thousand-page PDF file.

To establish a connection with a WLC, an access point needs an IP address. In lightweight mode, it is conventional for the access point to gain the information by DHCP. There may have been a way to set up DHCP internally on the WLC, however, I chose to use an external Cisco router as a DHCP server. The access point received addresses, but then complained because it didn't know the address of the controller. Fortunately, this problem is

solvable with DHCP option 43, a configurable setting that forwards WLC addresses in hexadecimal. With DHCP option 43 configured, everything should have been working. But it wasn't.

To top it all off, my PC stopped receiving DHCP information. While the timing may seem coincidental with the new DHCP server changes, the error raised, "*an error occurred while renewing interface Ethernet: The object already exists*", had no relation to DHCP option 43. I tried a couple online solutions: resetting device interfaces, resetting Winsock, and restarting my computer. Not even relating to the WLC, this problem took me some time to figure out. Eventually I went into Windows *adapter settings* and noticed VirtualBox, a VMware interface, was running. Out of ideas, I disabled everything that was not ethernet. Thankfully, the virtual interface was the problem, and I was able to resume troubleshooting the WLC.

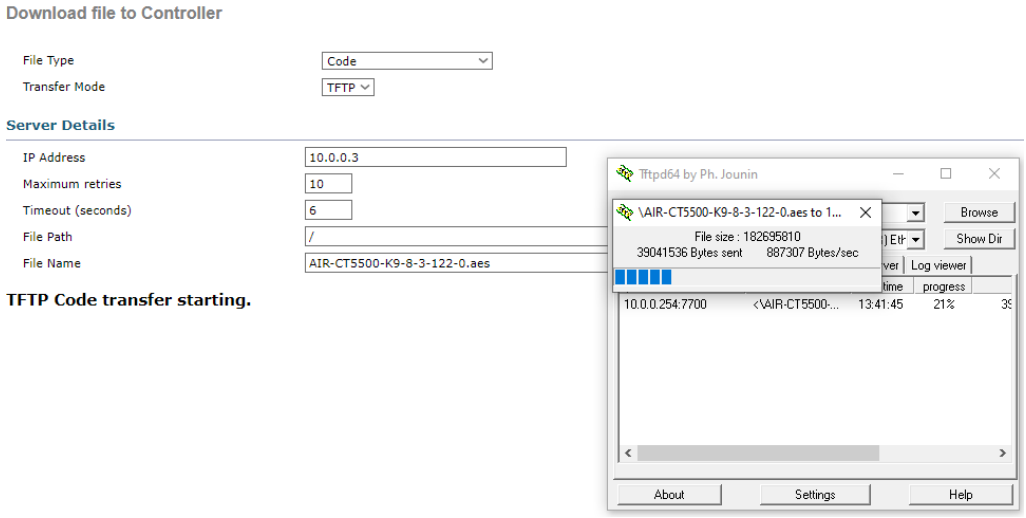
Assuming the DHCP server was properly configured, there were only two devices which could be causing issues: the WLC or access point. After viewing debug logs of both devices, I decided to start clean and reset the access point to factory default. The WLC had just been initialized from factory, so it was far more likely something was incorrectly configured on the access point. My first approach was clearing the CAPWAP configuration, the protocol used for a connection between access points and WLCs, which achieved nothing. Then there was the "complete wipe" approach. For this to work, I needed to enter ROM monitor (ROMMON) – low-level firmware that runs before the operating system boots – by holding down the power button then type *delete flash:private-multiple-fs*, deleting the configuration file. Once the old configuration file is removed from ROMMON, the access point will create a brand-new one on boot.

Resetting the access point to factory default did not fix any errors, oddly, it instead increased the log messages. These new log messages sent me down a new path – now they were claiming incompatibility between the WLC IOS and the access point IOS. The integrated operating system (IOS) on the access point was relatively new, which could only mean one thing. The WLC was outdated. This gave me two options: downgrade the access point IOS or upgrade the WLC IOS. But before I could work on that, I was hit with another problem.

The Cisco 5508 Wireless LAN Controller was released on the 18th of May 2009. How could this possibly be related? Well, access points and WLCs both require in-date Manufacturer-Installed Certificates (MIC) to establish a secure *DTLS-tunnel* between them for encrypting CAPWAP control traffic. These certificates expire ten years after the manufacturing date. As of configuring the WLC in early December 2021, the certificate had long expired. Now I had two problems: installing a new operating system and figuring out a solution for the MIC.

There was a workaround to ignore the MIC: running the command *config ap cert-expiry-ignore mic enable*. However, the current IOS, release *7.0.112.21*, on the WLC did not support that command. I could tackle both these problems at once, but that came at the cost of upgrading the IOS of the WLC. The reason for this being costly is quite literal – obtaining a newer IOS requires a license.

Eventually, I obtained the IOS with the help of a friend. I updated the controller, ran the command above, and marveled as the access point appeared in the list.



Setting up VLANs

Once the links established, my first order of business was to change the topology. After spending a lot of time troubleshooting and interacting with the WLC, I built up insight on how the WLC operates. I needed to correct some issues in my topology, such as introducing different VLANs for the management and user networks. I started by configuring the web interface, but made a mistake, loosing connectivity. I think I mistyped an IP. This meant was I had to switch to the console interface instead.

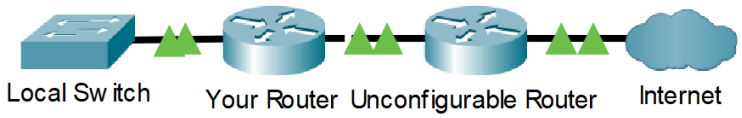
The console interface was much like any other Cisco console interface. Once connected, I found some commands online to change the management interface's IP and VLAN. I set up VLANs on a switch to accompany the new WLC configurations and added sub-interfaces on my router to handle the new VLANs on the switch.

Each interface of the WLC is mapped to a Wireless LAN. Any traffic that a WLAN receives is sent out its respective interface. During the mapping process, I had a problem where a configured WLAN would not appear as a mappable option on the interface that I wanted it mapped to. Searching the configurations, I discovered that *dynamic AP management*, the setting for the management interface, was wrongly enabled on the user-wlan interface. Essentially, this

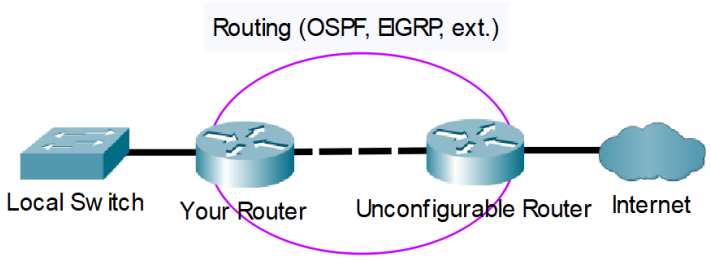
meant that management was being done within the same domain as the clients. Bad. Simply swapping the *dynamic AP management* setting to the correct interface solved the problem. At this point, any client could join the wireless network but could not access the internet.

Configuring NAT

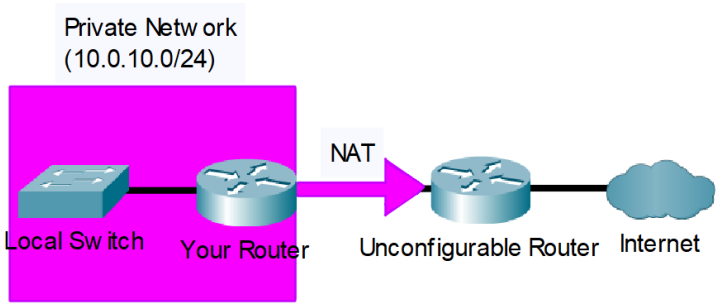
The only reason I needed to configure dynamic NAT was because there was a router blocking the path between my local network and the internet. Consider this topology.



In this topology, the unconfigurable router has access and routes to the internet but doesn't have any routes to the local network, therefore, isolating the local network from the internet. My initial idea was to set up some form of routing, perhaps using OSPF or EIGRP, between the two routers.



However, it was problematic setting up a routing protocol since I needed access to both routers. One of them was inaccessible. Luckily, the unconfigurable router had an interface connecting to my local network. Since the local router was directly connected to the unconfigurable router, that meant it had connectivity to the internet on the directly connected interface. Using dynamic NAT, I forwarded the local traffic to the unconfigurable router, bypassing the routing. I suppose one could think of NAT like a mayor, speaking for his people.



After NAT was configured, I could hop on to my wireless network and browse YouTube.

Lab Commands

command	A statement necessary for a configuration to work, denoted in bold
[*argument*]	An argument necessary for a command to function, denoted in bold italics.
optional-statement *<optional argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

// IPv4 DHCP Configuration

Router(config)# **ip dhcp exclude-address [*initial ip*]** *<final ip>*

- Set an IP or range of IPs to exclude from the pool

If the network administrator so chooses to exclude a range of IP addresses, the range would be from the Initial IP to the End IP, inclusive. The End IP argument is not necessary when excluding only one IP. Excludes are typically reserved for pre-configured static IP addresses, for example, interfaces on the router.

Router(config)# **ip dhcp pool [*pool name*]**

- Creates a pool for distributing routing information

Dynamic Host Configuration Protocol (DHCP) automates the assignment of IP addresses to devices on the local network. A DHCP pool is used to define the range of IP addresses that the server will divvy out to clients.

Router(dhcp-config)# **network [*network address*] [*subnet mask*]**

- Configures a pool that distributes the specified subnet

Router(dhcp-config)# **default-router [*ip*]**

- Sends the specified default gateway to clients

Router(dhcp-config)# **dns-server [*ip*]**

- Sends the specified DNS server to clients

Router(dhcp-config)# **option 43 hex [*value*]**

- Advertises Wireless LAN Controller(s) addresses in hexadecimal

// VLAN Configuration

Switch(config)# **vlan [*id*]**

- Create a vlan with specified *id*

A Virtual Lan (vlan) is used to partition groups of devices on a switched network.

Switch(config-if)# **switchport mode access**

- Configure an interface to be in access mode

Only one vlan is permitted across an access mode interface. Devices connected to an access-configured interface are usually end devices, such as clients.

Switch(config-if)# **switchport access vlan [*id*]**

- Configure an interface to be part of a specified vlan

*To configure the same command access across multiple interfaces, use the command **interface range [interface] [start-end id]**.*

Switch(config-if)# **switchport trunk encapsulation dot1q**

- Configures the interface to use IEEE 802.1Q encapsulation on frames when the interface is configured as a trunk

The user must define the encapsulation before setting an interface as a trunk.

Switch(config-if)# **switchport mode trunk**

- Configure an interface to be a trunk

// Router Sub-Interface Configuration

Router(config)# **interface [*id*].[*vlan id*]**

- Enter sub-interface configuration mode

Sub-Interfaces are used in conjunction with vlans. To route vlans, a sub-interface must be created for each respective VLAN.

Router(config-subif)# **encapsulation dot1Q [*vlan id*]**

- Configures the interface to use IEEE 802.1Q encapsulation on frames

While 802.1Q is not the only networking standard supporting vlans, it is the most prominent. The vlan id should be a vlan existing on a connected switch desired to be routed.

Router(config-subif)# **ip address [*ip*] [*mask*]**

- Configure an IP for the sub-interface, just like any normal interface

// Dynamic NAT Configuration

Router(config-if)# **ip nat [*inside/outside*]**

- Configure an interface to be internal or external

Inside interfaces are translated through the outside interface.

Router(config-if)# **access-list [*#*] permit [*network address*] [*wildcard mask*]**

- Create an access list that permits a subnet

The subnet specified here should be on an inside interface. NAT will translate the subnet out the outside interface. Keep note of the [#] defined, for that will be used in a later command.

Router(config-if)# **ip nat inside source list [*#*] interface [*id*] overload**

- Enable the translation of an access list through an outside interface

The source list [#] should be an access list created earlier; the interface id should be the outside interface.

// Wireless LAN Controller Console Commands

(Cisco Controller) > **config interface create [*name*] [*vlan id*]**

- Create an interface with a name and vlan

(Cisco Controller) > **config interface address management [*ip*] [*mask*] [*default gateway*]**

- Configure the pre-defined management interface

There are other pre-defined interfaces, such as the virtual interface, that are edited the same way. Before an interface can be edited, it needs to be disabled.

(Cisco Controller) > **config interface address dynamic-interface [*name*] [*ip*] [*mask*] [*default gateway*]**

- Configure a user-defined interface

Interfaces are mapped to wlans. Traffic that the interface receives from the wlan is tagged with interface's vlan. In other words, each wlan typically is part of a unique vlan. Before an interface can be edited, it needs to be disabled.

(Cisco Controller) > **config wlan create [*#*] [*interface name*] [*ssid*]**

- Create a wlan and map it to an interface

The SSID is the name of the wireless network that clients will join. Each wlan is numbered with no repeats. The model WLC that I use supports up to 512 wlans.

(Cisco Controller) > **config interface vlan [*interface name*] [*vlan id*]**

- Change the vlan of a defined interface

Configurations

Router

```
Router# show running-config
boot-start-marker
boot system flash bootflash:isr4300-universalk9.16.09.08.SPA.bin
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
 address-family ipv6
 exit-address-family
no aaa new-model
ip dhcp excluded-address 10.4.20.129
ip dhcp excluded-address 10.4.20.254
ip dhcp excluded-address 10.4.20.253
ip dhcp excluded-address 10.0.10.1
ip dhcp excluded-address 10.0.10.254
ip dhcp pool webuidhcp
ip dhcp pool USER-WLAN
 network 10.0.10.0 255.255.255.0
```

```
 default-router 10.0.10.1
 dns-server 8.8.8.8
ip dhcp pool AP-MANAGEMENT
 network 10.4.20.128 255.255.255.128
 default-router 10.4.20.1
 option 43 hex f104.0a04.14fe
login on-success log
subscriber templating
multilink bundle-name authenticated
crypto pki trustpoint TP-self-signed-2219300048
 enrollment selfsigned
 subject-name cn=IOS-Self-Signed-Certificate-2219300048
 revocation-check none
 rsakeypair TP-self-signed-2219300048
license udi pid ISR4321/K9 sn FLM240607T3
no license smart enable
diagnostic bootup level minimal
spanning-tree extend system-id
redundancy
 mode none
interface GigabitEthernet0/0/0
 ip address dhcp
 ip nat outside
 no shut
 negotiation auto
interface GigabitEthernet0/0/1
 no ip address
 no shut
 negotiation auto
interface GigabitEthernet0/0/1.10
 encapsulation dot1Q 10
 ip address 10.0.10.1 255.255.255.0
 ip nat inside
interface GigabitEthernet0/0/1.420
 encapsulation dot1Q 420
 ip address 10.4.20.129 255.255.255.128
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 shutdown
 negotiation auto
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
ip tftp source-interface GigabitEthernet0
ip nat inside source list 1 interface GigabitEthernet0/0/0 overload
access-list 1 permit 10.0.10.0 0.0.0.255
control-plane
line con 0
 transport input none
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
end
```

Switch

```
Switch#show running-config
no service pad
```

```
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
hostname Switch
boot-start-marker
boot-end-marker
no aaa new-model
system mtu routing 1500
vtp domain CCNP
vtp mode transparent
authentication mac-move permit
ip subnet-zero
spanning-tree mode pvst
spanning-tree etherchannel guard misconfig
spanning-tree extend system-id
vlan internal allocation policy ascending
vlan 10
 name USER-WLAN
vlan 11
 name Voice_Vlan
vlan 20
 name Data
vlan 30
 name MGT
vlan 40
 name MISC
vlan 50
 name NATIVE
vlan 420
 name MANAGEMENT
interface FastEthernet0/1
 switchport access vlan 10
interface FastEthernet0/2
 switchport access vlan 10
interface FastEthernet0/3
 switchport access vlan 10
interface FastEthernet0/4
 switchport access vlan 10
interface FastEthernet0/5
 switchport access vlan 10
interface FastEthernet0/6
 switchport access vlan 10
interface FastEthernet0/7
 switchport access vlan 10
interface FastEthernet0/8
 switchport access vlan 10
interface FastEthernet0/9
 switchport access vlan 10
interface FastEthernet0/10
 switchport access vlan 10
interface FastEthernet0/11
 switchport access vlan 10
interface FastEthernet0/12
 switchport access vlan 10
interface FastEthernet0/13
 switchport access vlan 420
interface FastEthernet0/14
 switchport access vlan 420
interface FastEthernet0/15
 switchport access vlan 420
interface FastEthernet0/16
 switchport access vlan 420
interface FastEthernet0/17
```

```
 switchport access vlan 420
interface FastEthernet0/18
 switchport access vlan 420
interface FastEthernet0/19
 switchport access vlan 420
interface FastEthernet0/20
 switchport access vlan 420
interface FastEthernet0/21
 switchport access vlan 420
interface FastEthernet0/22
 switchport access vlan 420
interface FastEthernet0/23
 switchport access vlan 420
interface FastEthernet0/24
 switchport trunk encapsulation dot1q
 switchport mode trunk
interface GigabitEthernet0/1
 switchport trunk encapsulation dot1q
 switchport mode trunk
interface GigabitEthernet0/2
interface Vlan1
 no ip address
interface Vlan10
 ip address 10.0.10.2 255.255.255.0
interface Vlan420
 ip address 10.4.20.253 255.255.255.128
ip classless
ip http server
ip sla enable reaction-alerts
line con 0
line vty 5 15
end
```

Wireless LAN Controller

```
(Cisco Controller) >show run-config commands
 802.11a 11nSupport a-mpdu tx scheduler enable
 802.11a 11nSupport a-mpdu tx scheduler timeout rt 10
 802.11a 11nSupport a-mpdu tx scheduler timeout nrt 200
 802.11a 11nSupport a-msdu max-subframes 3
 802.11b 11nSupport a-msdu max-subframes 3
 802.11a 11nSupport a-msdu max-length 8k
 802.11b 11nSupport a-msdu max-length 8k
 802.11a 11nSupport mcs tx 8 disable
 802.11a 11nSupport mcs tx 9 disable
 802.11a beacon range 0
 802.11a rx-sop threshold auto default
 802.11a cca threshold 0 default
 802.11a multicast buffer 0
 802.11a multicast data-rate 0 default
 802.11a cac video cac-method static
 802.11a max-clients 200
 802.11a dfs-peakdetect enable
 802.11b 11nSupport a-mpdu tx scheduler enable
 802.11b 11nSupport a-mpdu tx scheduler timeout rt 10
 802.11b 11nSupport a-mpdu tx scheduler timeout nrt 200
 802.11b beacon range 0
 802.11b rx-sop threshold auto default
 802.11b cca threshold 0 default
 802.11b multicast buffer 0
 802.11b multicast data-rate 0 default
 802.11b cac video cac-method static
```

```
 802.11b max-clients 200
802.11h channelswitch enable loud
 aaa auth mgmt  local radius
 acl url-acl disable
 flexconnect fallback-radio-shut disable
 advanced 802.11a channel dca interval 0
 advanced 802.11a channel dca startup-interval 0
 advanced 802.11a channel dca anchor-time 0
 advanced 802.11a channel dca chan-width 20
 advanced 802.11a channel dca best-width-max 80
 advanced 802.11a channel dca sensitivity 15
 advanced 802.11a channel dca min-metric -95
 advanced 802.11a channel delete 20
 advanced 802.11a channel delete 26
 advanced 802.11a channel delete 100
 advanced 802.11a channel delete 104
 advanced 802.11a channel delete 108
 advanced 802.11a channel delete 112
 advanced 802.11a channel delete 116
 advanced 802.11a channel delete 120
 advanced 802.11a channel delete 124
 advanced 802.11a channel delete 128
 advanced 802.11a channel delete 132
 advanced 802.11a channel delete 136
 advanced 802.11a channel delete 140
 advanced 802.11a channel delete 144
 advanced 802.11b channel dca interval 0
 advanced 802.11b channel dca startup-interval 0
 advanced 802.11b channel dca anchor-time 0
 advanced 802.11b channel dca sensitivity 10
 advanced 802.11b channel dca min-metric -95
 location info rogue extended
 location rssi-half-life tags 0
 location rssi-half-life client 0
 location rssi-half-life rogue-aps 0
 location expiry tags 5
 location expiry client 5
 location expiry calibrating-client 5
 location expiry rogue-aps 5
 advanced 802.11b client-network-preference default
 advanced 802.11a client-network-preference default
 advanced backup-controller primary
 advanced backup-controller secondary
 advanced backup-controller
 advanced backup-controller
 advanced sip-snooping-ports 0 0
 advanced eap bcast-key-interval 3600
 advanced 802.11-abgn pak-rssi-location threshold -100
 advanced 802.11-abgn pak-rssi-location trigger-threshold 10
 advanced 802.11-abgn pak-rssi-location reset-threshold 8
 advanced 802.11-abgn pak-rssi-location ntp 17.13.78.16
 advanced 802.11-abgn pak-rssi-location timeout 3
 advanced hotspot cmbk-delay 1
 ap cert-expiry-ignore mic enable
 ap syslog host global ::
 ap dtls-cipher-suite RSA-AES128-SHA
 ap dtls-wlc-mic sha2
 auth-list add mic 88:5a:92:bd:7d:df
 cdp advertise-v2 enable
 cts sxp disable
 cts sxp connection default password ****
 cts sxp retry period 120
 cts sxp sxpversion 2
```

```
 custom-web webmessage ""
 database size 2048
 dhcp opt-82 remote-id ap-mac
 qos qosmap disable
 qos qosmap trust-dscp-upstream disable
 flexconnect group default-flex-group add
 flexconnect group default-flex-group radius ap server-key <hidden>
 flexconnect group default-flex-group radius ap authority id
436973636f00000000000000000000000
 flexconnect group default-flex-group radius ap authority info Cisco A_ID
 flexconnect group default-flex-group http-proxy ip-address 0.0.0.0 http-proxy port 0
 flexconnect group default-flex-group template-vlan-map add none
 local-auth method fast server-key ****
interface create user-wlan 10
interface address management 10.4.20.254 255.255.255.128 10.4.20.129
interface address dynamic-interface user-wlan 10.0.10.254 255.255.255.0 10.0.10.1
interface address virtual 192.0.2.1
interface dhcp  management primary 10.0.0.1
interface vlan management 420
interface vlan user-wlan 10
interface nasid none user-wlan
 nasid apgroup  default-group
 wlan nasid none 1
 wlan nasid none 2
interface port management 13
interface port user-wlan 13
 mdns snooping disable
 mdns policy service-group create "default-mdns-policy" "Default Access Policy created
by WLC"
 mdns policy service-group user-role add default-mdns-policy admin
 mdns profile create "default-mdns-profile"
 mdns service create "AirPrint" _ipp._tcp.local. origin All LSS disable query enable
 mdns service create "AirTunes" _raop._tcp.local. origin All LSS disable query enable
 mdns service create "AppleTV" _airplay._tcp.local. origin All LSS disable query
enable
 mdns service create "HP_Photosmart_Printer_1" _universal._sub._ipp._tcp.local. origin
All LSS disable query enable
 mdns service create "HP_Photosmart_Printer_2" _cups._sub._ipp._tcp.local. origin All
LSS disable query enable
 mdns service create "Printer" _printer._tcp.local. origin All LSS disable query
enable
 mdns profile service add "default-mdns-profile" "AirPrint"
 mdns profile service add "default-mdns-profile" "AirTunes"
 mdns profile service add "default-mdns-profile" "AppleTV"
 mdns profile service add "default-mdns-profile" "HP_Photosmart_Printer_1"
 mdns profile service add "default-mdns-profile" "HP_Photosmart_Printer_2"
 mdns profile service add "default-mdns-profile" "Printer"
 mdns query interval 15
 wlan mdns enable 1
 wlan mdns enable 2
 wlan mdns profile 1 "default-mdns-profile"
 wlan mdns profile 2 "default-mdns-profile"
 ipv6 ra-guard ap enable
 ipv6 capwap udplite enable all
 ipv6 multicast mode unicast
 lag enable
 load-balancing aggressive enable
 load-balancing window 5
 wlan apgroup add default-group
 wlan apgroup qinq tagging eap-sim-aka default-group enable
 wlan apgroup interface-mapping add default-group 1 management
 wlan apgroup interface-mapping add default-group 2 user-wlan
 wlan apgroup nac-snmp disable default-group 1
```

```
 wlan apgroup nac-snmp disable default-group 2
 memory monitor errors enable
 memory monitor leak thresholds 10000 30000
Outdoor Mesh Ext.UNII B Domain channels: Disable
mesh security rad-mac-filter disable
mesh security rad-mac-filter disable
 mesh security eap
mesh background-scanning disable
mesh backhaul rrm disable
mesh backhaul rrm auto-rf global
mesh lsc advanced ap-provision open-window enable
 mgmtuser add cisco **** read-write
 mgmtuser termination-interval 0
 mobility dscp 0
 network http-proxy ip 0.0.0.0
 network http-proxy Port 80
 network webmode enable
 network secureweb csrfcheck enable
 network multicast igmp snooping enable
 network multicast mld snooping enable
 network profiling http-port 80
 network ap-priority disabled
 network rf-network-name default
 network client-ip-conflict-detection disable
 pmipv6 mag binding maximum AP 250
 qos protocol-type bronze dot1p
 qos protocol-type silver dot1p
 qos protocol-type gold dot1p
 qos protocol-type platinum dot1p
 qos priority bronze background background background
 qos priority gold video video video
 qos priority platinum voice voice voice
 qos priority silver besteffort besteffort besteffort
 qos dot1p-tag silver 0
 qos dot1p-tag gold 4
 qos dot1p-tag platinum 5
 radius callStationIdType macaddr
 radius auth callStationIdType ap-macaddr-ssid
 radius fallback-test mode passive
 radius fallback-test username cisco-probe
 radius fallback-test interval 300
 radius dns disable
 radius dns auth network enable
 radius dns auth management enable
 radius dns acct network enable
 radius dns auth rfc3576 disable
 tacacs dns disable
 rogue detection report-interval 10
 rogue detection min-rssi -90
 rogue detection transient-rogue-interval 0
 rogue detection client-threshold 0
 rogue detection security-level custom
 rogue ap aaa-auth disable
 rogue ap aaa-auth polling-interval 0
 rogue ap ssid alarm
 rogue ap valid-client alarm
 rogue adhoc enable
 rogue adhoc alert
 rogue ap rldp disable
 rogue ap rldp schedule disable
 rogue auto-contain level 1
 rogue containment flex-connect disable
 rogue containment auto-rate disable
```

```
rogue client aaa disable
rogue client mse disable
snmp version v2c enable
snmp version v3 enable
snmp snmpEngineId 0000376300005e200101fea9
switchconfig strong-pwd case-check enabled
switchconfig strong-pwd consecutive-check enabled
switchconfig strong-pwd default-check enabled
switchconfig strong-pwd username-check enabled
switchconfig strong-pwd position-check disabled
switchconfig strong-pwd case-digit-check disabled
switchconfig strong-pwd minimum upper-case 0
switchconfig strong-pwd minimum lower-case 0
switchconfig strong-pwd minimum digits-chars 0
switchconfig strong-pwd minimum special-chars 0
switchconfig strong-pwd min-length 3
sysname THEREALCONTROLLER
stats-timer realtime 5
stats-timer normal 180
tacacs fallback-test interval 0
rf-profile create 802.11a High-Client-Density-802.11a
rf-profile create 802.11b High-Client-Density-802.11bg
rf-profile create 802.11a Low-Client-Density-802.11a
rf-profile create 802.11b Low-Client-Density-802.11bg
rf-profile create 802.11a Typical-Client-Density-802.11a
rf-profile create 802.11b Typical-Client-Density-802.11bg
rf-profile tx-power-min 7 High-Client-Density-802.11a
rf-profile tx-power-min 7 High-Client-Density-802.11bg
rf-profile tx-power-control-thresh-v1 -65 High-Client-Density-802.11a
rf-profile tx-power-control-thresh-v1 -60 Low-Client-Density-802.11a
rf-profile tx-power-control-thresh-v1 -65 Low-Client-Density-802.11bg
rf-profile data-rates 802.11a disabled 6 High-Client-Density-802.11a
rf-profile data-rates 802.11a disabled 9 High-Client-Density-802.11a
rf-profile data-rates 802.11a mandatory 12 High-Client-Density-802.11a
rf-profile data-rates 802.11a supported 18 High-Client-Density-802.11a
rf-profile data-rates 802.11a mandatory 24 High-Client-Density-802.11a
rf-profile data-rates 802.11a supported 36 High-Client-Density-802.11a
rf-profile data-rates 802.11a supported 48 High-Client-Density-802.11a
rf-profile data-rates 802.11a supported 54 High-Client-Density-802.11a
rf-profile data-rates 802.11b disabled 1 High-Client-Density-802.11bg
rf-profile data-rates 802.11b disabled 2 High-Client-Density-802.11bg
rf-profile data-rates 802.11b disabled 5.5 High-Client-Density-802.11bg
rf-profile data-rates 802.11b disabled 11 High-Client-Density-802.11bg
rf-profile data-rates 802.11b disabled 6 High-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 9 High-Client-Density-802.11bg
rf-profile data-rates 802.11b mandatory 12 High-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 18 High-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 24 High-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 36 High-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 48 High-Client-Density-802.11bg
rf-profile data-rates 802.11a mandatory 6 Low-Client-Density-802.11a
rf-profile data-rates 802.11a supported 9 Low-Client-Density-802.11a
rf-profile data-rates 802.11a mandatory 12 Low-Client-Density-802.11a
rf-profile data-rates 802.11a supported 18 Low-Client-Density-802.11a
rf-profile data-rates 802.11a mandatory 24 Low-Client-Density-802.11a
rf-profile data-rates 802.11a supported 36 Low-Client-Density-802.11a
rf-profile data-rates 802.11a supported 48 Low-Client-Density-802.11a
rf-profile data-rates 802.11a supported 54 Low-Client-Density-802.11a
rf-profile data-rates 802.11b mandatory 1 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b mandatory 2 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b mandatory 5.5 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b mandatory 11 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 6 Low-Client-Density-802.11bg
```

```
rf-profile data-rates 802.11b supported 9 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 12 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 18 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 24 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 36 Low-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 48 Low-Client-Density-802.11bg
rf-profile data-rates 802.11a mandatory 6 Typical-Client-Density-802.11a
rf-profile data-rates 802.11a supported 9 Typical-Client-Density-802.11a
rf-profile data-rates 802.11a mandatory 12 Typical-Client-Density-802.11a
rf-profile data-rates 802.11a supported 18 Typical-Client-Density-802.11a
rf-profile data-rates 802.11a mandatory 24 Typical-Client-Density-802.11a
rf-profile data-rates 802.11a supported 36 Typical-Client-Density-802.11a
rf-profile data-rates 802.11a supported 48 Typical-Client-Density-802.11a
rf-profile data-rates 802.11a supported 54 Typical-Client-Density-802.11a
rf-profile data-rates 802.11b disabled 1 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b disabled 2 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b disabled 5.5 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b disabled 11 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b disabled 6 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 9 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b mandatory 12 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 18 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 24 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 36 Typical-Client-Density-802.11bg
rf-profile data-rates 802.11b supported 48 Typical-Client-Density-802.11bg
rf-profile rx-sop threshold medium High-Client-Density-802.11a
rf-profile rx-sop threshold medium High-Client-Density-802.11bg
rf-profile rx-sop threshold low Low-Client-Density-802.11a
rf-profile rx-sop threshold low Low-Client-Density-802.11bg
rf-profile coverage data -90 Low-Client-Density-802.11a
rf-profile coverage data -90 Low-Client-Density-802.11bg
rf-profile coverage voice -90 Low-Client-Density-802.11a
rf-profile coverage voice -90 Low-Client-Density-802.11bg
rf-profile coverage exception 2 Low-Client-Density-802.11a
rf-profile coverage exception 2 Low-Client-Density-802.11bg
rf-profile channel delete 20 High-Client-Density-802.11a
rf-profile channel delete 26 High-Client-Density-802.11a
rf-profile channel delete 100 High-Client-Density-802.11a
rf-profile channel delete 104 High-Client-Density-802.11a
rf-profile channel delete 108 High-Client-Density-802.11a
rf-profile channel delete 112 High-Client-Density-802.11a
rf-profile channel delete 116 High-Client-Density-802.11a
rf-profile channel delete 120 High-Client-Density-802.11a
rf-profile channel delete 124 High-Client-Density-802.11a
rf-profile channel delete 128 High-Client-Density-802.11a
rf-profile channel delete 132 High-Client-Density-802.11a
rf-profile channel delete 136 High-Client-Density-802.11a
rf-profile channel delete 140 High-Client-Density-802.11a
rf-profile channel delete 144 High-Client-Density-802.11a
rf-profile channel delete 20 Low-Client-Density-802.11a
rf-profile channel delete 26 Low-Client-Density-802.11a
rf-profile channel delete 100 Low-Client-Density-802.11a
rf-profile channel delete 104 Low-Client-Density-802.11a
rf-profile channel delete 108 Low-Client-Density-802.11a
rf-profile channel delete 112 Low-Client-Density-802.11a
rf-profile channel delete 116 Low-Client-Density-802.11a
rf-profile channel delete 120 Low-Client-Density-802.11a
rf-profile channel delete 124 Low-Client-Density-802.11a
rf-profile channel delete 128 Low-Client-Density-802.11a
rf-profile channel delete 132 Low-Client-Density-802.11a
rf-profile channel delete 136 Low-Client-Density-802.11a
rf-profile channel delete 140 Low-Client-Density-802.11a
rf-profile channel delete 144 Low-Client-Density-802.11a
```

```
rf-profile channel delete 20 Typical-Client-Density-802.11a
rf-profile channel delete 26 Typical-Client-Density-802.11a
rf-profile channel delete 100 Typical-Client-Density-802.11a
rf-profile channel delete 104 Typical-Client-Density-802.11a
rf-profile channel delete 108 Typical-Client-Density-802.11a
rf-profile channel delete 112 Typical-Client-Density-802.11a
rf-profile channel delete 116 Typical-Client-Density-802.11a
rf-profile channel delete 120 Typical-Client-Density-802.11a
rf-profile channel delete 124 Typical-Client-Density-802.11a
rf-profile channel delete 128 Typical-Client-Density-802.11a
rf-profile channel delete 132 Typical-Client-Density-802.11a
rf-profile channel delete 136 Typical-Client-Density-802.11a
rf-profile channel delete 140 Typical-Client-Density-802.11a
rf-profile channel delete 144 Typical-Client-Density-802.11a
rf-profile client-network-preference default High-Client-Density-802.11a
rf-profile client-network-preference default High-Client-Density-802.11bg
rf-profile client-network-preference default Low-Client-Density-802.11a
rf-profile client-network-preference default Low-Client-Density-802.11bg
rf-profile client-network-preference default Typical-Client-Density-802.11a
rf-profile client-network-preference default Typical-Client-Density-802.11bg
trapflags client nac-alert enable
trapflags client webAuthUserLogin enable
trapflags client webAuthUserLogout enable
trapflags ap ssidKeyConflict disable
trapflags ap timeSyncFailure disable
trapflags mfp disable
trapflags adjchannel-rogueap disable
trapflags mesh excessive hop count disable
trapflags mesh sec backhaul change disable
trapflags mesh psk auth failure disable
wlan create 1 Test1 Test
wlan create 2 user-wlan CONNECTTOME
wlan nac snmp disable 1
wlan nac snmp disable 2
wlan nac radius disable 1
wlan nac radius disable 2
wlan interface 2 user-wlan
wlan multicast interface 1 disable
wlan multicast interface 2 disable
wlan band-select allow disable 1
wlan band-select allow disable 2
wlan load-balance allow disable 1
wlan load-balance allow disable 2
wlan assisted-roaming prediction disable 1
wlan assisted-roaming prediction disable 2
wlan assisted-roaming neighbor-list disable 1
wlan assisted-roaming neighbor-list enable 2
wlan assisted-roaming dual-list disable 1
wlan assisted-roaming dual-list disable 2
wlan dms enable 2
wlan bssmaxidle enable 1
wlan bssmaxidle enable 2
wlan bss-transition enable 2
wlan bss-transition disassociation-imminent timer 200 1
wlan bss-transition disassociation-imminent timer 200 2
wlan bss-transition disassociation-imminent oproam-timer 40 1
wlan bss-transition disassociation-imminent oproam-timer 40 2
wlan multicast buffer disable 0 1
wlan multicast buffer disable 0 2
wlan session-timeout 1 1800
wlan session-timeout 2 1800
wlan flexconnect local-switching 1 disable
wlan flexconnect local-switching 2 disable
```

```
 wlan flexconnect learn-ipaddr 1 enable
 wlan flexconnect learn-ipaddr 2 enable
 wlan wgb broadcast-tagging disable 1
 wlan wgb broadcast-tagging disable 2
 wlan security wpa disable 2
 wlan security splash-page-web-redir disable 1
 wlan security splash-page-web-redir disable 2
 wlan security wpa akm 802.1x enable 1
 wlan security wpa akm cckm timestamp-tolerance  1000 1
 wlan security wpa akm cckm timestamp-tolerance  1000 2
 wlan security ft adaptive enable 2
 wlan security wpa gtk-random disable 1
 wlan security wpa gtk-random disable 2
 wlan security pmf association-comeback 1 1
 wlan security pmf association-comeback 1 2
 wlan security pmf saquery-retrytimeout 200 1
 wlan security pmf saquery-retrytimeout 200 2
 wlan dhcp_server 2 10.0.10.1 required
 wlan profiling radius dhcp disable 1
 wlan profiling radius http disable 1
 wlan profiling radius dhcp disable 2
 wlan profiling radius http disable 2
 wlan enable 1
 wlan enable 2
 license boot base
 WMM-AC disabled
 HS2 QOS disabled
 coredump disable
media-stream multicast-direct disable
media-stream message url
media-stream message email
media-stream message phone
media-stream message note denial
media-stream message state disable
802.11a media-stream multicast-direct enable
802.11b media-stream multicast-direct enable
802.11a media-stream multicast-direct radio-maximum 0
802.11b media-stream multicast-direct radio-maximum 0
802.11a media-stream multicast-direct client-maximum 0
802.11b media-stream multicast-direct client-maximum 0
802.11a media-stream multicast-direct admission-besteffort disable
802.11b media-stream multicast-direct admission-besteffort disable
802.11a media-stream video-redirect enable
802.11b media-stream video-redirect enable
 ipv6 neighbor-binding timers reachable-lifetime 300
 ipv6 neighbor-binding timers stale-lifetime 86400
 ipv6 neighbor-binding timers down-lifetime 30
 ipv6 neighbor-binding ra-throttle disable
 ipv6 neighbor-binding ra-throttle allow at-least 1 at-most 1
 ipv6 neighbor-binding ra-throttle max-through 10
 ipv6 neighbor-binding ra-throttle throttle-period 600
 ipv6 neighbor-binding ra-throttle interval-option passthrough
 ipv6 ns-mcast-fwd disable
 ipv6 na-mcast-fwd enable
 ipv6 enable
 nmheartbeat disable
 ipv6 slaac service-port disable
 sys-nas
 tunnel eogre heart-beat interval 60
 tunnel eogre heart-beat primary-fallback-timeout 30
 tunnel eogre heart-beat max-skip-count 3
 wlan mobility selective-reanchoring disable  1
 wlan mobility selective-reanchoring disable  2
```

```
 cloud-services wsa mode Disable
network assurance server url
 WSA Backhaul SSID
 WSA Backhaul Username
 WSA Backhaul Authentication Type psk
Eap Type ..................................... <none>
cloud-services server url
 cloud-services cmx disabled
```

Conclusion

While it may have taken some time familiarizing myself with the features and configurations of a Wireless LAN Controller, I will say the payoff is worth the effort when managing many access points. With only two access points in my arsenal, it would likely have been better to configure them standalone in a production environment. However, this was not for a production environment, as I was configuring the WLC for educational purposes.

CUCME

Configuring the Cisco Unified Communicaions Manager Express to manage a VoIP Call Session

Purpose

When I want to talk to an English friend remotely from Seattle, I can easily call them and receive near instantaneous audio transmission, if I am connected to the internet. *Voice over Internet Protocol* (VoIP) allows voice to transmit over the internet instead of the old-fashioned copper phone lines. With the internet expanding ever so vastly into the average person's life, especially during the pandemic, it only seemed natural to take a gander behind the scenes of VoIP.

Background Information

Why do we use VoIP?

Originally, copper landlines were used as the media transmitting incoming and outgoing calls. Since then, advancements have been made in networking allowing voice to be sent as packets over the internet. While some might relate VoIP to on-premises hardware – the classic, physical phones typically used in offices for marketing – other software, such as Skype or Discord, both incorporate VoIP to manage calls.

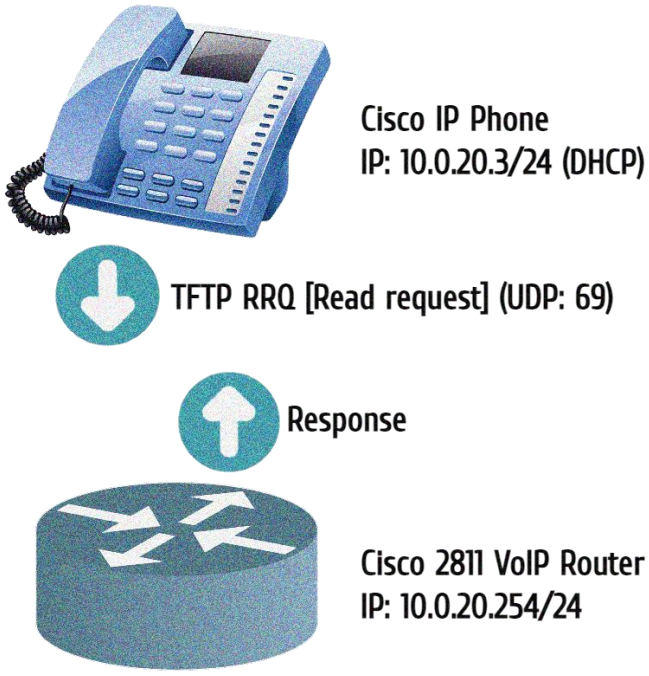
People opt for VoIP since a connection to the internet is the only medium needed for conference. Other infrastructure, such as extra copper phone lines, is not necessary. Unlike landline phones, VoIP is capable of high-definition phone calls.

What is the Cisco Unified Communications Manager?

The Cisco Unified Communications Manager (CUCM) functions as a controller for IP-based communications. In other words, calls begin, are maintained, and end with the CUCM. While the CUCM can manage calls, it also hosts files for IP phones on the network for when they initially boot up. There are two ways of configuring VoIP with the CUCM: with the full CUCM that runs on a server, or the CUCM Express (CUCME) which runs on certain routers. The full CUCM has many services, such as call manager, dhcp service, tftp, and service analyzers. The CUCME is ported down to be runnable in a router. In this lab, I host a call between two IP phones using the CUCME running on the Cisco 2811 Integrated Services Router.

Trivial File Transfer Protocol

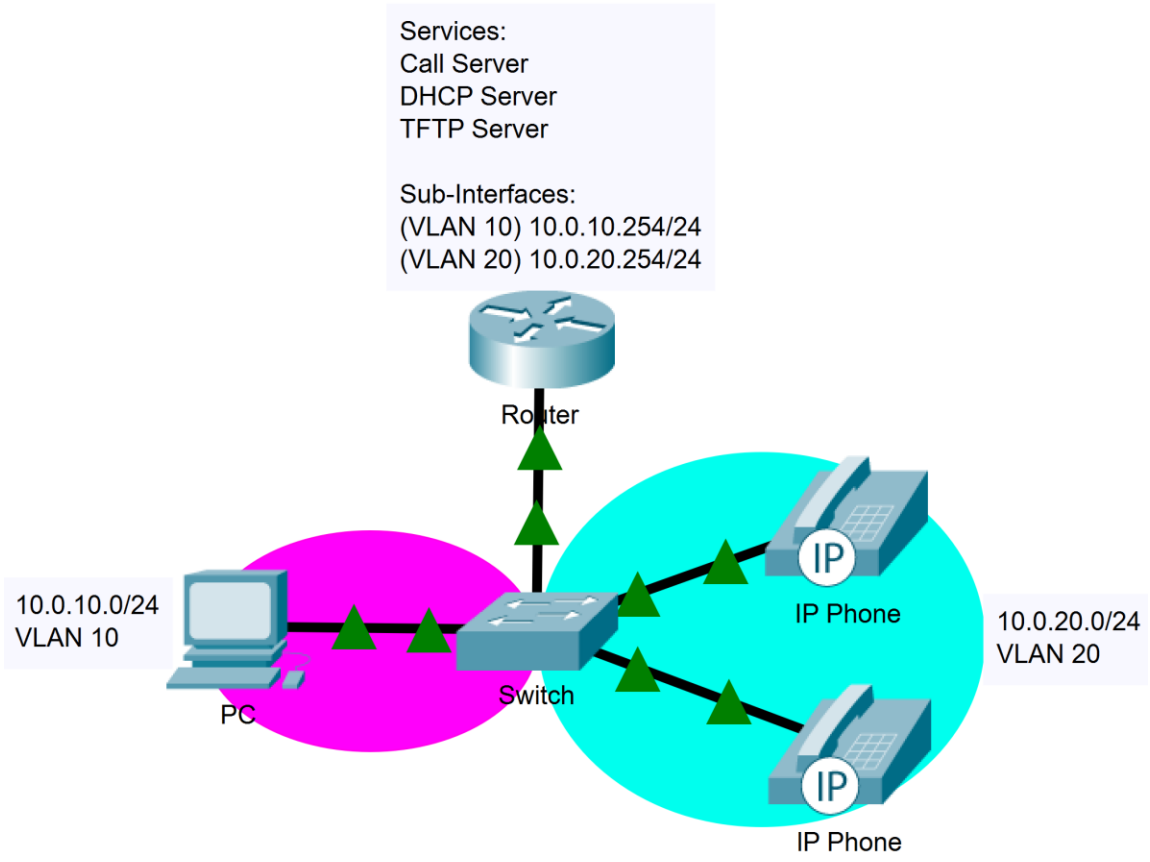
Trivial File Transfer Protocol (TFTP) allows a client to obtain a file from or upload a file to a remote host. Unlike File Transfer Protocol (FTP), which uses TCP ports 20 and 21, TFTP uses UDP port 69 to establish network connections. While some may question the application of UDP, the faster but less secure method of data communication, the reasoning becomes clearer if one considers the *triviality* of TFTP. TFTP is much simpler than FTP. Files hosted by a TFTP server are generally quite small; you are more likely to find TFTP used on a LAN while FTP is implemented on a WAN; network engineers use TFTP to distribute software across a LAN because packet loss is unlikely.



Why do we care about TFTP? Cisco SCCP IP phones attempt to download software on boot that we need to host. This process is known as bootstrapping. Bootstrapping allows machines that are new to the network, or machines that have lost everything, to download

software files which get them up and running. Cisco routers let engineers host files using TFTP. However, a Cisco router is not a fully functioning TFTP server, it can only serve files for download. Any file in the router's flash can be hosted, including the IP phone files we need.

Network Diagram



Process

The objective for this lab was to get two IP phones making and receiving calls from each other. Without knowing anything about VoIP or the CUCM, my first goal was to construct a topology with two IP phones and worry about the configurations later. I've found that, like math, visually plotting out the data is a great first start for any project. To construct the topology, I needed a basic grasp on how VoIP works with the CUCM.

I ended up referencing a YouTube video which provided general configurations for what I wanted to achieve. Following the video, I was able to design the topology above and create points of each major step needed to get VoIP running.

Steps

- Vlans
  - Create two vlans – one for VoIP and one for the computers.
  - Enable spanning tree portfast on the VoIP interfaces. While it is not necessary, portfast will reduce the delay a phone has when receiving its IPs and configuration files.
  - Set up the respective vlans and sub-interfaces on a VoIP router.
- DHCP
  - Create two DHCP pools – one per vlan. For the VoIP pool, define a DHCP option 150 address (the IP of your TFTP interface, likely the VoIP sub-interface address).
- CUCM (telephony-service)
  - Load an IP phone file.
  - Define the max ephones of the network.
  - Define the max number of virtual voice ports (*ephone-dn*).
  - Define the call server address (*ip source-address*).
  - Create a system message (optional).
- Ephone
  - Manually define each ephone connected to the network.
  - Define the mac address.
  - Define what some buttons on the IP phone do.
- TFTP
  - Host local IP Phone files on the router using *tftp-server*.
  - IP phones will request files from the interface defined in DHCP option 150.
- Create CNF files
  - Go back into the CUCM and create files (*create cnf-files*).

Problems

After creating the vlans and setting up DHCP, I was left with the new content: Telephony configuration. My DHCP service was working across both vlans, so up until this point, I had no problems. I defined the call server address in *telephony-service* and hosted *tftp-server* IP phone

files in global config. I decided to configure one ephone before configuring the other, assuming that both ephones would run by simply tweaking a working config. The difference between *ephone* and *ephone-dn* initially confused me, but I eventually realized ephone dial numbers (*ephone-dn*) "attach" to ephones. Once I created the ephone config by assigning a *mac-address* and an *ephone-dn*, the respective IP phone booted and received files. All I needed to do was replicate that config for my other IP phone.

Though I used the same ephone configuration – except for a different *mac-address* and *ephone-dn* – my second IP phone ran into an error. I kept receiving the message "TFTP Files Not Found". It was strange how one ephone was able to locate the files while the other was not, considering the TFTP server hosted the same files. Both IP phones were the same model, *Cisco 7940s*, so that could not be the issue.

Running through forums on the internet, I noticed a pattern in most *tftp-server* configurations: multiple files were listed. Some even had "alias's" that were directly referenced in *telephony-service*. For example, the command *tftp-server flash:P00308000500.bin alias P00308000500* in global configuration mode is then referenced by the command *load 7960-7940 P00308000500* in *telephony-service* mode. After adding more of these references to files in my configuration and using the *debug tftp events* command to show what files the IP phone is trying (but failing) to grab, I got the second IP phone to boot.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement <*Optional Argument*>	An optional argument or statement, not necessary for a command to function, denoted in italics

// IPv4 DHCP Configuration

Router(config)# **ip dhcp exclude-address [*initial ip*]** <*final ip*>

- Set an IP or range of IPs to exclude from the pool

If the network administrator so chooses to exclude a range of IP addresses, the range would be from the Initial IP *to the* End IP*, inclusive. The* End IP *argument is not necessary when excluding only one IP. Excludes are typically reserved for pre-configured static IP addresses, for example, interfaces on the router.*

Router(config)# **ip dhcp pool [*pool name*]**

- Creates a pool for distributing routing information

Dynamic Host Configuration Protocol (DHCP) automates the assignment of IP addresses to devices on the local network. A DHCP pool is used to define the range of IP addresses that the server will divvy out to clients.

Router(dhcp-config)# **network [*network address*] [*subnet mask*]**

- Configures a pool that distributes the specified subnet

Router(dhcp-config)# **default-router [*ip*]**

- Sends the specified default gateway to clients

Router(dhcp-config)# **option 150 ip [*call server address*]**

- Advertises the address of the TFTP server where IP phones will request files

Cisco IP phones download their files from a TFTP server. Once an IP phone receives DHCP information, gaining an IP, it will then attempt to download the files.

// VLAN Configuration

Switch(config)# **vlan [*id*]**

- Create a vlan with specified *id*

A Virtual Lan (vlan) is used to partition groups of devices on a switched network.

Switch(config-if)# **switchport mode access**

- Configure an interface to be in access mode

Only one vlan is permitted across an access mode interface. Devices connected to an access-configured interface are usually end devices, such as clients.

Switch(config-if)# **switchport access vlan [*id*]**

- Configure an interface to be part of a specified vlan

*To configure the same command access across multiple interfaces, use the command **interface range [interface] [start-end id]**.*

Switch(config-if)# **switchport trunk encapsulation dot1q**

- Configures the interface to use IEEE 802.1Q encapsulation on frames when the interface is configured as a trunk

The user must define the encapsulation before setting an interface as a trunk.

Switch(config-if)# **switchport mode trunk**

- Configure an interface to be a trunk

// Router Sub-Interface Configuration

Router(config)# **interface [*id*].[*vlan id*]**

- Enter sub-interface configuration mode

Sub-Interfaces are used in conjunction with vlans. To route vlans, a sub-interface must be created for each respective VLAN.

Router(config-subif)# **encapsulation dot1Q [*vlan id*]**

- Configures the interface to use IEEE 802.1Q encapsulation on frames

While 802.1Q is not the only networking standard supporting vlans, it is the most prominent. The vlan id should be a vlan existing on a connected switch desired to be routed.

Router(config-subif)# **ip address [*ip*] [*mask*]**

- Configure an IP for the sub-interface, just like any normal interface

// TFTP Server Configuration

Router(config)# **tftp-server [*local file address*]**

- Host local files on the router

// Telephony Service Configuration

Router(config)# **telephony-service**

- Enter the CUCM interface

Router(config-telephony)# **load [*ip phone version*] [*firmware name*]**

- Updates the Cisco IOS Telephony Services configuration file for a specific type of IP phone

Cisco IP phones update themselves with new phone firmware whenever they are initially started or reloaded. When a phone starts or reboots, the phone reads the configuration file to determine the name of the firmware file it should load and then looks for that firmware file on a TFTP server. The firmware name should be defined without a file extension.

Router(config-telephony)# **max-ephones [number]**

- Define the maximum number of SCCP phones that can register to the CUCM

Router(config-telephony)# **max-dn [number]**

- Define the max number of voice ports that can open

Router(config-telephony)# **ip source-address [*ip*] port [*voice port*]**

- Define the interface on the router that will act as the call server

The default voice port is 2000.

Router(config-telephony)# **create cnf-files**

- Generate configuration files for SCCP phones

// Ephone and Dial Number Configuration

Router(config)# **ephone-dn [*number*]**

- Enter the ephone dial number interface

Router(config-ephone-dn)# **number [*phone number*]**

- Set the phone number of the ephone

Configurations

Cisco 2811 VoIP Router

```
Router#show running-config
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
hostname Router
boot-start-marker
boot-end-marker
logging message-counter syslog
no aaa new-model
memory-size iomem 25
no network-clock-participate slot 1
dot11 syslog
ip source-route
ip cef
ip dhcp pool PC
   network 10.0.10.0 255.255.255.0
   default-router 10.0.10.254
ip dhcp pool VOIP
   network 10.0.20.0 255.255.255.0
   default-router 10.0.20.254
   option 150 ip 10.0.20.254
no ipv6 cef
multilink bundle-name authenticated
voice-card 0
 no dspfarm
voice-card 1
 no dspfarm
vtp domain CCNP
vtp mode transparent
archive
 log config
  hidekeys
vlan 2,10,20
interface FastEthernet0/0
 no ip address
 shutdown
 duplex auto
 speed auto
interface FastEthernet0/1
 no shutdown
 no ip address
 duplex auto
 speed auto
 h323-gateway voip interface
interface FastEthernet0/1.10
 encapsulation dot1Q 10
 ip address 10.0.10.254 255.255.255.0
interface FastEthernet0/1.20
 encapsulation dot1Q 20
 ip address 10.0.20.254 255.255.255.0
interface FastEthernet0/0/0
interface FastEthernet0/0/1
interface FastEthernet0/0/2
interface FastEthernet0/0/3
interface Serial0/1/0
 no ip address
```

```
 shutdown
interface Serial0/2/0
 no ip address
 shutdown
 clock rate 2000000
interface Serial0/2/1
 no ip address
 shutdown
 clock rate 2000000
interface Serial0/3/0
 no ip address
 shutdown
 clock rate 2000000
interface Serial0/3/1
 no ip address
 shutdown
 clock rate 2000000
interface Vlan1
 no ip address
ip forward-protocol nd
no ip http server
no ip http secure-server
tftp-server P00308000500.sbn
tftp-server P00308000500.loads
tftp-server flash:P00308000500.bin alias P00308000500
tftp-server flash:P00308000500.sb2
control-plane
voice-port 1/0/0
voice-port 1/0/1
voice-port 1/0/2
voice-port 1/0/3
voice-port 1/1/0
voice-port 1/1/1
telephony-service
 load 7960-7940 P00308000500
 max-ephones 10
 max-dn 10
 ip source-address 10.0.20.254 port 2000
 system message deez
 max-conferences 5 gain -6
 transfer-system full-consult
 create cnf-files version-stamp Jan 01 2002 00:00:00
ephone-dn  1
 number 911
ephone-dn  2
 number 420
ephone  1
 device-security-mode none
 mac-address 001B.D512.B06E
 button  1:1
ephone  2
 device-security-mode none
 mac-address 001B.D512.A57A
 button  2:2
line con 0
line aux 0
line vty 0 4
 login
scheduler allocate 20000 1000
end
```

PoE Switch

```
Switch#sh run
version 12.2
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
hostname Switch
boot-start-marker
boot-end-marker
no aaa new-model
system mtu routing 1500
vtp domain CCNP
vtp mode transparent
authentication mac-move permit
ip subnet-zero
crypto pki trustpoint TP-self-signed-1928519808
 enrollment selfsigned
 subject-name cn=IOS-Self-Signed-Certificate-1928519808
 revocation-check none
 rsakeypair TP-self-signed-1928519808
spanning-tree mode pvst
spanning-tree etherchannel guard misconfig
spanning-tree extend system-id
vlan internal allocation policy ascending
vlan 2
 name forleft
vlan 3
 name forright
vlan 4-5,7
vlan 10
 name PC
vlan 12
vlan 20
 name VOIP
vlan 30
 name Expedia
vlan 40
 name forty
vlan 99
vlan 100
 name Microsoft
vlan 192
 name Guest
vlan 996
 name CUSTOMER_NATIVE
interface FastEthernet0/1
 switchport access vlan 10
 switchport mode access
interface FastEthernet0/2
 switchport access vlan 10
 switchport mode access
interface FastEthernet0/3
 switchport access vlan 10
 switchport mode access
interface FastEthernet0/4
 switchport access vlan 10
 switchport mode access
interface FastEthernet0/5
 switchport access vlan 10
 switchport mode access
```

```
interface FastEthernet0/6
 switchport access vlan 20
 switchport mode access
 spanning-tree portfast
interface FastEthernet0/7
 switchport access vlan 20
 switchport mode access
 spanning-tree portfast
interface FastEthernet0/8
 switchport access vlan 20
 switchport mode access
 spanning-tree portfast
interface FastEthernet0/9
 switchport access vlan 20
 switchport mode access
 spanning-tree portfast
interface FastEthernet0/10
 switchport access vlan 20
 switchport mode access
 spanning-tree portfast
interface FastEthernet0/11
 switchport trunk encapsulation dot1q
 switchport mode trunk
interface FastEthernet0/12
interface FastEthernet0/13
interface FastEthernet0/14
interface FastEthernet0/15
interface FastEthernet0/16
interface FastEthernet0/17
interface FastEthernet0/18
interface FastEthernet0/19
interface FastEthernet0/20
interface FastEthernet0/21
interface FastEthernet0/22
interface FastEthernet0/23
interface FastEthernet0/24
interface FastEthernet0/25
interface FastEthernet0/26
interface FastEthernet0/27
interface FastEthernet0/28
interface FastEthernet0/29
interface FastEthernet0/30
interface FastEthernet0/31
interface FastEthernet0/32
interface FastEthernet0/33
interface FastEthernet0/34
interface FastEthernet0/35
interface FastEthernet0/36
interface FastEthernet0/37
interface FastEthernet0/38
interface FastEthernet0/39
interface FastEthernet0/40
interface FastEthernet0/41
interface FastEthernet0/42
interface FastEthernet0/43
interface FastEthernet0/44
interface FastEthernet0/45
interface FastEthernet0/46
interface FastEthernet0/47
interface FastEthernet0/48
interface GigabitEthernet0/1
interface GigabitEthernet0/2
interface GigabitEthernet0/3
```

```
interface GigabitEthernet0/4
interface Vlan1
 no ip address
 shutdown
ip classless
ip http server
ip http secure-server
ip sla enable reaction-alerts
line con 0
line vty 0 4
 login
line vty 5 15
 login
end
```

Conclusion

Considering the amount of time that I personally spend using VoIP services, digging deeper has been an intriguing experience which I hope to learn more about. I can speak with friends in England from America in almost real time. It is fascinating how we are thousands of miles apart but can remain close due to a protocol first released before I was born. Now I got to investigate that protocol, likely one of the most meaningful ones in our world today.

CUCM

Installing and Configuring the Cisco Unified Communicaions Manager on an ESXI Server

Purpose

When I want to talk to an English friend remotely from Seattle, I can easily call them and receive near instantaneous audio transmission, if I am connected to the internet. *Voice over Internet Protocol* (VoIP) allows voice to transmit over the internet instead of the old-fashioned copper phone lines. With the internet expanding ever so vastly into the average person's life, especially during the pandemic, it only seemed natural to take a gander behind the scenes of VoIP.

Background Information

Why do we use VoIP?

Originally, copper landlines were used as the media transmitting incoming and outgoing calls. Since then, advancements have been made in networking allowing voice to be sent as packets over the internet. Some might relate VoIP to on-premises hardware – the stereotypic, bulky physical phones used in offices for marketing – though modern software, such as Skype or Discord, use VoIP to manage calls.

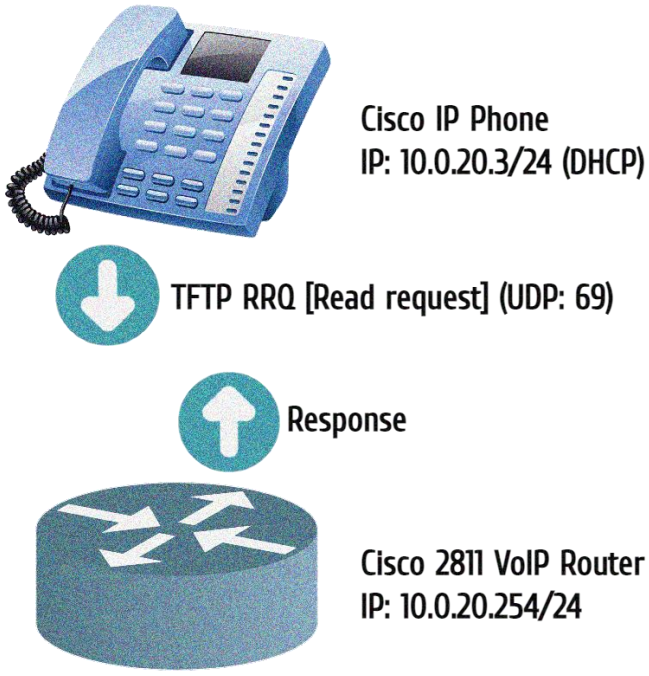
People opt for VoIP because a phone call only requires a connection to the internet. Other infrastructure, such as extra copper phone lines, is not necessary. Why go through the trouble of installing a landline-like phone when VoIP is so much simpler, capable of high-definition phone calls, and only requires 100 kbps of bandwidth?

What is the Cisco Unified Communications Manager?

The Cisco Unified Communications Manager (CUCM) is an IP-based communications system that allows users to contact their coworkers, customers, or friends through audio or video regardless of physical location. All calls begin, are maintained, and end with the CUCM. While the CUCM can manage calls, it also hosts files for IP phones on the network for when they initially boot up. There are two ways of configuring VoIP with the CUCM: with the full CUCM that runs on a server, or the CUCM Express (CUCME) which runs on certain Cisco routers. The full CUCM has many services, such as call manager, dhcp service, tftp, service analyzers, and others, granting all sorts of unique configurations for different purposes. The CUCME is ported down to be runnable in a router. In this lab, I host a call between two IP phones using the CUCM running on an ESXi server.

Trivial File Transfer Protocol

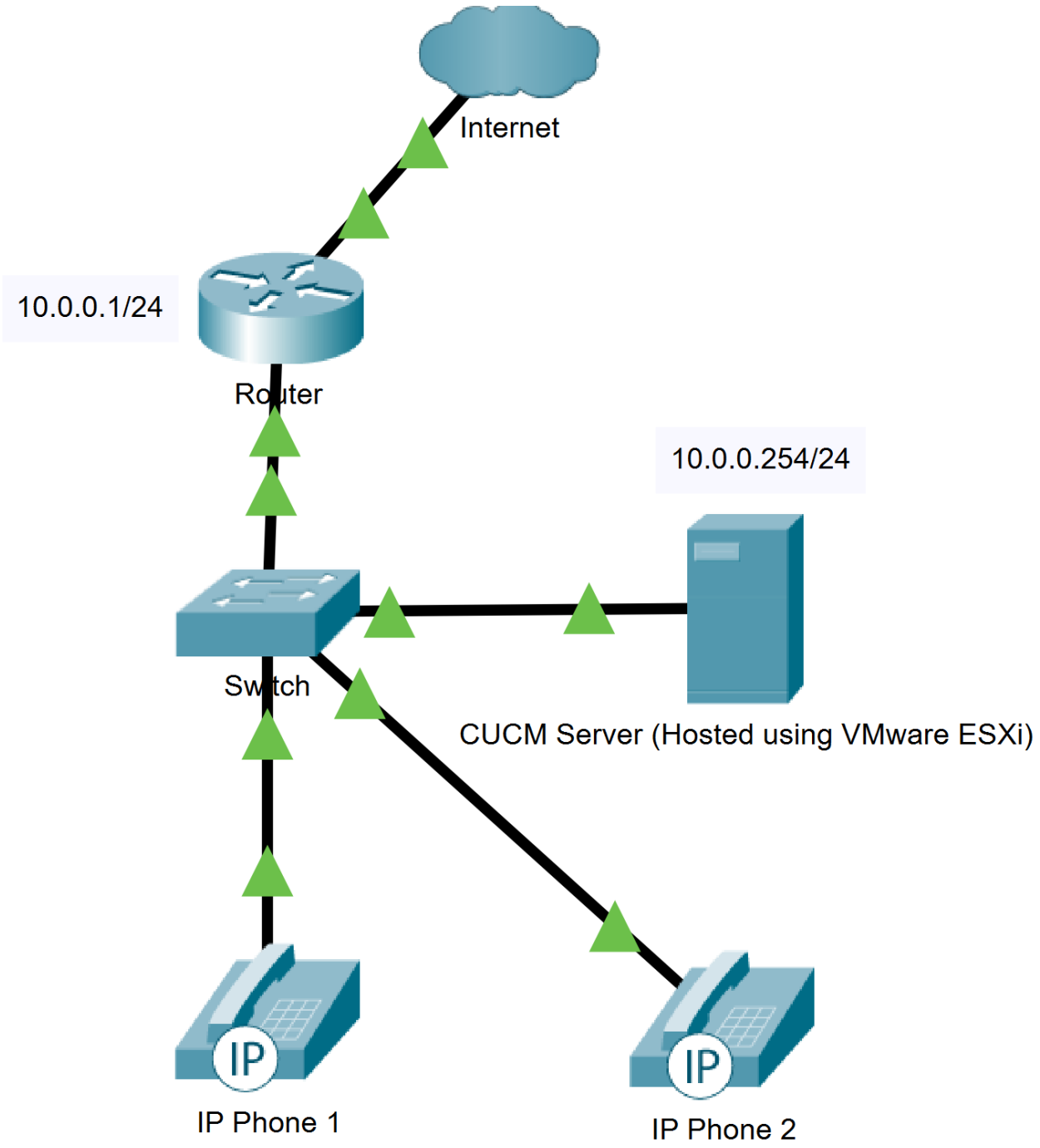
Trivial File Transfer Protocol (TFTP) allows a client to obtain a file from or upload a file to a remote host. Unlike File Transfer Protocol (FTP), which uses TCP ports 20 and 21, TFTP uses UDP port 69 to establish network connections. While some may question the application of UDP (the faster but less secure method of data communication) the reasoning becomes clearer if one considers the *triviality* of TFTP. TFTP is much simpler than FTP. Files hosted by a TFTP server are generally quite small; you are more likely to find TFTP used on a LAN while FTP is implemented on a WAN; network engineers use TFTP to distribute software across a LAN because packet loss is unlikely.



Why do we care about TFTP? Cisco SCCP IP phones attempt to download software on boot that we need to host. This process is known as bootstrapping. Bootstrapping allows

machines that are new to the network, or machines that have lost everything, to download software files which gets them up and running. We can use the TFTP service provided by the CUCM to host these files for IP Phones.

Network Diagram



Process

It took a while for me to breach the surface of this project since I was not clear on what I needed to do. Much of the struggle in the beginning could have been avoided if I asked for clarifications on certain things, such as a topology. Eventually I did. Having worked with the CUCME, the express version on a Cisco 2811 service router, I had insight on configurations that I likely needed to get working. For example, dial numbers and phone profiles. However, before I could do that, I needed to install the CUCM.

Installing the CUCM in VMware ESXi

The CUCM is an ISO, an operating system image, so you either need a physical device to boot it on or a Virtual Machine (VM) service such as ESXi. I chose to run the CUCM on ESXi to familiarize myself with the service (and conveniently we already had a server running it). I downloaded the ISO on the local ESXi server using TFTP and hastily created a VM.

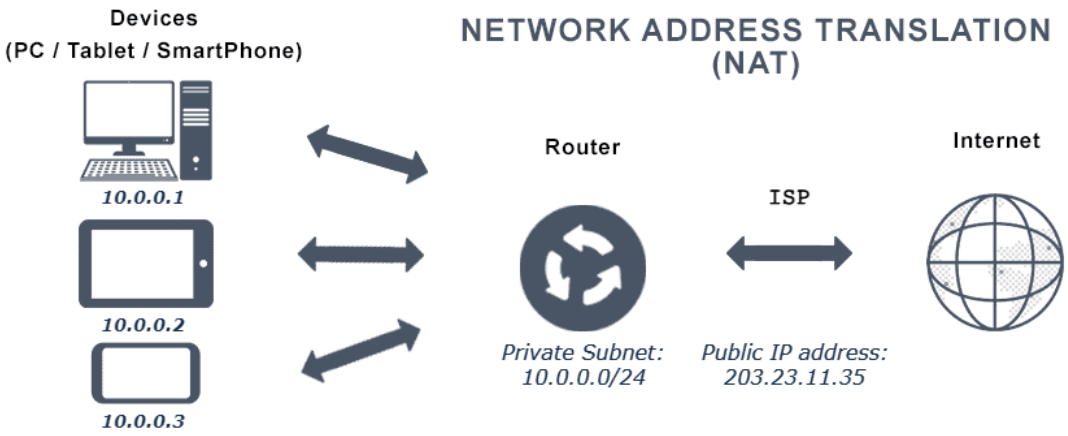
When creating a VM, there are several configurable options, such as type, RAM, and storage, but I left them default. The CUCM VM booted, shortly running into a "critical error": *there are no deployments provided by this software that support this hardware. Installation will now halt.* The solution was not too hard to find, I just needed to allocate more RAM and storage.

Initial Setup

With the CUCM as a runnable VM, my next step was to complete the initial setup: configuring settings such as addressing and defining administrative accounts. I left most settings as default. The trouble came a couple of pages later, when I was forced to define a *Network Time* (NTP) server.

I decided to configure my router as an NTP server, so I did a little research and found a command that should do just that: *ntp master*. Theoretically, *ntp master* is the only command needed for a router to forward its local clock, but the CUCM was not cooperating. Perhaps it was that my router had a high "stratum" number, and the CUCM would only accept responses from a more credible source. In NTP, "stratums" are a form of hierarchy amongst NTP servers where lower stratums send more accurate versions of time. For example, stratum one servers typically have 10 microseconds of accuracy. Stratum two servers get their time directly from stratum one servers but have more delay. Stratum three get theirs from stratum two, and such. Higher stratum servers are more common but have more delay.

My solution was to forget about creating a local NTP server and instead use one from *pool.ntp.org*, a free timeserver host with many pools across the world. However, the CUCM was in an internal part of my network, unable to access the NTP servers hosted on the internet. The fix here was simple: *NAT*. *Network Address Translation* (NAT) is a process that enables a unique IP address to represent an entire group of devices. Typically, this IP address is "Public", administered by one's ISP.



In this example, the public IP, *203.23.11.35*, is representing the *10.0.0.0/24* subnet. There can be multiple networks that use the same private subnet – what matters is that the public IP is unique. With so many devices nowadays, the total IPv4 address pool is running out. NAT helps conserve these addresses by translating multiple private IPs through one public IP. This is known as dynamic NAT.

Using my experience from previous labs, I was able to set up NAT relatively quickly with no issues. Once implemented, the CUCM could access the internet and request time from legitimate NTP servers. Now I could move on and finalize the initial setup. I left the CUCM to finalize over the weekend, as it takes a couple of hours to prepare after the initial setup.

Configuring the CUCM

Much like other labs, I compiled steps from tutorials online on how to configure what I want. In this case, it was registering two IP phones and starting a call. There was a tutorial online that provided a lot of steps, so I wanted to follow that through. Many of these steps were unnecessary for my purpose, but were good practice, so I decided to try and implement all of them. Here are the steps I compiled:

- Enter the CUCM Web Interface
- Under "Navigation" select *Cisco Unified Serviceability*
    - Under the "Tools" tab, select "Service Activation"
    - Activate all Services but DHCP (will activate this later)
    - [Save]
- Under "Navigation" select *Cisco Unified CM Administration*
    - Under the "System" tab, select "Server"
    - Click "Find" (Should find a Server) and select that one
        - Change the Host name to the CUCM IP address
        - [Save]
- Under the "System" tab, select *Enterprise Parameters*
  (Note that you can use "CTRL+F" to search for these parameters)
    - Change "Auto Registration Phone Protocol" to "SIP"
    - Find Phone URL Parameters and Secure Phone URL Parameters (There should be a bunch of URLs)
    - Change the Domain name of each URL to the IP of your CUCM
        - (For example, I would change the URL "http://The-real-CUCM.lol:8080/ccmcip/authenticate.jsp" to "http://10.0.0.254:8080/ccmcip/authenticate.jsp")

    - [Save]
    - [Reset]
- Under the "System" tab, select *Service Parameters*
    - Select your Server
    - Select the service "Cisco CallManager"
    - Change the delay of the "T302 Timer" to 3 seconds
        - (Note the unit is milliseconds)
    - Change the delay of the "H225 T302 Timer" to 3 seconds
        - (Note the unit is milliseconds)
    - Change each "Stop Routing" parameter to "False"
    - Change the "Default Interregion Max Audio Bit Rate" to "16 kbs (iLBC, G.728)"
    - Change "Automated Alternate Routing" to "True"
    - Change "Enable Enterprise Feature Access" to "True"
    - Change "Enable Mobile Voice Access" to "True"
    - Change "Matching Caller ID with Remote Destination" to "Partial Match"
    - Change "Number of Digits for Caller ID Partial Match" to "7"
    - [Save]
- Under the "System" tab, select *Phone NTP Reference*
    - Click "Add New"
    - Define an NTP server
    - Set the "Mode" to "Unicast"
- Under the "System" tab, select *Date/Time Group*
    - Click "Find"
        - (A default one should appear, likely named "CMLocal")
    - Configure the Name, Time Zone, and Date Formats
    - Select "Add Phone NTP References"
    - Add the NTP reference previously made
    - [Save]
    - Click [Copy] and repeat steps above for each Date/Time Group desired
- Under the "System" tab, select *Region Information/Region*
    - Click "Find" and select "Default"
    - Create new regions to match the ones in your Date/Time Group
- Under the "System" tab, select *Location Info/Location*
    - Click "Find"
    - Select create new Locations for each Date/Time Group made
- Under the "Call Routing" tab, select *Class of Control/Partition*
    - Click "Add New"
    - Separating each partition with a new line, enter the partitions: INTERNAL, PSTN, STRIP, CM-ANI-IN, CM-DNIS-OUT, and a partition for each Date/Time Group
    - [Save]
- Under the "Call Routing" tab, select *Class of Control/Calling Search Space*
    - Click "Add New"
    - Name: "INTERNAL", selected partitions: "INTERNAL"
    - [Save]
    - [Copy]
    - Name: "CM", selected partitions: "INTERNAL", "CMLocal"
    - [Save]
    - [Copy]
    - For each region, create partitions with "INTERNAL" and *Region
    - [Copy]
    - Name: "PSTN", selected partitions: "PSTN"
    - [Save]
    - [Copy]
    - Name: "CM-DNIS-OUT", selected partitions: "CM-DNIS-OUT"
    - [Save]
    - [Copy]
    - Name: "CM-ANI-IN", selected partitions: "CM-ANI-IN"
    - [Save]
- Under the "Media Resources" tab, select *Media Resource Group*
    - Click "Add New"

    - Name: "CM", selected media resources: *all of the available
    - [Save]
    - [Copy]
    - For each region, create a media resource group with all available media resources
- Under the "Media Resources" tab, select *Media Resource Group List*
    - Click "Add New"
    - For each media resource group, make a list
    - Example: Name: "CM", selected Media Resource Groups: "CM"
    - Under the "System" tab, select "Device Pool"
    - Click "Find" and select the default
    - Change "Device Pool Name" to "CM"
    - Change "Media Resource Group List" to "CM"
    - Change "Location" to "CMLocal"
    - [Save]
    - [Reset]
    - [Copy]
    - Repeat steps above for each Region previously created
- Under the "User Management" tab, select *End User*
    - Click "Add New"
    - Define a User ID
    - Define a Password
    - Define a PIN
    - Define Names (Last, First, Display names)
    - [Save]
    - Click "Add to Access Control Group" (Scroll down)
    - Click "Find" and select "Standard CCM End Users", "Standard CTI Enabled"
    - Click "Add Selected"
    - [Save]
    - Follow the steps above to create another end user
- Under the "Device" tab, select *Phone*
    - Click "Add New"
    - Enter the type of your IP phone
    - Enter "SIP" for the device protocol
    - Change "MAC Address" to match your IP Phone
    - Change "Device Pool" to "CM"
    - Change "Phone Button Template" to match your IP Phone
    - Change "Calling Search Space" to "CM"
    - Change "Owner User ID" to match a profile you created previously
    - Change "Device Security Profile" to match your IP Phone with SIP
    - Change "SIP Profile" to "Standard SIP Profile"
    - [Save]
    - A new section called "Association" should appear. Click "Line [1] - Add a new DN"
        - Define a Directory Number
        - Change "Route Partition" to "INTERNAL"
        - Add a Description, Alerting Name, ASCII Alerting Name
        - Add a Display (Caller ID), ASCII Display (Caller ID), External Phone Number Mask
        - Change "Busy Trigger" to "1"
        - Click "Associate End Users"
        - Click "Find"
        - Add an end user
        - [Save]
    - Follow the steps above to create a second phone

Registration Rejected

After configuring the CUCM using the steps above, I was ready to launch the IP phones. Nothing happened. The IP phones got addresses but failed beyond that step. In my previous lab with the CUCME, I set up a DHCP pool that points the IP phones to a TFTP server (in this case

the CUCM) for their bootstrap files. I forgot to configure that option, *option 150*, on my new DHCP server (the router). I quickly added in this line: *option 150 ip 10.0.0.254*, pointing the IP phones to the CUCM. Now that the IP phones were directed to the CUCM, they should be able to access their files. But they still could not, which became a problem that would haunt me for the next couple of weeks: trying to solve the error "*Registration Rejected*". How very descriptive.

My first thought was to add a DNS pointer in the DHCP pool so the IP phones could use DNS if they needed it. I do not think they did, so I began researching other solutions. The Cisco IP phone I used, the 7940 model, has a small display that provides information on the booting phase. I had not noticed before, but another message would blink briefly after the *Registration Rejected* error: "*Version Error*". This prompted me to two conclusions: the CUCM version was outdated, or the IP phones were outdated. I really did not want to reinstall the CUCM, so I opted to try and fix the phones. I reset both to factory default in hope that they would download their new model files from the CUCM, though it had no impact.

At this point a nagging dread was creeping up in me – did I compile the steps I wrong? There was always a thought in the back of my mind to start over and use the very minimal configs to get IP phones communicating. That, however, would undermine the good practices I had implemented, such as the various groups. I did not want to restart without these configurations, but I was coming up with less and less other solutions.

I tried enabling *auto-registration*, so that the IP phones might still register even if I configured their profiles incorrectly. I tinkered with TFTP service parameters in the CUCM, phone profiles, even different physical IP phone models. All methods ended with the same "*Registration Rejected*" message. I tried changing the default *auto-registration* profile to SCCP, the simpler protocol for communicating with the CUCM. Eventually I decided to take a break and work on a different project for a distraction, then come back to the CUCM after a little time away.

A weekend later, I accepted that I needed to start from the beginning. I reverted to the initial state of the CUCM before any configurations and started again. Though I skipped a lot of the good practices, I managed to get the IP phones communicating with basic configurations, using a different resource than the first time. Having got the IP phones connecting, I went back to root out why they failed the first time. I believe the issue was in using SIP: the working configurations used SCCP. I tried changing the protocol in my troubleshooting, but not every setting relating to it. Perhaps if I enabled more settings related to SCCP the phones would have received their files.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

// IPv4 DHCP Configuration

Router(config)# **ip dhcp exclude-address [*initial ip*]** *<final ip>*

- Set an IP or range of IPs to exclude from the pool

If the network administrator so chooses to exclude a range of IP addresses, the range would be from the Initial IP *to the* End IP*, inclusive. The* End IP *argument is not necessary when excluding only one IP. Excludes are typically reserved for pre-configured static IP addresses, for example, interfaces on the router.*

Router(config)# **ip dhcp pool [*pool name*]**

- Creates a pool for distributing routing information

Dynamic Host Configuration Protocol (DHCP) automates the assignment of IP addresses to devices on the local network. A DHCP pool is used to define the range of IP addresses that the server will divvy out to clients.

Router(dhcp-config)# **network [*network address*] [*subnet mask*]**

- Configures a pool that distributes the specified subnet

Router(dhcp-config)# **default-router [*ip*]**

- Sends the specified default gateway to clients

Router(dhcp-config)# **option 150 ip [*call server address*]**

- Advertises the address of the TFTP server where IP phones will request files

Cisco IP phones download their files from a TFTP server. Once an IP phone receives DHCP information, gaining an IP, it will then attempt to download the files.

// Dynamic NAT Configuration

Router(config-if)# **ip nat [*inside/outside*]**

- Configure an interface to be internal or external

Inside interfaces are translated through the outside interface.

Router(config)# **access-list [*#*] permit [*network address*] [*wildcard mask*]**

- Create an access list that permits a subnet

The subnet specified here should be on an inside interface. NAT will translate the subnet out the outside interface. Keep note of the [#] defined, for that will be used in a later command.

Router(config)# **ip nat inside source list [*#*] interface [*id*] overload**

- Enable the translation of an access list through an outside interface

The source list [#] should be an access list created earlier; the interface id should be the outside interface.

Configurations

Router title

```
Router# show running-config
service timestamps debug datetime msec
service timestamps log datetime msec
platform qfp utilization monitor load 80
platform punt-keepalive disable-kernel-core
hostname Router
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
 address-family ipv6
 exit-address-family
no aaa new-model
ip dhcp excluded-address 10.0.0.1
ip dhcp excluded-address 10.0.0.254
ip dhcp pool PHONES
 network 10.0.0.0 255.255.255.0
```

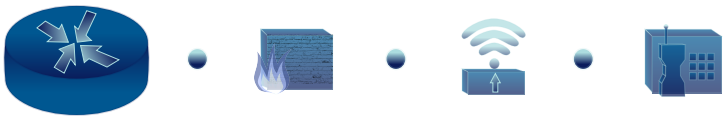
```
 default-router 10.0.0.1
 option 150 ip 10.0.0.254
 dns-server 8.8.8.8
login on-success log
subscriber templating
multilink bundle-name authenticated
crypto pki trustpoint TP-self-signed-187689846
 enrollment selfsigned
 subject-name cn=IOS-Self-Signed-Certificate-187689846
 revocation-check none
 rsakeypair TP-self-signed-187689846
license udi pid ISR4321/K9 sn FLM2407014H
no license smart enable
diagnostic bootup level minimal
spanning-tree extend system-id
redundancy
 mode none
interface GigabitEthernet0/0/0
 ip address 10.0.0.1 255.255.255.0
 no shut
 ip nat inside
 negotiation auto
interface GigabitEthernet0/0/1
 ip address dhcp
 no shut
 ip nat outside
 negotiation auto
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 shutdown
 negotiation auto
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
ip tftp source-interface GigabitEthernet0
ip nat inside source list 1 interface GigabitEthernet0/0/1 overload
access-list 1 permit 10.0.0.0 0.0.0.255
control-plane
line con 0
 transport input none
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
end
```

Conclusion

I learnt a lot about working with ESXi and the CUCM, along with the major concepts of VoIP. In the future I would like to revisit this lab and try to implement more features, such as ringtones, video, and paging. Though I ran into a lot of problems, I'm glad I persevered to reach a working solution. As a wise entity once said, *it's about the journey, not the destination*.

Virtual Routing and Forwarding

Isolating Traffic Across a Physical Network



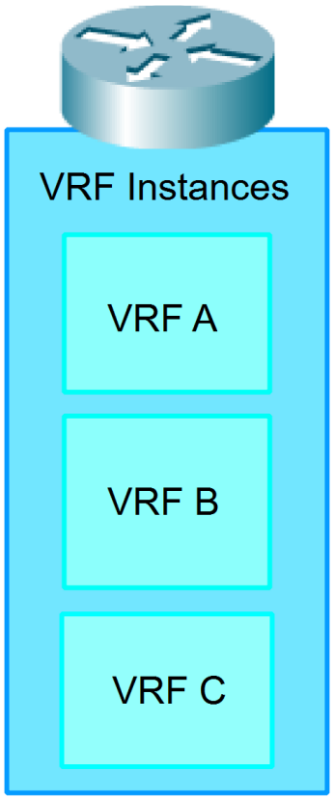
Purpose

Do you hate sharing? Want your packets to travel privately, isolated from other traffic? *Virtual Routing and Forwarding* (VRF) is perfect for you! In this lab, I configure VRF across four routers, splitting these routers into two VRFs: one that routes using EIGRP and one that routes using BGP. Packets routed on the EIGRP network are separated from those routed on the BGP network.

Background Information

What is VRF?

Have you ever thought to run Linux on Windows? It is possible through *virtualization*, which enables one host to run multiple "guest" operating systems. *Virtual Routing and Forwarding* provides the similar features, but for routers. Suppose I want a router to route three different types of traffic; I could create three VRF instances, *VRF A*, *VRF B*, and *VRF C*:



One VRF may route the traffic of a department, another for public use, and the last for network admins. We don't want these groups interacting, so we can split the router into multiple instances to route each group separately. Unlike VLANs, a hacker cannot "hop" and gain access to a different VRF, making them more secure and protected. Each VRF instance is dedicated to

routing, and as such, a unique routing table is created for each VRF. But what is a routing table?

Routing tables

Like a signpost at the edge of a crossroads, routers contain a list of directions for different destinations. A packet arrives at a router. You can think of a router like an *intersection* or *crossroads*, interfaces representing a different path to take. This router has three interfaces: north, south, and east. The packet arrived on the east interface, so it either must turn north or south, assuming one of these paths lead to the destination. Luckily, there are directions in the router: *10.0.0.0/24* out interface *north*; *172.16.0.0/24* out interface *south*. The packet has a destination address of *10.0.0.3*, which matches up with the *north* interface. The router sends the packet out the north interface. Routes are either generated statically, by the admin, or automatically by routing protocols such as OSPF, BGP, etc.

Here is an example of a routing table:

```
Gateway of last resort is not set
     10.0.0.0/8 is variably subnetted, 11 subnets, 2 masks
O IA    10.10.10.0/30 [110/128] via 10.10.10.5, 01:03:27, Serial0/1/1
C       10.10.10.4/30 is directly connected, Serial0/1/1
L       10.10.10.6/32 is directly connected, Serial0/1/1
C       10.10.10.8/30 is directly connected, Serial0/1/0
L       10.10.10.9/32 is directly connected, Serial0/1/0
O IA    10.10.10.12/30 [110/128] via 10.10.10.10, 01:03:27, Serial0/1/0
C       10.10.10.16/30 is directly connected, Serial0/2/0
L       10.10.10.17/32 is directly connected, Serial0/2/0
O IA    10.10.10.20/30 [110/128] via 10.10.10.18, 01:03:27, Serial0/2/0
O IA    10.10.10.24/30 [110/192] via 10.10.10.18, 01:03:27, Serial0/2/0
O E2    10.10.10.28/30 [110/100] via 10.10.10.18, 01:03:27, Serial0/2/0
```

Ignoring the letters on the left (the origin of the route), we can see a range of accessible addresses and the corresponding interface leading towards them. For example, *10.10.10.0/30* addresses direct out the *Serial0/1/1* interface. *Via *ip**, is also commonly seen as a direction, indicating that a packet should be sent to the specified neighboring router. Sometimes there is a combination of directions: both *interface* and *neighboring ips*.

VRF Route Distinguishers

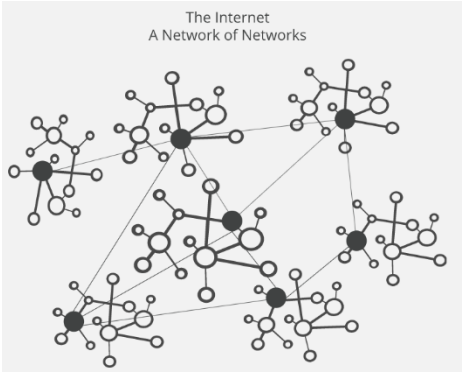
Route distinguishers (RD) allow routers to keep track of what routes belong to what VRF. Like the name implies, they *distinguish* one set of routes from another. There are different formats of RD:

1. Type 0: 2-byte ASN; 4-byte value
2. Type 1: 4-byte IP; 2-byte value
3. Type 2: 4-byte ASN; 2-byte value

For example, *100:4200000000*, *10.0.0.1:200*, *30000000:300*, are all valid RDs in the various formats, all solely cosmetic. VRF routes are identified in the form *RD:IP*, for example, *65000:1:10.0.0.0/24*.

What is BGP?

BGP (Border Gateway Protocol) is the most popular routing protocol, commonly used by ISPs (Internet Service Providers) to route customer traffic. Without BGP, the internet would not function nearly as well, if at all. Think of BGP as the postal service that delivers a letter to the recipient in the fastest and most efficient manner possible. When someone submits data across the internet, BGP is responsible for choosing the best path out of all preexisting available paths, which usually means passing through autonomous systems.



An example of Autonomous systems and their local networks

So, what are autonomous systems? Autonomous systems (AS) are a collection of routers, each with their own lesser hierarchy of routers that eventually connects to local networks. Each autonomous system is aware of other autonomous system(s) and can broadly determine where to route traffic based on which autonomous system holds the desired destination. ASes typically belong to ISPs (Internet service providers) or other large high-tech organizations, such as tech companies, universities, or scientific institutions. The internet is run under a collection of autonomous systems.

Kingdom Analogy

I suppose one could think of an autonomous system as a form of kingdom. Each kingdom has a ruler that dictates certain policies that the underlying citizens and infrastructure abide to. For example, if a kingdom is landlocked, it likely has a high demand for fish and salt. Therefore, a *policy* is implemented where all traders from the nearest port town have free access to and from the kingdom. Different autonomous systems often have these unique routing *policies*.

There are many paths and roads in the kingdom internally, so much so that if one goes down, alternate routes are readily available. Some kingdoms have routes bridging them, but often a traveler (packet) will have to journey through multiple kingdoms to reach their desired destination. In other words, a packet may have to pass through multiple ASes to reach its destination.

Each AS is assigned a unique, 32-bit number, the *Autonomous System Number* (ASN). These numbers differentiate what “kingdom” a router falls under. Routers with the same AS are part of the same kingdom. To qualify for an ASN, one needs proof of a unique routing policy,

knowledge on how to link autonomous systems, and a plentiful quantity of hosts. There is no point in creating a kingdom with only a handful of hosts. If you satisfy these rules, then the closest *regional internet registry* (RIR), may delegate you an ASN.

Internal and External BGP

As I vaguely covered in my analogy, there are two types of BGP: *internal BGP* (within kingdoms) and *external BGP* (between kingdoms).

External BGP (eBGP) is the bridge that connects autonomous systems, where neighbors can broadly exchange network prefixes to learn more about each other's networks.

Internal BGP (iBGP) is a TCP based protocol to help advertise and support eBGP routes. The kicker: iBGP alone does not do any routing. To route, one needs an IP based protocol. So why bother with iBGP at all?

Consider an old, flimsy wooden bridge. Driving a cargo truck across would collapse the bridge. But now, with iBGP, that bridge is reinforced with a concrete foundation, metal bearings, and arches to brace the heavy loads. BGP is the only protocol designed to support the hundreds of thousands of routes that make up the internet. As of writing this, the size of the full IPv4 BGP routing table is around 800,000 prefixes without even accounting for IPv6. For reference, the average OSPF router would suffer at around 6000 prefixes. This is why iBGP is oftenly used in conjunction with an IGP; the IGP does the local routing whilst iBGP contains the major routing table.

Both internal and external BGP sessions establish neighbors based on a peering system. You define a peer with a neighbor statement: for example, *neighbor 10.0.0.1 remote-as 100* states that there is a router connected, *10.0.0.1* running under ASN *100*. The neighbor *10.0.0.1* would need to define this router as a neighbor for a complete peer adjacency to form. Once both routers point to each other, they are peered. Networks are advertised with network statements: for example, *network 10.0.0.0 mask 255.255.255.0* will add the prefix *10.0.0.0/24* to the routing table. Other routers will direct traffic for *10.0.0.0/24* towards the router with the network statement.

How does BGP function?

The main purpose BGP serves is forwarding traffic to an external network in the most efficient manner possible. Some factors that determine the best path are:

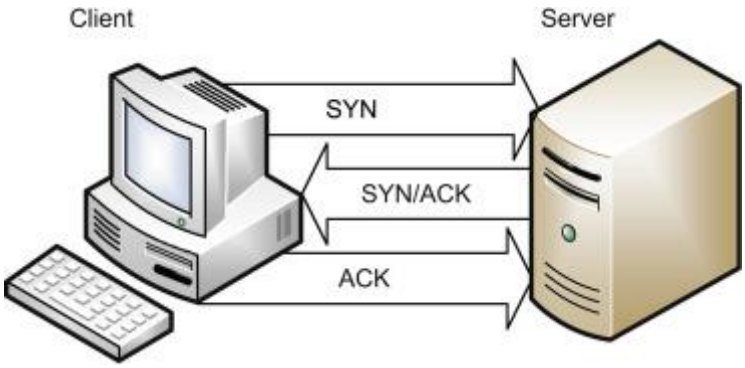
1. The path with the highest *weight*. This is a user defined variable.
2. The path with the highest *LOCAL_PREF*. Local preference determines which path is preferred when leaving a local AS.
3. The path with the highest *AS_PATH*. The main purpose of *AS_PATH* is to prevent infinite routing table updates. It is rather complicated, but essentially if a router goes down in a network, then this might cause the other routers to falsely change their paths,

resulting in an infinite loop of changing paths. This can only happen in a distance-vector routing protocol such as BGP or RIP.

4. Favoring *eBGP* paths over iBGP paths.

BGP is a *distance-vector* routing protocol. Distance-vector routing protocols work by advertising routing tables to their neighbors. If the routes from the neighbor are better than the ones they currently have, the router will update its routing table to the preferable routes. Like all other routing protocols, BGP must first establish a neighbor adjacency with another BGP router to be able to exchange routing information. Unlike other routing protocols, BGP does not broadcast or multicast to discover other BGP neighbors. Neighbor relationships must be established manually and BGP uses TCP port 179 for the connection. There are a couple of different states BGP routers may encounter when becoming neighbors:

1. Idle. In Idle, BGP waits for a "start event". This could be when a new BGP neighbor is configured or when a reset occurs between peers that already had a connection. After the start event, BGP will initialize a TCP connection with the remote neighbor and initialize some functions. In success, BGP moves to the *Connect* state, while in failure, BGP remains in the *Idle* state.
2. Connect. In *Connect*, BGP waits for the TCP three-way handshake to complete. Both sides need to *synchronize* (SYN) and *acknowledge* (ACK) each other in a TCP three-way handshake. If the results are successful, BGP advances to the *OpenSent* state. If the results are unsuccessful, BGP begins the *Active* state.

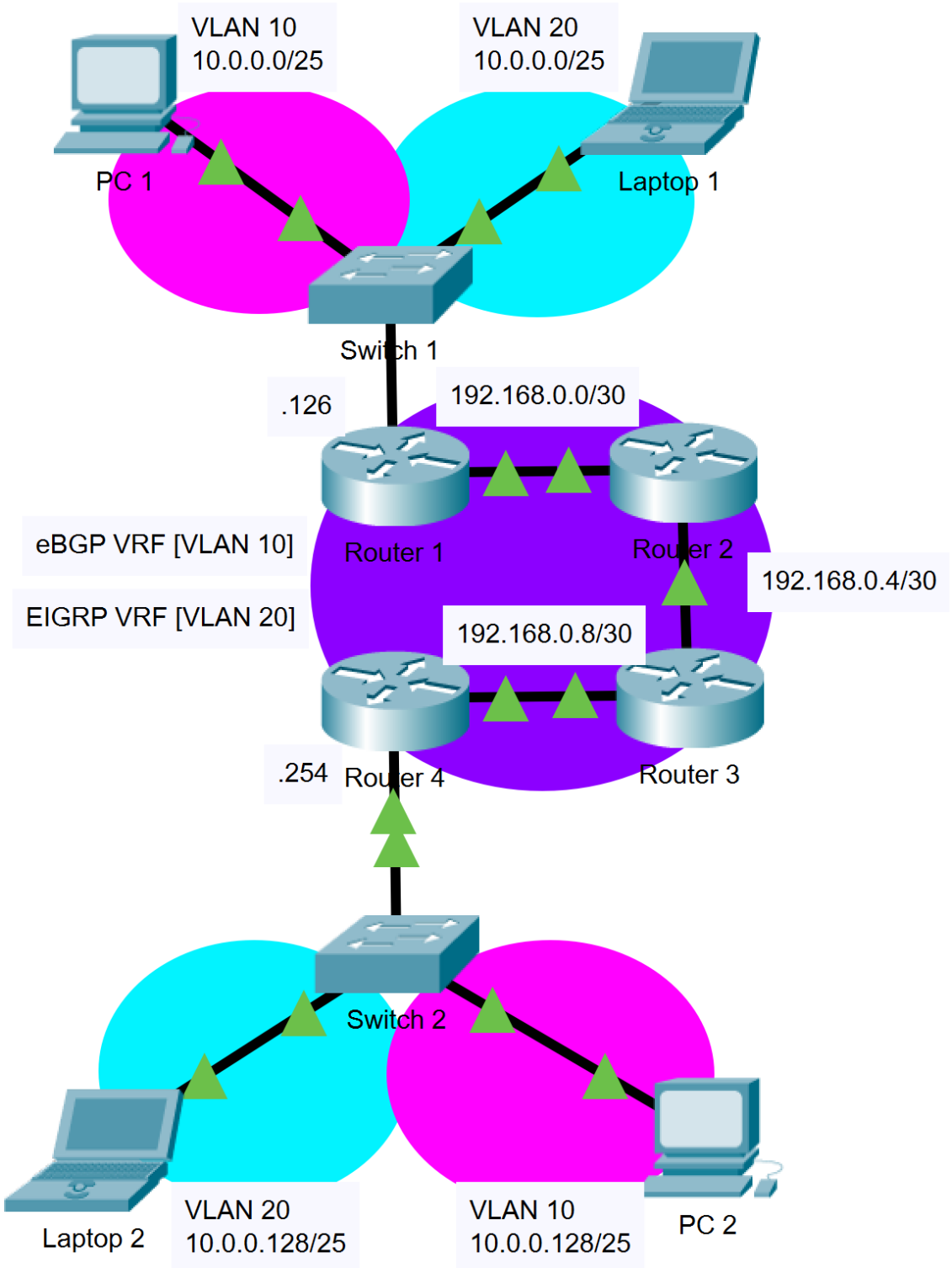


3. Active. BGP will try another TCP three-way handshake to establish a connection with the remote neighbor. On success, BGP will transition to the *OpenSent* state. After a certain amount of time has passed with no success, BGP will revert to the *Connect* state.
4. OpenSent. BGP will wait for an "open message" from the remote neighbor. Open messages contain information about the BGP router, such as version, ASN, BGP router ID, and hold time. If the versions or hold times mismatch, BGP reverts to the *Idle* state. The ASN determines whether the BGP session will be running iBGP or eBGP. If the TCP session ever fails, BGP will revert to the *Active* state. If all passes, BGP will start sending keepalive messages to maintain the TCP session.

5. OpenConfirm. BGP waits for a keepalive message from the remote BGP neighbor. When keepalive messages are consistently received, BGP moves to the *Established* state. In any other case, BGP falls back to the *Idle* state.
6. Established. The neighbor adjacency has been formed. As long as keepalive messages are sent, the neighborship remains up. Otherwise, BGP resets back to Idle state.

Adjacencies are often formed by defining the *directly connected* interface as a neighbor, a common trait in most routing protocols. However, a technique when working with BGP is to use loopback interfaces as neighbors. Using loopbacks is common for iBGP but it also works with eBGP. Loopbacks are preferred because of redundancy: if the physical interface goes down, perhaps due to hardware, loopback interfaces will stay up since they are *virtual*.

Network Diagram



Process

I began my lab with some brief research on the concepts of VRF. All example configurations I found were very similar to regular routing, with the occasional "*vrf*" statement embedded in the typical command. VRF was beginning to seem trivial – the real complications would be the routing protocols themselves. But before I could implement routing, I needed a clear topology with proper addressing.

The goal was to have two VRF networks, incorporating four routers, each VRF routed through any protocol we chose. I decided to route using *BGP* and *EIGRP*. In my topology, I created two VLANs, 10 and 20, which my *BGP* and *EIGRP* VRFs would route, respectively. Addressing came next: for point-to-point networks (router to router), I like using [/30] subnets, allocating me two addresses, perfect for a network with two devices; for the end users in the VLANs, I chose [/25] subnets so each VLAN pair would end up as a clean *10.0.0.0/24* network.

With the topology complete, I could finally configure the routers. On each router, I set up interfaces with IPs according to the topology and created two VRFs, creatively named *BGP* and *EIGRP*. You can imagine which VRF was assigned what routing protocol. Once the green lights started blinking on the router, indicating the interfaces were up, I was ready to get my first VRF working: *BGP*.

Configuring a BGP VRF was much like "normal" BGP, which I have done in the past. The main difference was the new concept of *route distinguishers*, with the introduction of VRF. BGP requires a route distinguisher when creating a routing instance, something I learnt firsthand after attempting to enter the *address-family*. "*VRF BGP does not have an RD configured*" would pop up each time. I fixed this problem by adding "*rd 10:1*" in the BGP VRF.

Lastly was EIGRP. Unlike BGP, EIGRP did not require a defined *route distinguisher*. My partner and I set up the EIGRP network, but nothing was appearing in the routing table. We tried assigning *route distinguishers*, checked *IPs*, and referenced sources online. Everything appeared correct. A while later, after double checking for the fifth time, I noticed the *neighbor statement* might be wrong. Since we were using sub-interfaces, EIGRP's neighbor statements looked almost identical: for example, *GigabitEthernet0/0/0.10* and *GigabitEthernet0/0/0.20* both point out *GigabitEthernet0/0/0* but are tagged differently. Surely enough, this slight difference was the problem.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

Router(config)# **interface [*interface*] [*id*]**

- Enables configuration on a specific interface.

// VRF

Router(config)# **ip vrf [*name*]**

- Creates a vrf

VRFs are used to instance routing tables.

Router(config-vrf)# **rd [*asn*]:[*arbitrary number*]**

- Creates a route distinguisher

The route distinguisher is like an id *for a VRF. Each routing instance is grouped based on the rd prefix at the beginning of routes in the table.*

Router(config-subif)# **ip vrf forwarding [*name*]**

- Assigns a *sub-interface* to route traffic of a specified vrf

This command is an interface command, typed in interface *or* sub-interface *mode.*

// BGP

Router(config)# **router bgp [*autonomous system number*]**

- Activates a BGP router and enters router configuration mode

The autonomous system number (ASN) is a number that identifies a large collection of routers on the internet. Typically, there are networks run under an ASN by a technical administration. eBGP connects different autonomous systems while iBGP is run within each ASN.

Router(config-router)# **address-family [*protocol*]** *vrf <vrf name>*

- Enters configuration mode for a BGP address family

I like to think of address-families as workspaces for certain IP protocols. For example, IPv6 is configured in the IPv6 address family. This is where redistribution, network statements or activation commands occur. Address-families can be implemented for separate VRFs by adding "vrf VRF".

Router(config-router-af)# **neighbor [*ip address*] remote-as [*neighbor's asn*]**

- Used in forming BGP neighbor adjacencies.

This command takes an IP address of a neighbor's interface and the ASN of the neighbor. For a BGP neighborship to be established, each router must have routes to the neighbor's IP *and the* correct IP and ASN of their neighbor.

Router(config-router-af)# **network [*network address*] mask [*subnet mask*]**

- Advertises a directly connected network to the BGP routing table

BGP's network statements are not to be confused with OSPF or EIGRPs; they aren't used to form adjacencies between BGP routers. A BGP network statement is configured alongside a neighbor statement, the former advertising the network and the latter the neighbor establishment.

Configurations

Router 1

```
R1#show running-config
service timestamps debug datetime msec
service timestamps log datetime msec
no platform punt-keepalive disable-kernel-core
hostname R1
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
```

```
 address-family ipv6
 exit-address-family
no aaa new-model
ip vrf BGP
 rd 10:1
ip vrf EIGRP
subscriber templating
multilink bundle-name authenticated
license udi pid ISR4321/K9 sn FDO214421CF
spanning-tree extend system-id
redundancy
 mode none
vlan internal allocation policy ascending
interface GigabitEthernet0/0/0
 no ip address
 negotiation auto
interface GigabitEthernet0/0/0.10
 encapsulation dot1Q 10
 ip vrf forwarding BGP
 ip address 192.168.0.1 255.255.255.252
interface GigabitEthernet0/0/0.20
 encapsulation dot1Q 20
 ip vrf forwarding EIGRP
 ip address 192.168.0.1 255.255.255.252
interface GigabitEthernet0/0/1
 no ip address
 negotiation auto
interface GigabitEthernet0/0/1.10
 encapsulation dot1Q 10
 ip vrf forwarding BGP
 ip address 10.0.0.126 255.255.255.128
interface GigabitEthernet0/0/1.20
 encapsulation dot1Q 20
 ip vrf forwarding EIGRP
 ip address 10.0.0.126 255.255.255.128
interface Serial0/1/0
 no ip address
interface Serial0/1/1
 no ip address
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 negotiation auto
interface Vlan1
 no ip address
router eigrp 20
 address-family ipv4 vrf EIGRP
  network 10.0.0.0
  network 192.168.0.0
  neighbor 192.168.0.2 GigabitEthernet0/0/0.20
  autonomous-system 20
  eigrp router-id 1.1.1.1
 exit-address-family
router bgp 10
 bgp router-id 1.1.1.1
 bgp log-neighbor-changes
 address-family ipv4 vrf BGP
  network 10.0.0.0 mask 255.255.255.128
  neighbor 192.168.0.2 remote-as 20
  neighbor 192.168.0.2 activate
 exit-address-family
ip forward-protocol nd
no ip http server
```

```
no ip http secure-server
ip tftp source-interface GigabitEthernet0
control-plane
line con 0
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
end
```

R1#sh ip route vrf BGP

```
Routing Table: BGP
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
C        10.0.0.0/25 is directly connected, GigabitEthernet0/0/1.10
L        10.0.0.126/32 is directly connected, GigabitEthernet0/0/1.10
B        10.0.0.128/25 [20/0] via 192.168.0.2, 00:04:20
      192.168.0.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.0.0/30 is directly connected, GigabitEthernet0/0/0.10
L        192.168.0.1/32 is directly connected, GigabitEthernet0/0/0.10
```

R1#sh ip route vrf EIGRP

```
Routing Table: EIGRP
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
C        10.0.0.0/25 is directly connected, GigabitEthernet0/0/1.20
L        10.0.0.126/32 is directly connected, GigabitEthernet0/0/1.20
D        10.0.0.128/25
           [90/28928] via 192.168.0.2, 00:02:15, GigabitEthernet0/0/0.20
      192.168.0.0/24 is variably subnetted, 4 subnets, 2 masks
C        192.168.0.0/30 is directly connected, GigabitEthernet0/0/0.20
L        192.168.0.1/32 is directly connected, GigabitEthernet0/0/0.20
D        192.168.0.4/30
           [90/3072] via 192.168.0.2, 00:16:21, GigabitEthernet0/0/0.20
D        192.168.0.8/30
           [90/3328] via 192.168.0.2, 00:16:17, GigabitEthernet0/0/0.20
```

R1#sh vrf detail

```
VRF BGP (VRF Id = 2); default RD 10:1; default VPNID <not set>
  Old CLI format, supports IPv4 only
  Flags: 0xC
  Interfaces:
    Gi0/0/0.10              Gi0/0/1.10
Address family ipv4 unicast (Table ID = 0x2):
```

```
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast not active
Address family ipv4 multicast not active
VRF EIGRP (VRF Id = 3); default RD <not set>; default VPNID <not set>
  Old CLI format, supports IPv4 only
  Flags: 0x8
  Interfaces:
    Gi0/0/0.20              Gi0/0/1.20
Address family ipv4 unicast (Table ID = 0x3):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast not active
Address family ipv4 multicast not active
VRF Mgmt-intf (VRF Id = 1); default RD <not set>; default VPNID <not set>
  New CLI format, supports multiple address-families
  Flags: 0x1808
  Interfaces:
    Gi0
Address family ipv4 unicast (Table ID = 0x1):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast (Table ID = 0x1E000001):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv4 multicast not active
```

Router 2

```
R2#show running-config
service timestamps debug datetime msec
service timestamps log datetime msec
no platform punt-keepalive disable-kernel-core
hostname R2
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
```

```
 exit-address-family
 address-family ipv6
 exit-address-family
no aaa new-model
ip vrf BGP
 rd 10:1
ip vrf EIGRP
subscriber templating
multilink bundle-name authenticated
license udi pid ISR4321/K9 sn FDO211216BL
spanning-tree extend system-id
redundancy
 mode none
vlan internal allocation policy ascending
interface GigabitEthernet0/0/0
 no ip address
 negotiation auto
interface GigabitEthernet0/0/0.10
 encapsulation dot1Q 10
 ip vrf forwarding BGP
 ip address 192.168.0.5 255.255.255.252
interface GigabitEthernet0/0/0.20
 encapsulation dot1Q 20
 ip vrf forwarding EIGRP
 ip address 192.168.0.5 255.255.255.252
interface GigabitEthernet0/0/1
 no ip address
 negotiation auto
interface GigabitEthernet0/0/1.10
 encapsulation dot1Q 10
 ip vrf forwarding BGP
 ip address 192.168.0.2 255.255.255.252
interface GigabitEthernet0/0/1.20
 encapsulation dot1Q 20
 ip vrf forwarding EIGRP
 ip address 192.168.0.2 255.255.255.252
interface Serial0/1/0
 no ip address
interface Serial0/1/1
 no ip address
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 negotiation auto
interface Vlan1
 no ip address
router eigrp 20
 address-family ipv4 vrf EIGRP
  network 192.168.0.0 0.0.0.3
  network 192.168.0.4 0.0.0.3
  neighbor 192.168.0.6 GigabitEthernet0/0/0.20
  neighbor 192.168.0.1 GigabitEthernet0/0/1.20
  autonomous-system 20
  eigrp router-id 2.2.2.2
 exit-address-family
router bgp 20
 bgp router-id 2.2.2.2
 bgp log-neighbor-changes
 address-family ipv4 vrf BGP
  neighbor 192.168.0.1 remote-as 10
  neighbor 192.168.0.1 activate
  neighbor 192.168.0.6 remote-as 30
  neighbor 192.168.0.6 activate
```

```
 exit-address-family
ip forward-protocol nd
no ip http server
no ip http secure-server
ip tftp source-interface GigabitEthernet0
control-plane
line con 0
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
end
```

R2#sh ip route vrf BGP

```
Routing Table: BGP
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR
Gateway of last resort is not set
      10.0.0.0/25 is subnetted, 2 subnets
B        10.0.0.0 [20/0] via 192.168.0.1, 00:08:48
B        10.0.0.128 [20/0] via 192.168.0.6, 00:08:05
      192.168.0.0/24 is variably subnetted, 4 subnets, 2 masks
C        192.168.0.0/30 is directly connected, GigabitEthernet0/0/1.10
L        192.168.0.2/32 is directly connected, GigabitEthernet0/0/1.10
C        192.168.0.4/30 is directly connected, GigabitEthernet0/0/0.10
L        192.168.0.5/32 is directly connected, GigabitEthernet0/0/0.10
```

R2#sh ip route vrf EIGRP

```
Routing Table: EIGRP
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR
Gateway of last resort is not set
      10.0.0.0/25 is subnetted, 2 subnets
D        10.0.0.0
           [90/28416] via 192.168.0.1, 00:15:51, GigabitEthernet0/0/1.20
D        10.0.0.128
           [90/28672] via 192.168.0.6, 00:00:42, GigabitEthernet0/0/0.20
      192.168.0.0/24 is variably subnetted, 5 subnets, 2 masks
C        192.168.0.0/30 is directly connected, GigabitEthernet0/0/1.20
L        192.168.0.2/32 is directly connected, GigabitEthernet0/0/1.20
C        192.168.0.4/30 is directly connected, GigabitEthernet0/0/0.20
L        192.168.0.5/32 is directly connected, GigabitEthernet0/0/0.20
D        192.168.0.8/30
           [90/3072] via 192.168.0.6, 00:14:43, GigabitEthernet0/0/0.20
```

R2#sh vrf detail

```
VRF BGP (VRF Id = 2); default RD 10:1; default VPNID <not set>
  Old CLI format, supports IPv4 only
```

```
  Flags: 0xC
  Interfaces:
    Gi0/0/0.10              Gi0/0/1.10
Address family ipv4 unicast (Table ID = 0x2):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast not active
Address family ipv4 multicast not active
VRF EIGRP (VRF Id = 3); default RD <not set>; default VPNID <not set>
  Old CLI format, supports IPv4 only
  Flags: 0x8
  Interfaces:
    Gi0/0/0.20              Gi0/0/1.20
Address family ipv4 unicast (Table ID = 0x3):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast not active
Address family ipv4 multicast not active
VRF Mgmt-intf (VRF Id = 1); default RD <not set>; default VPNID <not set>
  New CLI format, supports multiple address-families
  Flags: 0x1808
  Interfaces:
    Gi0
Address family ipv4 unicast (Table ID = 0x1):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast (Table ID = 0x1E000001):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv4 multicast not active
```

Router 3

```
R3#show running-config
service timestamps debug datetime msec
service timestamps log datetime msec
no platform punt-keepalive disable-kernel-core
hostname R3
```

```
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
 address-family ipv6
 exit-address-family
no aaa new-model
ip vrf BGP
 rd 10:1
ip vrf EIGRP
subscriber templating
vtp domain cisco
vtp mode transparent
multilink bundle-name authenticated
license udi pid ISR4321/K9 sn FDO214420G7
spanning-tree extend system-id
redundancy
 mode none
vlan internal allocation policy ascending
interface GigabitEthernet0/0/0
 no ip address
 negotiation auto
interface GigabitEthernet0/0/0.10
 encapsulation dot1Q 10
 ip vrf forwarding BGP
 ip address 192.168.0.9 255.255.255.252
interface GigabitEthernet0/0/0.20
 encapsulation dot1Q 20
 ip vrf forwarding EIGRP
 ip address 192.168.0.9 255.255.255.252
interface GigabitEthernet0/0/1
 no ip address
 negotiation auto
interface GigabitEthernet0/0/1.10
 encapsulation dot1Q 10
 ip vrf forwarding BGP
 ip address 192.168.0.6 255.255.255.252
interface GigabitEthernet0/0/1.20
 encapsulation dot1Q 20
 ip vrf forwarding EIGRP
 ip address 192.168.0.6 255.255.255.252
interface Serial0/1/0
 no ip address
 shutdown
interface Serial0/1/1
 no ip address
 shutdown
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 shutdown
 negotiation auto
interface Vlan1
 no ip address
 shutdown
router eigrp 20
 address-family ipv4 vrf EIGRP
  network 192.168.0.4 0.0.0.3
  network 192.168.0.8 0.0.0.3
  neighbor 192.168.0.10 GigabitEthernet0/0/0.20
  neighbor 192.168.0.5 GigabitEthernet0/0/1.20
  autonomous-system 20
```

```
  eigrp router-id 3.3.3.3
 exit-address-family
router bgp 30
 bgp router-id 3.3.3.3
 bgp log-neighbor-changes
 address-family ipv4 vrf BGP
  neighbor 192.168.0.5 remote-as 20
  neighbor 192.168.0.5 activate
  neighbor 192.168.0.10 remote-as 40
  neighbor 192.168.0.10 activate
 exit-address-family
ip forward-protocol nd
no ip http server
no ip http secure-server
ip tftp source-interface GigabitEthernet0
control-plane
line con 0
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
end
```

R3#sh ip route vrf BGP

```
Routing Table: BGP
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR
Gateway of last resort is not set
      10.0.0.0/25 is subnetted, 2 subnets
B        10.0.0.0 [20/0] via 192.168.0.5, 00:09:32
B        10.0.0.128 [20/0] via 192.168.0.10, 00:09:32
      192.168.0.0/24 is variably subnetted, 4 subnets, 2 masks
C        192.168.0.4/30 is directly connected, GigabitEthernet0/0/1.10
L        192.168.0.6/32 is directly connected, GigabitEthernet0/0/1.10
C        192.168.0.8/30 is directly connected, GigabitEthernet0/0/0.10
L        192.168.0.9/32 is directly connected, GigabitEthernet0/0/0.10
```

R3#sh ip route vrf EIGRP

```
Routing Table: EIGRP
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR
Gateway of last resort is not set
      10.0.0.0/25 is subnetted, 2 subnets
D        10.0.0.0
           [90/28672] via 192.168.0.5, 00:17:31, GigabitEthernet0/0/1.20
D        10.0.0.128
           [90/28416] via 192.168.0.10, 00:03:29, GigabitEthernet0/0/0.20
      192.168.0.0/24 is variably subnetted, 5 subnets, 2 masks
```

```
D        192.168.0.0/30
           [90/3072] via 192.168.0.5, 00:17:31, GigabitEthernet0/0/1.20
C        192.168.0.4/30 is directly connected, GigabitEthernet0/0/1.20
L        192.168.0.6/32 is directly connected, GigabitEthernet0/0/1.20
C        192.168.0.8/30 is directly connected, GigabitEthernet0/0/0.20
L        192.168.0.9/32 is directly connected, GigabitEthernet0/0/0.20
```

R3#sh vrf detail

```
VRF BGP (VRF Id = 2); default RD 10:1; default VPNID <not set>
  Old CLI format, supports IPv4 only
  Flags: 0xC
  Interfaces:
    Gi0/0/0.10              Gi0/0/1.10
Address family ipv4 unicast (Table ID = 0x2):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast not active
Address family ipv4 multicast not active
VRF EIGRP (VRF Id = 3); default RD 20:1; default VPNID <not set>
  Old CLI format, supports IPv4 only
  Flags: 0xC
  Interfaces:
    Gi0/0/0.20              Gi0/0/1.20
Address family ipv4 unicast (Table ID = 0x3):
  Flags: 0x0
  Export VPN route-target communities
    RT:20:1
  Import VPN route-target communities
    RT:20:1
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast not active
Address family ipv4 multicast not active
VRF Mgmt-intf (VRF Id = 1); default RD <not set>; default VPNID <not set>
  New CLI format, supports multiple address-families
  Flags: 0x1808
  Interfaces:
    Gi0
Address family ipv4 unicast (Table ID = 0x1):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast (Table ID = 0x1E000001):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
```

```
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv4 multicast not active
```

Router 4

```
R4#show running-config
service timestamps debug datetime msec
service timestamps log datetime msec
no platform punt-keepalive disable-kernel-core
hostname R4
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
 address-family ipv6
 exit-address-family
no aaa new-model
ip vrf BGP
 rd 10:1
ip vrf EIGRP
subscriber templating
vtp domain cisco
vtp mode transparent
multilink bundle-name authenticated
license udi pid ISR4321/K9 sn FDO21442B21
spanning-tree extend system-id
redundancy
 mode none
vlan internal allocation policy ascending
interface GigabitEthernet0/0/0
 no ip address
 negotiation auto
interface GigabitEthernet0/0/0.10
 encapsulation dot1Q 10
 ip vrf forwarding BGP
 ip address 10.0.0.254 255.255.255.128
interface GigabitEthernet0/0/0.20
 encapsulation dot1Q 20
 ip vrf forwarding EIGRP
 ip address 10.0.0.254 255.255.255.128
interface GigabitEthernet0/0/1
 no ip address
 negotiation auto
interface GigabitEthernet0/0/1.10
 encapsulation dot1Q 10
 ip vrf forwarding BGP
 ip address 192.168.0.10 255.255.255.252
interface GigabitEthernet0/0/1.20
 encapsulation dot1Q 20
 ip vrf forwarding EIGRP
 ip address 192.168.0.10 255.255.255.252
interface Serial0/1/0
 no ip address
 shutdown
interface Serial0/1/1
 no ip address
 shutdown
interface GigabitEthernet0/2/0
 no ip address
 shutdown
```

```
 negotiation auto
interface GigabitEthernet0/2/1
 no ip address
 shutdown
 negotiation auto
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 shutdown
 negotiation auto
interface Vlan1
 no ip address
 shutdown
router eigrp 20
 address-family ipv4 vrf EIGRP
  network 192.168.0.8 0.0.0.3
  network 10.0.0.128 mask 0.0.0.127
  neighbor 192.168.0.9 GigabitEthernet0/0/1.20
  autonomous-system 20
  eigrp router-id 4.4.4.4
 exit-address-family
router bgp 40
 bgp router-id 4.4.4.4
 bgp log-neighbor-changes
 address-family ipv4 vrf BGP
  network 10.0.0.128 mask 255.255.255.128
  neighbor 192.168.0.9 remote-as 30
  neighbor 192.168.0.9 activate
 exit-address-family
ip forward-protocol nd
no ip http server
no ip http secure-server
ip tftp source-interface GigabitEthernet0
control-plane
line con 0
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
end
```

R4#sh ip route vrf BGP

```
Routing Table: BGP
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
B        10.0.0.0/25 [20/0] via 192.168.0.9, 00:17:24
C        10.0.0.128/25 is directly connected, GigabitEthernet0/0/0.10
L        10.0.0.254/32 is directly connected, GigabitEthernet0/0/0.10
      192.168.0.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.0.8/30 is directly connected, GigabitEthernet0/0/1.10
L        192.168.0.10/32 is directly connected, GigabitEthernet0/0/1.10
```

R4#sh ip route vrf EIGRP

```
Routing Table: EIGRP
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
D        10.0.0.0/25
           [90/28928] via 192.168.0.9, 00:18:22, GigabitEthernet0/0/1.20
C        10.0.0.128/25 is directly connected, GigabitEthernet0/0/0.20
L        10.0.0.254/32 is directly connected, GigabitEthernet0/0/0.20
      192.168.0.0/24 is variably subnetted, 4 subnets, 2 masks
D        192.168.0.0/30
           [90/3328] via 192.168.0.9, 00:18:22, GigabitEthernet0/0/1.20
D        192.168.0.4/30
           [90/3072] via 192.168.0.9, 00:18:26, GigabitEthernet0/0/1.20
C        192.168.0.8/30 is directly connected, GigabitEthernet0/0/1.20
L        192.168.0.10/32 is directly connected, GigabitEthernet0/0/1.20
```

R4#sh vrf detail

```
VRF BGP (VRF Id = 2); default RD 10:1; default VPNID <not set>
  Old CLI format, supports IPv4 only
  Flags: 0xC
  Interfaces:
    Gi0/0/0.10              Gi0/0/1.10
Address family ipv4 unicast (Table ID = 0x2):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast not active
Address family ipv4 multicast not active
VRF EIGRP (VRF Id = 3); default RD <not set>; default VPNID <not set>
  Old CLI format, supports IPv4 only
  Flags: 0x8
  Interfaces:
    Gi0/0/0.20              Gi0/0/1.20
Address family ipv4 unicast (Table ID = 0x3):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast not active
Address family ipv4 multicast not active
VRF Mgmt-intf (VRF Id = 1); default RD <not set>; default VPNID <not set>
  New CLI format, supports multiple address-families
  Flags: 0x1808
  Interfaces:
    Gi0
Address family ipv4 unicast (Table ID = 0x1):
```

```
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv6 unicast (Table ID = 0x1E000001):
  Flags: 0x0
  No Export VPN route-target communities
  No Import VPN route-target communities
  No import route-map
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: per-prefix
Address family ipv4 multicast not active
```

Switch

(Both Switches have the same configuration)

```
SW#show running-config
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
hostname S1
boot-start-marker
boot-end-marker
no aaa new-model
system mtu routing 1500
vtp domain CCNP
vtp mode transparent
spanning-tree mode pvst
spanning-tree extend system-id
vlan internal allocation policy ascending
vlan 2
 name forleft
vlan 3
 name forright
vlan 10
 name BGP
vlan 20
 name EIGRP
vlan 30
 name Expedia
vlan 40
 name forty
vlan 100
vlan 996
 name CUSTOMER_NATIVE
interface FastEthernet1/0/1
 switchport access vlan 10
 switchport mode access
interface FastEthernet1/0/2
 switchport access vlan 10
 switchport mode access
interface FastEthernet1/0/3
 switchport access vlan 10
 switchport mode access
```

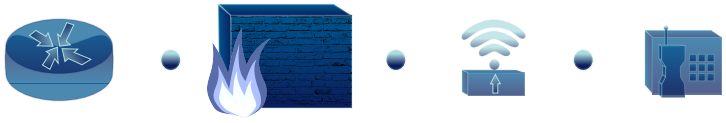
```
interface FastEthernet1/0/4
 switchport access vlan 10
 switchport mode access
interface FastEthernet1/0/5
 switchport access vlan 10
 switchport mode access
interface FastEthernet1/0/6
 switchport access vlan 20
 switchport mode access
interface FastEthernet1/0/7
 switchport access vlan 20
 switchport mode access
interface FastEthernet1/0/8
 switchport access vlan 20
 switchport mode access
interface FastEthernet1/0/9
 switchport access vlan 20
 switchport mode access
interface FastEthernet1/0/10
 switchport access vlan 20
 switchport mode access
interface FastEthernet1/0/11
 switchport access vlan 20
 switchport mode access
interface FastEthernet1/0/12
 switchport trunk encapsulation dot1q
 switchport mode trunk
interface FastEthernet1/0/13
interface FastEthernet1/0/14
interface FastEthernet1/0/15
interface FastEthernet1/0/16
interface FastEthernet1/0/17
interface FastEthernet1/0/18
interface FastEthernet1/0/19
interface FastEthernet1/0/20
interface FastEthernet1/0/21
interface FastEthernet1/0/22
interface FastEthernet1/0/23
interface FastEthernet1/0/24
interface GigabitEthernet1/0/1
interface GigabitEthernet1/0/2
interface GigabitEthernet1/1/1
 speed auto 1000
interface GigabitEthernet1/1/2
 speed auto 1000
interface Vlan1
 no ip address
 shutdown
ip http server
ip http secure-server
logging esm config
line con 0
line vty 0 4
 login
line vty 5 15
 login
end
```

Conclusion

Like VLANs, VRFs are significant in keeping portions of a network private. I learnt how to set up VRFs for BGP and EIGRP, though I'm sure I could translate the concepts to other routing protocols. At first, I was apprehensive of VRF's use cases, but I came to appreciate how convenient it could be for larger organizations who may rely on isolated traffic.

Authentication, Authorization, Accounting

Securing Login to a Cisco Router with Radius and Tacacs



Purpose

An administrator can access a router or device through a console cable, but that would require them to be within physical range. What if they monitor and control devices across the world? An admin can't be everywhere at once, so thankfully they can use remote connection to access a faraway device. Unfortunately, this presents many security risks. If an admin can remotely connect to a device, why wouldn't a hacker be able to do the same? These are the questions that AAA attempts to solve, providing security for remote devices via authentication, authorization, and accounting.

Background Information

This lab will be split up into two sections: one part on *RADIUS* and another for *TACACS+*. I set up RADIUS on a Linux server and TACACS+ on Windows Server 2019. The goal was to secure login for administrative access. *Secure Shell* (SSH) is a common remote connection protocol used to access network devices that are not local. It is a secure alternative to other remote login protocols, such as telnet or rlogin. To test login access, I will *SSH* into the router and be prompted to enter login details verified by a AAA server. Guides I made for configuring both RADIUS and TACACS can be found on my GitHub or under the *Configuration Guides* section of the physical portfolio copy.

Authentication, Authorization, Accounting

Authentication, Authorization, Accounting (AAA, pronounced "triple A") is a standard used to control who has access to network devices, what level of permission they have, and the logging of a user's activity. If AAA is not used, then authentication would be handled locally on each individual device, likely using shared usernames and passwords. Having to manage everything locally and individually is a huge strain on human resources and a security risk, prompting admins to use AAA to secure their services. Popular AAA services include RADIUS, TACACS+, and Diameter.

- *Authentication* provides a way to identify a user through a valid username and password before access is granted. When a user wants to remotely access a device, the device will compare the user's credentials with those stored on the AAA server. If the credentials match on the AAA server, the user is granted access to the device. Otherwise, the device is denied.

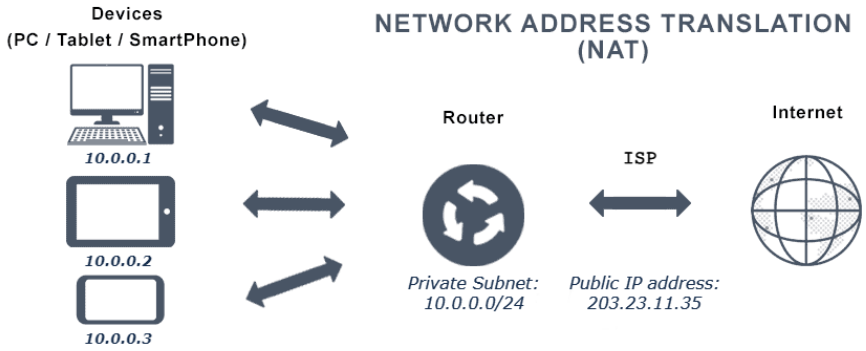
- *Authorization* is the process of determining the level of permission a user has, namely, what they are permitted to access. After a user is authenticated, they may be authorized to view or edit certain files.

- *Accounting* logs the activity of a user during access, which may include the length of time spent, what they accessed, or changes they made. These statistics can be used by higher officials to determine billing, trend analysis, time management, and such.

In this lab, I configured both RADIUS and TACACS to test different AAA services.

Network Address Translation

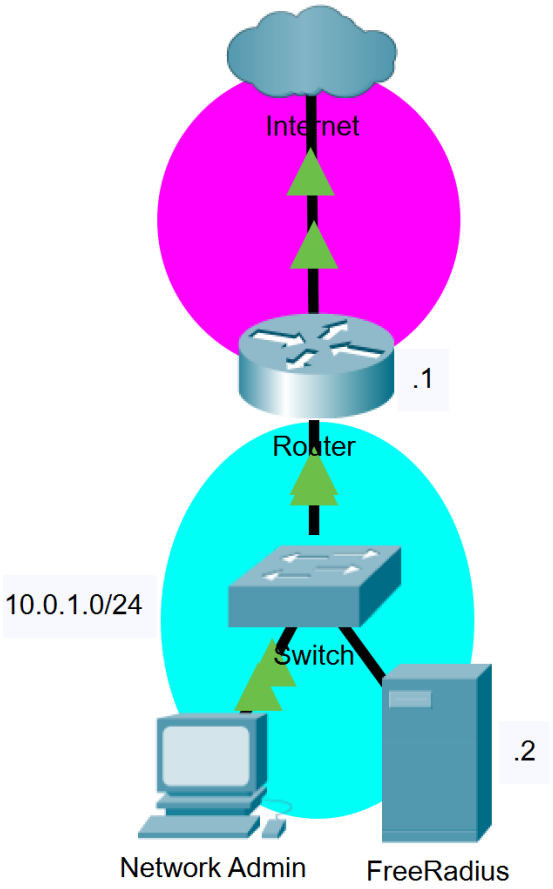
Network Address Translation (NAT) is a process that enables a unique IP address to represent an entire group of devices. Typically, this IP address is "Public", administered by one's ISP.



In this example, the public IP, *203.23.11.35*, is representing the *10.0.0.0/24* subnet. There can be multiple networks that use the same private subnet – what matters is that the public IP is unique. With so many devices nowadays, the total IPv4 address pool is running out. NAT helps conserve these addresses by translating multiple private IPs through one public IP. This is known as dynamic NAT. In this lab, I used NAT to provide internet access for my Linux server so I could download *FreeRADIUS* dependencies.

RADIUS

Topology



Background Information

Like Windows and MacOS, Linux is an operating system which can run applications, but particularly excels at server-based services, such as hosting a RADIUS server, since it is much more lightweight than other software. In computing, something "lightweight" refers to software designed to have a small memory footprint (RAM), low CPU, and low overall usage of system resources. Perfect for something like a server, but less user friendly. Users will typically navigate Linux through the command-line with less focus on graphical applications.

Since Linux is an open-source OS, where anyone can take the base code and manufacture it to their liking, there are many *distributions* designed for specific purposes. For example, in a previous project I worked with *Kali* Linux, a distribution focused on network penetration testing. In this project I used *Raspbian* – a modification of *Debian* – also a distribution of Linux. Raspbian

is the official OS of the Raspberry Pi, which is just a tiny computer. I used the Raspberry Pi as my server, though this project could be replicated simply with any distribution of Linux.

Process

RADIUS was the first AAA protocol I stumbled upon when investigating AAA services. After looking into possible RADIUS server options, *FreeRADIUS* on Linux appeared to be popular among developers and I wanted to get more familiar with Linux as an OS. It seemed like a win-win situation. In an older project I installed Kali Linux on a laptop, so my initial thought was to use that as my server. I ran the command *sudo apt-get install freeradius freeradius-utils* on the Kali laptop to install the FreeRADIUS packages. Everything went smoothly until I tried running the server, which ended up crashing every time with an incomprehensible failure error. I tried reinstalling the packages, but the error persisted.

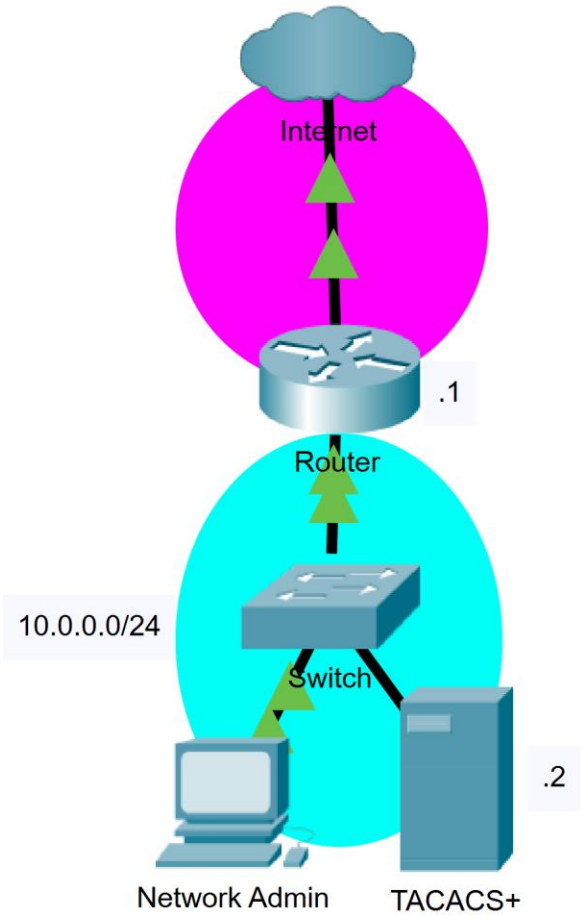
Perhaps Kali Linux wasn't the best Linux distribution to choose as my server. Why not use the spare Raspberry Pi laying around instead? Raspbian is a distribution of Linux, so most (if not all) of the commands I used on the Kali should work on the Pi. I began by installing the FreeRADIUS packages on the Pi the same way I did for Kali. When I ran the server with no configuration, there was no error like Kali, so I felt safe to continue.

I added the router as a client in my FreeRADIUS configuration by editing the *clients.conf* file, then edited the user file to create an account. On the router, I configured various *aaa authentication* commands. These commands tell the router to verify credentials with a specified AAA server, which you can define using the *radius server* command. I ran the AAA server and attempted to login to the router. The authentication failed.

To troubleshoot, my first idea was checking the debug log of the FreeRADIUS server. The debug log contained the username and password I entered on the router that the server intended to verify. The username I entered was correct, but the password was a mess of random Unicode characters that I didn't enter, like so: *\304<\306D\202\346jh\371\n\233…* I presumed the router might be encrypting the password, sending it to the AAA server, and the server not doing the necessary decryption. My solution was to rework the *aaa authentication* commands I had on the router and restart the FreeRADIUS server. In my opinion, the best feature of FreeRADIUS was the *debug* mode (*sudo freeradius -X*), which would run the server and output helpful information in real time like login attempts or login failures.

TACACS+

Topology



Background Information

Windows Server is an OS developed by Microsoft that supports enterprise-level management, data storage, applications, and communications. Many argue that Windows Server is more beginner friendly due to its intuitive graphical interface and requires less maintenance thanks to automated updates.

TACACS is a Cisco proprietary protocol. In this regard, it is better suited to handle AAA between Cisco devices, which is ideal for this project. I found that TACACS provided many more *authorization* features than RADIUS. For example, if I wanted a user whose sole purpose is to debug, I would only let them type "show" commands. TACACS turns this seemingly complicated

task into a simple process with access-list like syntax. I would recommend TACACS over RADIUS for securing Cisco devices just because of this feature.

Process

After configuring RADIUS using a Linux server, I was intrigued in doing the same for *TACACS+*, but on Windows Server. The concepts of AAA should translate the same across the services, so there was not much extra to learn. I intended to host the Windows Server on VirtualBox, a virtualization program.

I ran through the process of installing a Windows Server 2019 VM without any issues. We decided to use the TACACS service from *TACACS.net*, which was kindly provided to me by Harsha Bhat on a USB drive. The program is an executable that can be run as a Windows service. The main problem was transferring the file from the USB to my virtual machine. It sounds like a trivial issue, *the VM is running on your PC, so just copy the file*, but frustratingly was not so easy. There are two options in VirtualBox to transfer files: through USB or through "file sharing". I got an error when I tried to access my USB and file sharing was not sharing my files. Eventually, I found out that VirtualBox does not support USB 3.0 by default. You need to install an extension pack for it to work.

With the extension pack installed, I transferred the TACACS file from the USB to my virtual machine. I ran the program and went through the initial TACACS install. Much like RADIUS on Linux, I had to edit files to configure TACACS on Windows Server. Once I had the client (the router) and a user account set up, I checked my configuration with TACAC's testing commands. *Tacverify* can be used to check the files for any syntax errors and *tactest* can be used for troubleshooting authentication requests. For example, by running the command *tactest -s 127.0.0.1 -k key -u giga -p chad*, I'm asking the server to verify an account with a username of "giga" and password of "chad". The "key" is a credential that the AAA server and router share since the router is a client of the AAA server. The AAA server must first verify the router with the key. I had a minor lapse when *tactest* failed for the IP *10.0.0.2* (the AAA server), though restarting the TACACS service with *sc start/stop TACACS.net* solved that problem.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

Router(config)# **aaa new-model**

- Specifies AAA as the authentication method for VTY lines on the router

To configure any AAA services, you must first define aaa new-model.

// RADIUS Authentication

Router(config)# **aaa authentication attempts login [#]**

- Specifics the number of login attempts a user gets before the connection terminates

Router(config)# **aaa authentication banner** `*message*`

- Set a message for a user when they connect to the device

Router(config)# **aaa authentication fail-message** `*message*`

- Set a message if the user fails their credentials

Router(config)# **aaa authentication *login default group radius***

- Make the router verify login credentials with a radius server

Router(config)# **aaa authentication *enable default group radius***

- Make the router verify privilege exec mode credentials with a radius server

// Defining a RADIUS Server

Router(config)# **radius server [*name*]**

- Define a radius server

The router will use the ip of the radius server subsequently provided to verify credentials. This command can only be typed after aaa new-model *has been declared.*

Router(config-radius-server)# **address ipv4 [*ip*] auth-port *1812* acct-port *1813***

- Define the *ip* of the radius server

Router(config-radius-server)# **key [*key*]**

- Define the *key* of the radius server

The key on the router should match the key in the radius server's configuration files.

// TACACS+ Authentication

Router(config)# **aaa authentication *login default group tacacs+***

- Make the router verify login credentials with a tacacs server

Router(config)# **aaa authentication *enable default group tacacs+***

- Make the router verify privilege exec mode credentials with a tacacs server

// Defining a TACACS Server

Router(config)# **tacacs server [*name*]**

- Define a tacacs server

The router will use the ip of the tacacs server subsequently provided to verify credentials. This command can only be typed after aaa new-model *has been declared.*

Router(config-server-tacacs)# **address ipv4 [*ip*]**

- Define the *ip* of the tacacs server

Router(config-server-tacacs)# **key [*key*]**

- Define the *key* of the tacacs server

The key on the router should match the key in the tacacs server's configuration files.

// Dynamic NAT Configuration

Router(config-if)# **ip nat [*inside/outside*]**

- Configure an interface to be internal or external

Inside interfaces are translated through the outside interface.

Router(config)# **access-list [*#*] permit [*network address*] [*wildcard mask*]**

- Create an access list that permits a subnet

The subnet specified here should be on an *inside* interface. NAT will translate the subnet out the *outside* interface. Keep note of the [*#*] defined, for that will be used in a later command.

Router(config)# **ip nat inside source list [*#*] interface [*id*] overload**

- Enable the translation of an access list through an outside interface

The source list [#] should be an access list created earlier; the interface id should be the outside interface.

Configurations

Router (RADIUS)

```
#show running-config
service timestamps debug datetime msec
service timestamps log datetime msec
platform qfp utilization monitor load 80
platform punt-keepalive disable-kernel-core
hostname Router
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
 address-family ipv6
 exit-address-family
aaa new-model
aaa authentication attempts login 5
aaa authentication banner `Stop mortal. Speak thine Words of Entry.`
aaa authentication fail-message `You have failed. Begone.`
aaa authentication login default group radius
aaa authentication enable default group radius
aaa session-id common
login on-success log
subscriber templating
multilink bundle-name authenticated
no license smart enable
diagnostic bootup level minimal
spanning-tree extend system-id
redundancy
 mode none
interface GigabitEthernet0/0/0
 ip address dhcp
 ip nat outside
 negotiation auto
 no shutdown
interface GigabitEthernet0/0/1
 ip address 10.0.1.1 255.255.255.0
```

```
 ip nat inside
 negotiation auto
 no shutdown
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 shutdown
 negotiation auto
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
ip http client source-interface GigabitEthernet0/0/0
ip nat inside source list 1 interface GigabitEthernet0/0/0 overload
access-list 1 permit 10.0.1.0 0.0.0.255
radius server PI_RADIUS
 address ipv4 10.0.1.2 auth-port 1812 acct-port 1813
 timeout 30
 retransmit 3
 key chad
control-plane
line con 0
 exec-timeout 0 0
 transport input none
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
end
```

Router (TACACS)

```
Router#show running-config
service timestamps debug datetime msec
service timestamps log datetime msec
platform qfp utilization monitor load 80
platform punt-keepalive disable-kernel-core
hostname Router
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
 address-family ipv6
 exit-address-family
aaa new-model
aaa authentication login default group tacacs+
aaa authentication enable default group tacacs+
aaa session-id common
login on-success log
subscriber templating
multilink bundle-name authenticated
no license smart enable
diagnostic bootup level minimal
spanning-tree extend system-id
redundancy
 mode none
interface GigabitEthernet0/0/0
 ip address dhcp
 ip nat outside
 negotiation auto
 no shutdown
```

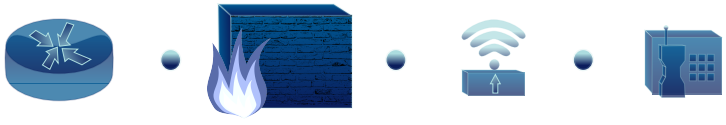
```
interface GigabitEthernet0/0/1
 ip address 10.0.0.1 255.255.255.0
 ip nat inside
 negotiation auto
 no shutdown
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 shutdown
 negotiation auto
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
ip http client source-interface GigabitEthernet0/0/0
ip nat inside source list 1 interface GigabitEthernet0/0/0 overload
access-list 1 permit 10.0.0.0 0.0.0.255
tacacs server WINSER2019
 address ipv4 10.0.0.2
 key chad
control-plane
line con 0
 exec-timeout 0 0
 transport input none
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
end
```

Conclusion

In this lab, I secured a Cisco router using *RADIUS* then *TACACS+*, implementing authentication and authorization. Considering how widely used AAA is in industry, it is important to learn for network security. Links to AAA guides can be found on my GitHub, as can many of my other projects.

Policy Based Routing

Routing Based on Packet Protocol Types



Purpose

Imagine a scenario where you decide to host a server on the internet. Perhaps a website, or maybe a file server. Potentially something else entirely. Without adding any traffic *rules* to your router, anyone from anywhere using any protocol would have access to the server with no restraints. *Policy-based Routing* (PBR) is a routing technique that forwards or denies traffic based on an implementation of rules and filters, known as *policies*. Suddenly, those unused ports on the network can be blocked, and only the necessary ones left open.

Background Information

Introduction

Two servers would run the same two protocols on the same network. However, only one type of traffic was to be permitted to each server from an external network. The goal of this project was to use *policy-based routing* to permit a certain protocol to each server. In this case, we chose to permit HTTP to one server and SSH to the other, with both servers running HTTP and SSH.

Access Lists

Routers and firewalls can use *Access-lists* (ACL) to permit or deny incoming or outgoing traffic. They function as a set of rules, read from the first to last rule. If a packet matches a rule before it is denied, then it "passes" the ACL and continues its path. If a packet does not match any rule in the ACL, then it is denied implicitly, a feature known as the *implicit deny*. There are two main types of ACL: standard and extended.

- Standard
  - Can only match source IP.
  - Cannot match port.
  - Doesn't distinguish between TCP / UDP / other traffic.
  - Standard ACL identifiers: [*1-99*] & [*1300-1999*] or *named*.
  - *"ip access-list standard 10"*
    "permit 10.0.0.0 255.255.255.0".
- Extended
  - Can match traffic based on source IP, destination IP, source port, destination port.
  - Can distinguish between TCP / UDP / other traffic.
  - Extended ACL identifiers: [*100-199*] & [*2000-2699*] or *named*.
  - *"ip access-list extended PERMIT_HTTP"*
    "permit tcp any host 10.0.0.1 eq www".

For example, let's suppose I want to discard all ping (*icmp*) packets entering my network. A flood of ping requests can overwhelm a network device, denying users service. I could configure the following on a Cisco router:

ip access-list extended DENY_ICMP
deny icmp any any
permit ip any any

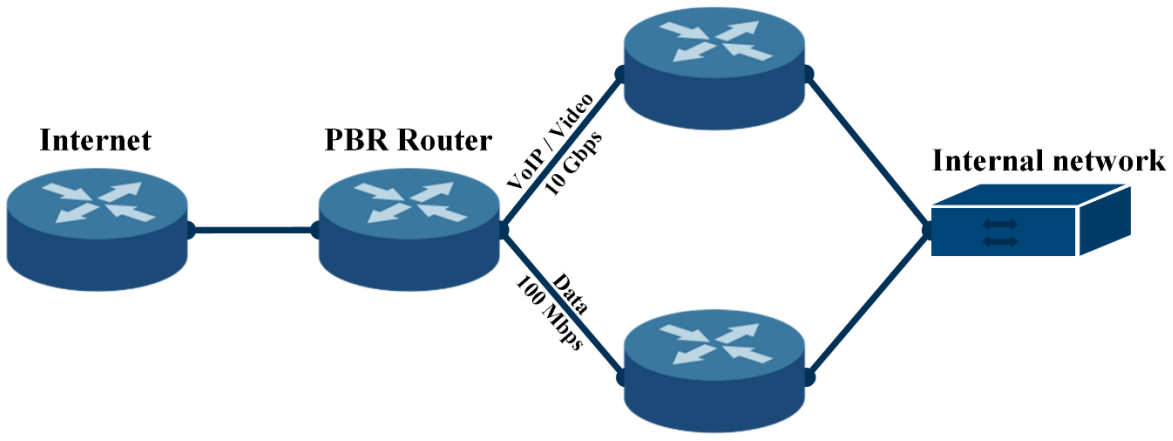
If applied on the correct interface, this access list will deny *icmp* packets entering the network but permit all other types of traffic. But what if an administrator from outside wants to ping our internal network? We would edit the ACL to permit the admin *before* we deny all icmp traffic. Since access lists are read from top to bottom, a permit line before the deny would save the admin's traffic from being dropped. The admin's ip address is *232.45.6.23*.

ip access-list extended DENY_ICMP
permit icmp host 232.45.6.23 any
deny icmp any any
permit ip any any

Once the desired access list is created and written, we must apply it on an interface and specify whether we want the traffic coming in or going out to be examined. If I wanted to apply the icmp ACL on all packets *entering* the *GigabitEthernet0/0/0* interface of my router, then the command would be: *ip access-group DENY_ICMP in* (in interface configuration mode).

Policy Based Routing

Now we know a little about access lists; so, how do they differ from *policy-based routing*? Policy-based routing uses *route maps* – containing access lists – to manipulate traffic like generic access lists. However, route maps provide the additional feature of editing and adding parts to traffic after it has passed or failed the access lists. Let's look at an example.



In this case, we have a router (*PBR Router*) that forwards traffic from the internet towards an internal network. The PBR Router has two choices of which to send the traffic: through a *10-*

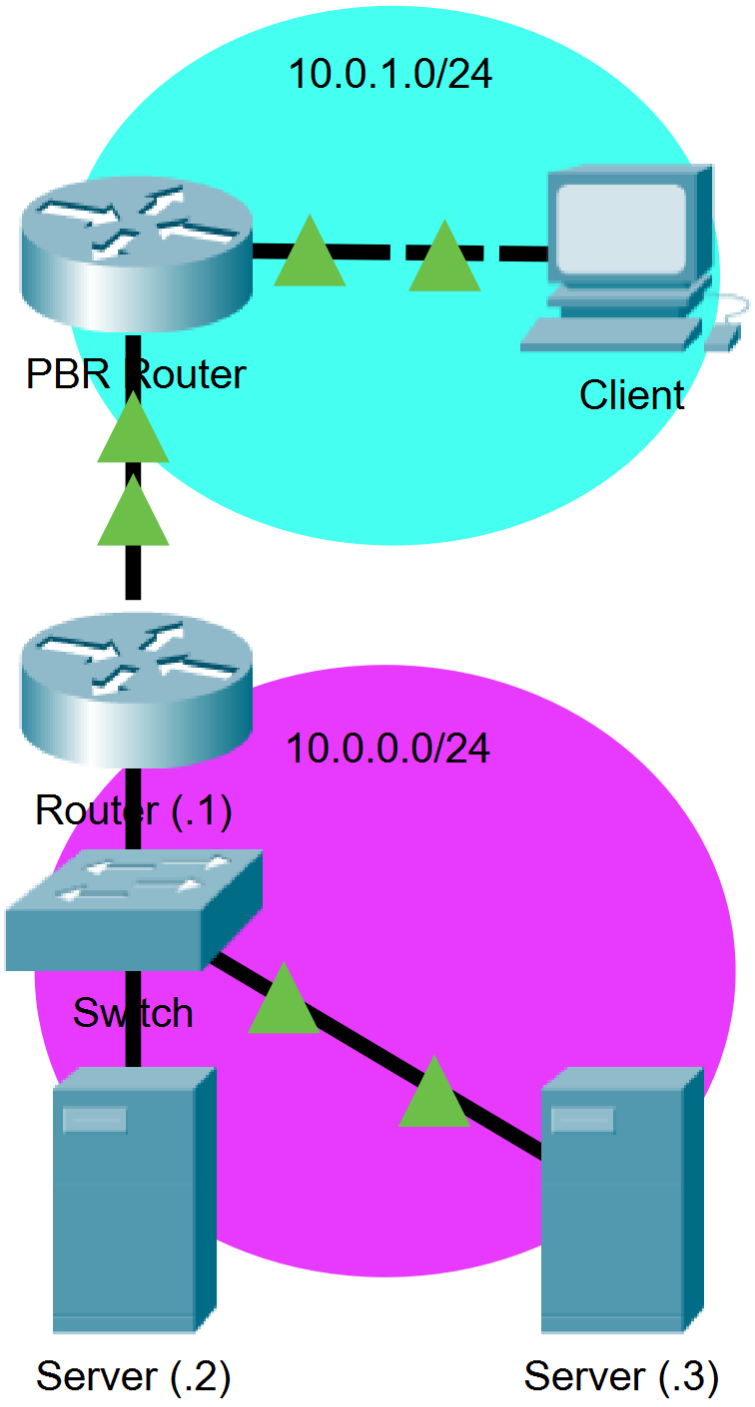
gigabit ethernet connection or through a *100-megabit* ethernet connection. Both routes eventually lead to the internal network. The network administrator wants to divide traffic so that VoIP and Video traffic is sent down the 10 *Gbps* link while other traffic is sent down the 100 *Mbps* link. With *policy-based routing*, we can *edit* parts of a packet, such as the *destination* or *next-hop ip*, and forward it along our desired path based on some attribute of the packet. This is a step up from filtering traffic via access list; now we can take filtered traffic and edit parts to exercise more control over our network.

Linux

Like Windows and MacOS, Linux is an operating system which can run applications, but particularly excels at server-based services since it is much more lightweight than other software. In computing, something "lightweight" refers to software designed to have a small memory footprint (RAM), low CPU usage, and low overall usage of system resources. Perfect for something like a server, but less friendly to the average user. Users will typically navigate Linux through the command-line with less focus on graphical applications.

Since Linux is an open-source OS, where anyone can take the base code and manufacture it to their liking, there are many *distributions* designed for specific purposes. For example, in previous projects I worked with *Kali* Linux, a distribution focused on network penetration testing; and *Raspbian*, the official operating system of the Raspberry Pi. In this project, I hopped back on Raspbian to host an Apache Web Server as an end device.

Network Diagram



HTTP & SSH Servers

Lab Summary

I began by designing a topology on packet tracer. Originally, this topology consisted of one router which became two because of later problems. While the main focus was directed toward setting up routing policies, we would first need to configure web servers to give the policies a purpose.

Setting up Apache on Raspbian

We opted with Apache on a Linux-based system for our *http* service. I prepared Apache on my Raspberry Pi while my partner, Harsha Bhat, set up the other on an Ubuntu virtual machine. Installing Apache was straightforward on both machines: *sudo apt-get install apache2*. I followed an official guide by *ubuntu.com*, which provided the minimum configuration necessary to get the service up and running. During the rest of the day, we experimented with *nginx*, an alternative to Apache, installing it on the Raspberry Pi because it was recommended by an acquaintance. After messing around with nginx, we decided to remain with Apache because of its lesser complexity.

When I came back the next day to start up the Apache service on my Raspberry Pi, I encountered an odd error that denied me access when starting the web server: *"job for apache2.service failed because the control process exited with error code"*. I assumed that installing nginx caused some compatibility issues with Apache since Apache worked perfectly prior to the install of nginx. I uninstalled nginx with the command *sudo apt-get --purge remove nginx*, yet the error persisted. After a while, I came across a command that listed open ports currently on the Raspberry Pi: *sudo netstat -ltnp | grep :80*. Much like the *netstat* command on Windows, the output displayed current TCP sockets listening on *port 80* (hence the *grep :80*). A service that was not Apache popped up in the output log. The significance here was that *nothing should have been listening on port 80*.

The Apache server was attempting to listen on port 80 but was being denied because of a service already running on port 80. When Apache could not claim the socket, it threw the *apache2.service failed* error. So, what was listening on the http port? It turns out nginx was still haunting my system, because nginx popped up as the service hogging port 80. I could not figure out how to properly remove nginx, so instead I terminated the socket using the command *sudo kill -9 [id]*. Then I was able to start Apache.

Configuring Policy-Based Routing

With the servers working, it was time to implement the policies. I began by creating an access list rule that permitted all hosts using http to my Raspberry Pi and another permitting ssh to the Ubuntu VM, implicitly denying all other traffic. I applied this access list to the interface of the router connected to the servers for all the traffic exiting. This was all we needed to achieve the purpose of the lab – permitting certain protocols to different servers – however it was not truly *policy-based routing*. For policy-based routing, we needed a route-map to "set" attributes on packets that pass or fail the access-list. The attribute we would set is the *next-hop ip*. To accommodate passing a next-hop ip, we needed to add another router to our topology.

The "internal router" would bridge the policy router and the internal network. We could set the next-hop ip for packets that pass the access list of our policy router as this internal router. After implementing the relevant configuration in both routers, the network was working with a *route-map*.

Improvements

While in the process of writing, I realized there may have been an easier way to implement the policies. This change would only have required one of the routers, instead of both the "Policy Router" and the "Internal Router". Instead of setting a *next-hop ip*, I could have set the *destination address* to the specified servers. This would theoretically eliminate the need for two routers in the topology.

Lab Commands

Command	A statement necessary for a configuration to work, denoted in bold
[*Argument*]	An argument necessary for a command to function, denoted in bold italics.
Optional-Statement *<Optional Argument>*	An optional argument or statement, not necessary for a command to function, denoted in italics

// Static Route

Router(config)# **ip route [*network address*] [*subnet mask*] [*interface*] [*next-hop ip*]**

- Create a static route to a specified destination network

// Access List

Router(config)# **ip access-list [*type*] [*id*]**

- Enter access list configuration mode

The type will typically be either a standard or extended access list. Ids range from [1-99] & [1300-1999] for standard ACLs and [100-199] & [2000-2699] for extended. Ids can just be a unique name.

Router(config-ext-nacl)# *<sequence number>* **permit tcp [*source address*] [*wildcard mask*] [*destination address*] [*wildcard mask*]** *eq <protocol>*

- Add a rule to an access list permitting tcp traffic

The source and destination addresses can be replaced by "host" to then specify a particular host or can be replaced by "any" for all ip addresses. An optional sequence number can be used to determine the order of rules in the access list. An optional protocol can be added to match traffic of the specified protocol.

// Route Map

Router(config)# **route-map [*name*] permit** <*sequence number*>

- Create a route map and enter route map configuration mode

Much like access lists, route maps are read from the first to last sequence number. Other route maps with the same name can do different set operations, like "if branches" in programming.

Router(config-route-map)# **match ip address [*access list id*]**

- Get the route map to match traffic based on an access list

A set operation will be applied to all traffic matched.

Router(config-route-map)# **set ip next-hop [*ip address*]**

- Set a next hop ip for matched traffic

Configurations

PBR Router

```
PBR-Router#show running-config
service timestamps debug datetime msec
service timestamps log datetime msec
no platform punt-keepalive disable-kernel-core
hostname PBR-Router
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
 address-family ipv6
 exit-address-family
no aaa new-model
subscriber templating
multilink bundle-name authenticated
license udi pid ISR4321/K9 sn FDO214421BU
spanning-tree extend system-id
```

```
redundancy
 mode none
vlan internal allocation policy ascending
interface GigabitEthernet0/0/0
 ip address 192.168.1.1 255.255.255.252
 negotiation auto
interface GigabitEthernet0/0/1
 ip address 10.0.1.1 255.255.255.0
 ip policy route-map http-ssh-policy
 negotiation auto
interface Serial0/1/0
 no ip address
 shutdown
interface Serial0/1/1
 no ip address
 shutdown
interface Service-Engine0/2/0
 no ip address
 shutdown
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 shutdown
 negotiation auto
interface Vlan1
 no ip address
 shutdown
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
ip tftp source-interface GigabitEthernet0
ip route 10.0.0.0 255.255.255.0 GigabitEthernet0/0/0 192.168.1.2
access-list 100 permit tcp any host 10.0.0.2 eq www
access-list 100 permit tcp any host 10.0.0.3 eq 22
route-map http-ssh-policy permit 10
 match ip address 100
 set ip next-hop 192.168.1.2
route-map http-ssh-policy permit 20
 set interface Null0
control-plane
line con 0
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
end
```

Internal Router

Router#show running-config

```
service timestamps debug datetime msec
service timestamps log datetime msec
platform qfp utilization monitor load 80
platform punt-keepalive disable-kernel-core
hostname Router
boot-start-marker
boot-end-marker
vrf definition Mgmt-intf
 address-family ipv4
 exit-address-family
```

```
 address-family ipv6
 exit-address-family
no aaa new-model
login on-success log
subscriber templating
multilink bundle-name authenticated
no license smart enable
diagnostic bootup level minimal
spanning-tree extend system-id
redundancy
 mode none
interface GigabitEthernet0/0/0
 ip address 10.0.0.1 255.255.255.0
 negotiation auto
interface GigabitEthernet0/0/1
 ip address 192.168.1.2 255.255.255.252
 negotiation auto
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 shutdown
 negotiation auto
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
ip tftp source-interface GigabitEthernet0
ip route 10.0.1.0 255.255.255.0 GigabitEthernet0/0/1 192.168.1.1
control-plane
line con 0
 transport input none
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
end
```

Conclusion

In this lab, we configured *policy-based routing* to secure two servers in an isolated network. Access lists are a big part of modern security, so it was good to implement them ourselves. In the future I may try to do this lab again but with only one router.

Configuration Guides

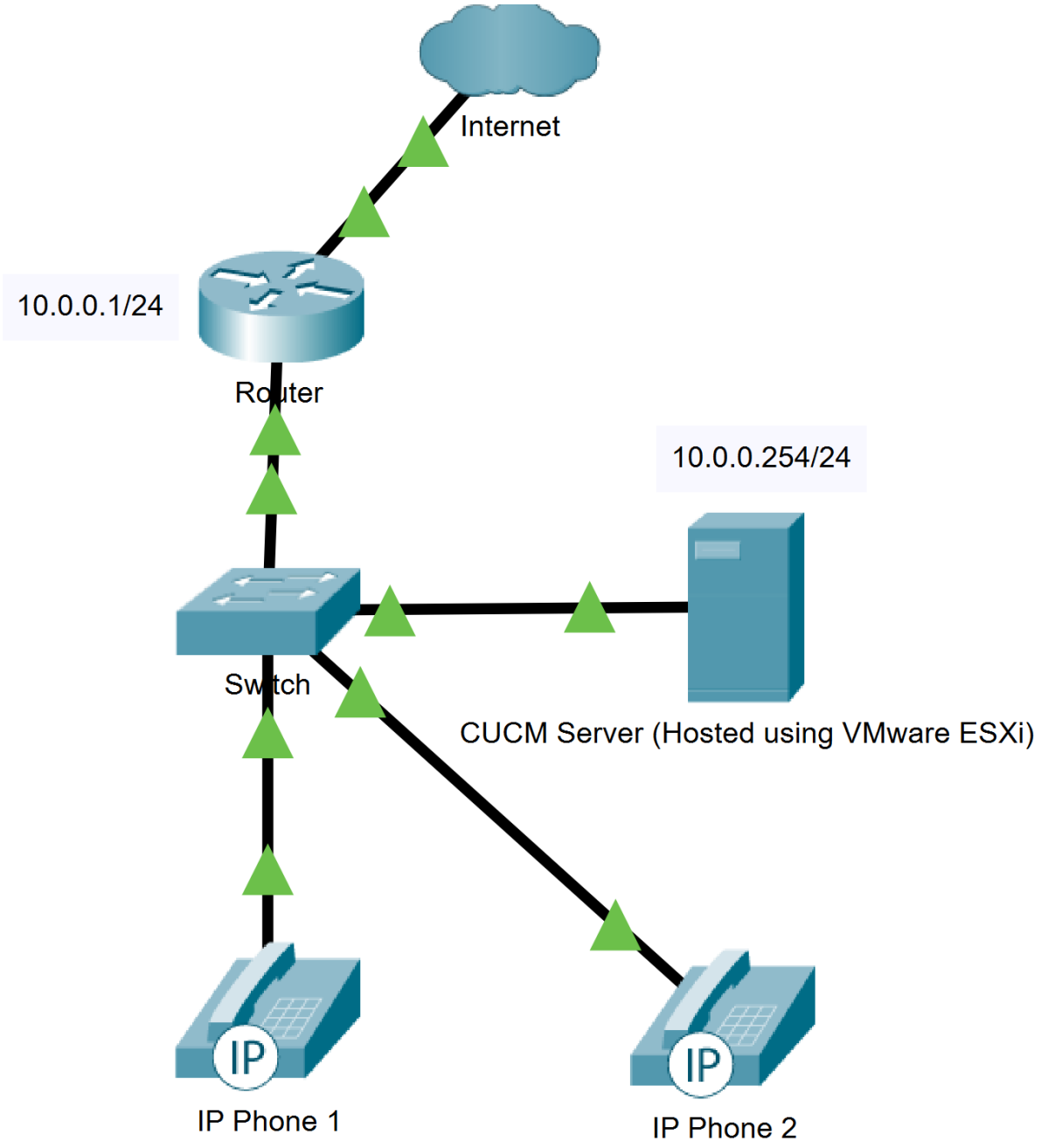
Step by Step tutorials Covering Topics From Previous Chapters

Cisco Unified Communications Manager V12.0.1.21900

By Gabriel Rosas

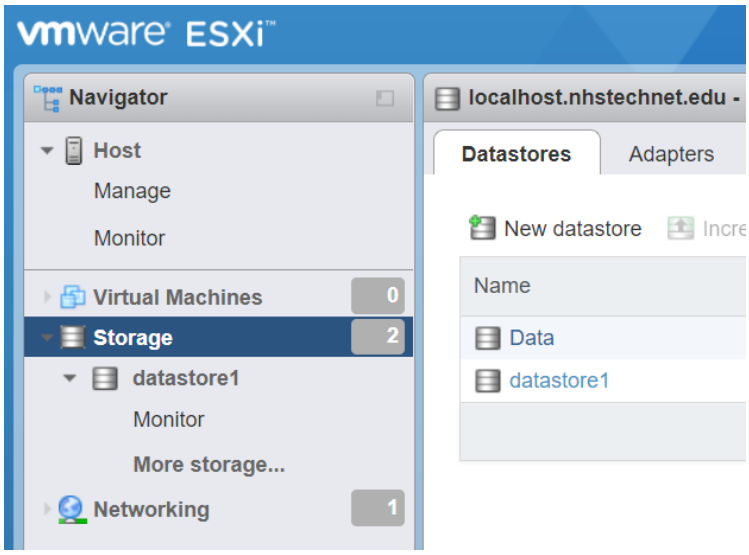
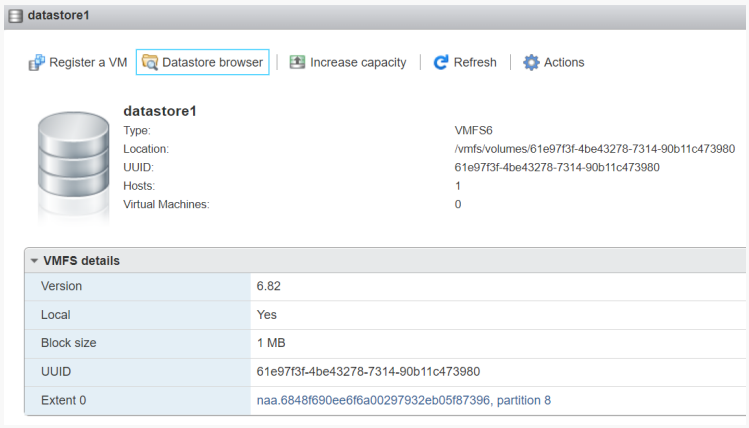
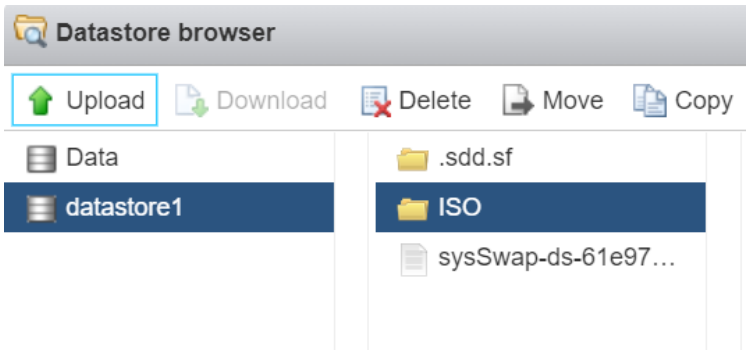
A basic guide for getting IP Phones connected with the Cisco Unified Communications Manager. This guide assumes you have an ESXi Server.

Topology

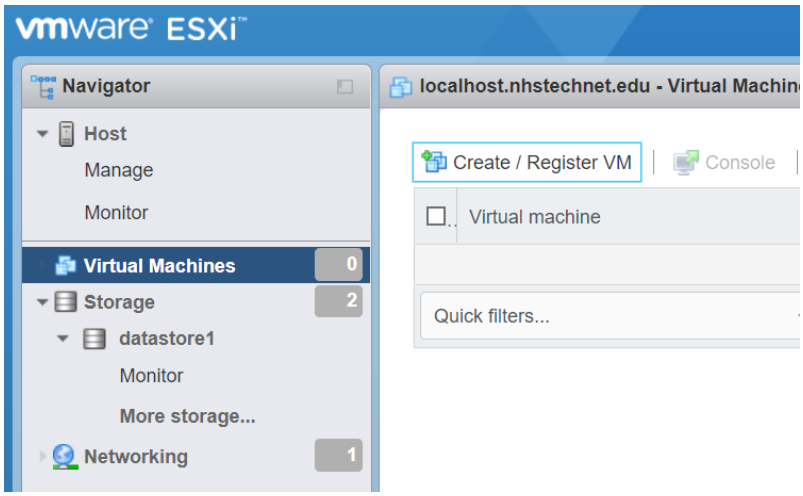
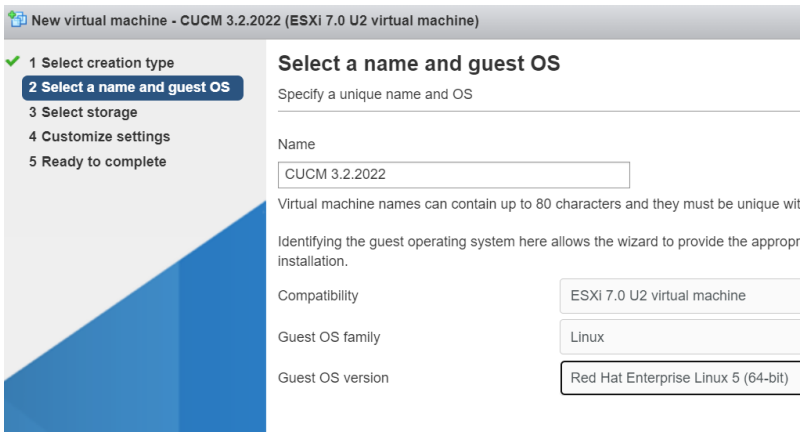


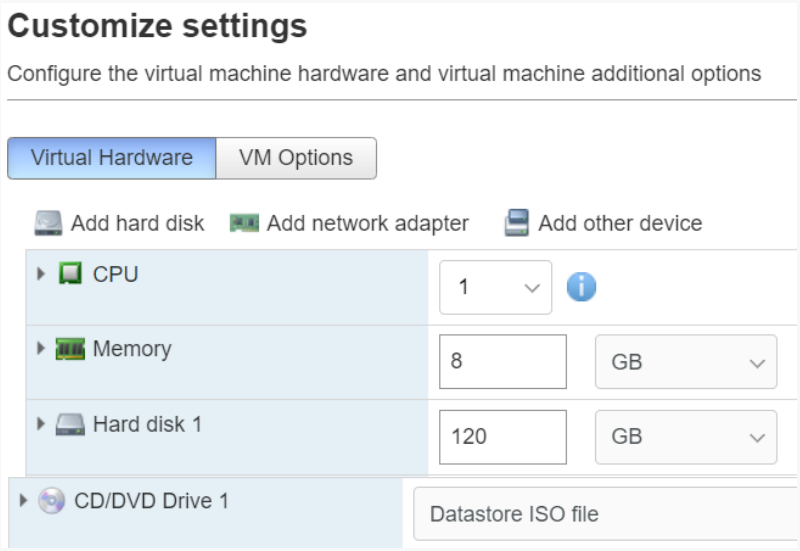
Installing the CUCM on an ESXi Server

1. Downloading the CUCM ISO to local storage

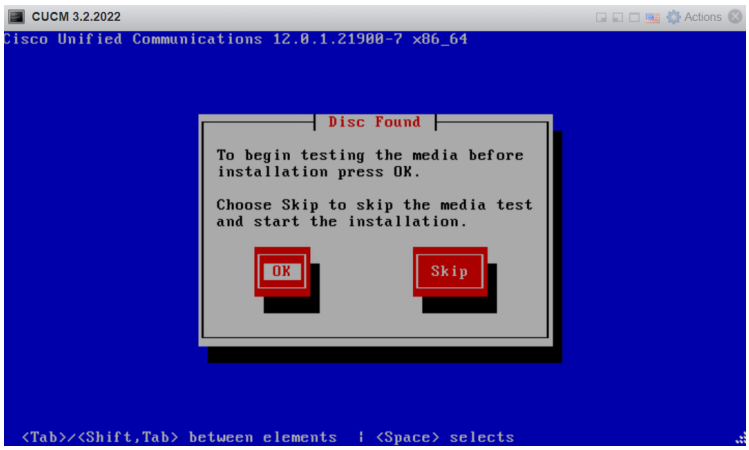
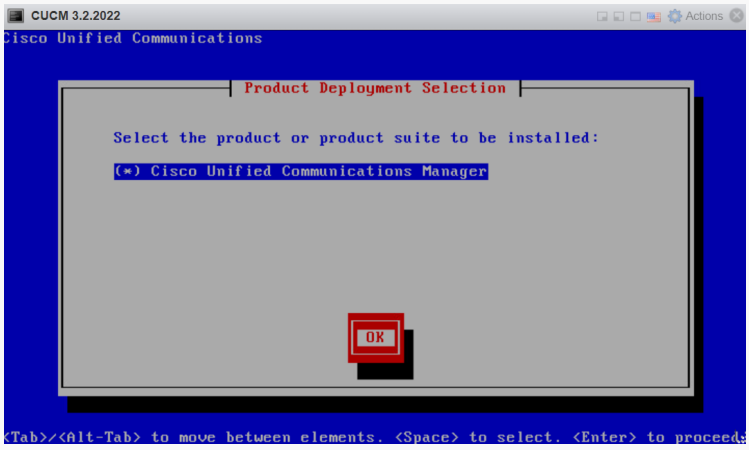
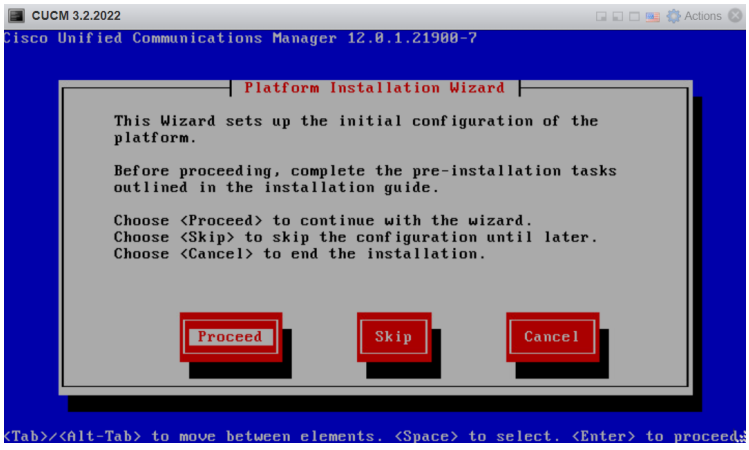
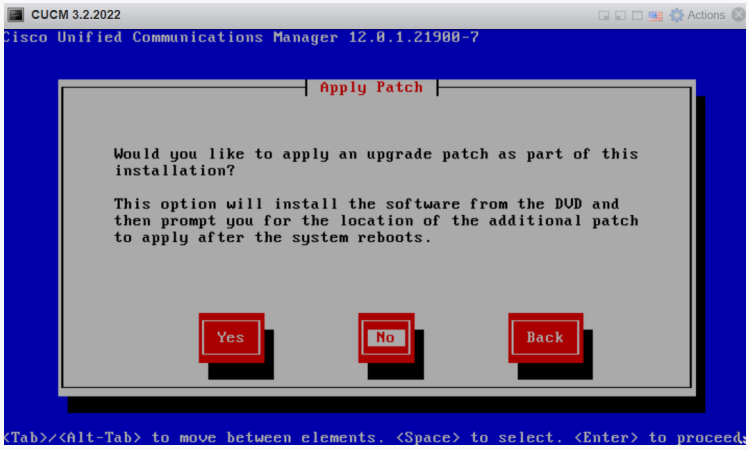
Instruction	Reference
To upload an image, navigate to `Storage`	
Click `Datastore browser`	
In the desired directory, click `Upload`	

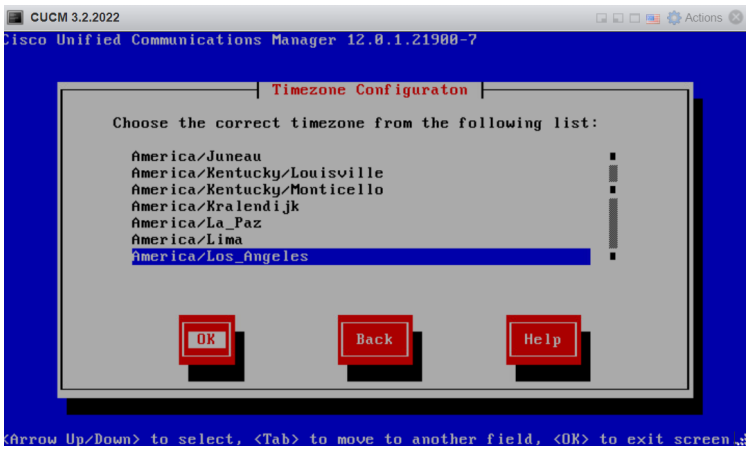
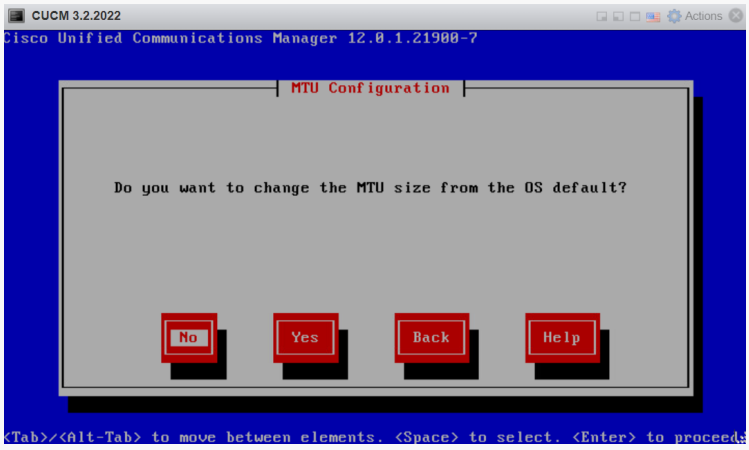
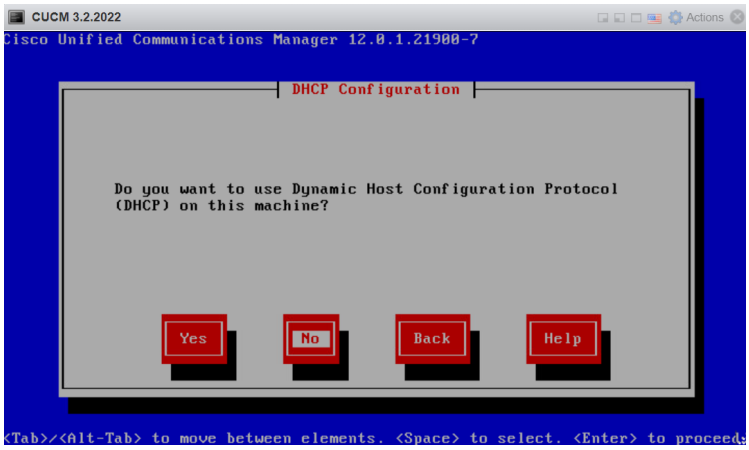
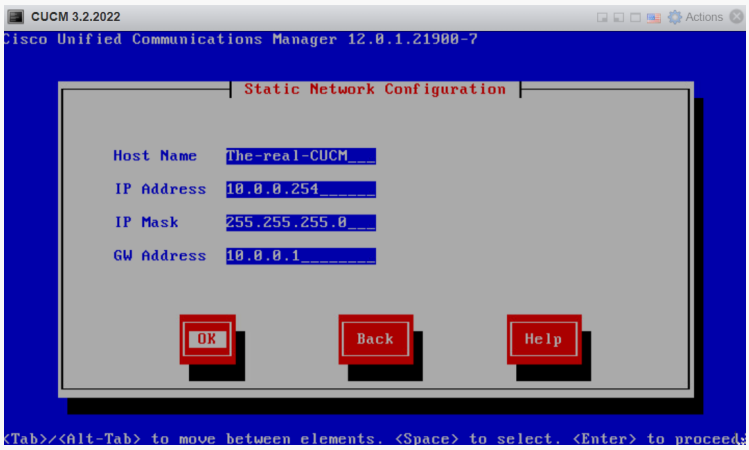
2. Creating a Virtual Machine for the CUCM

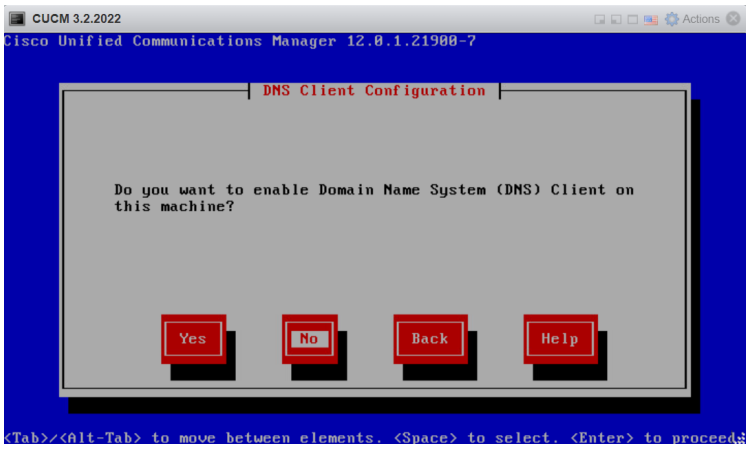
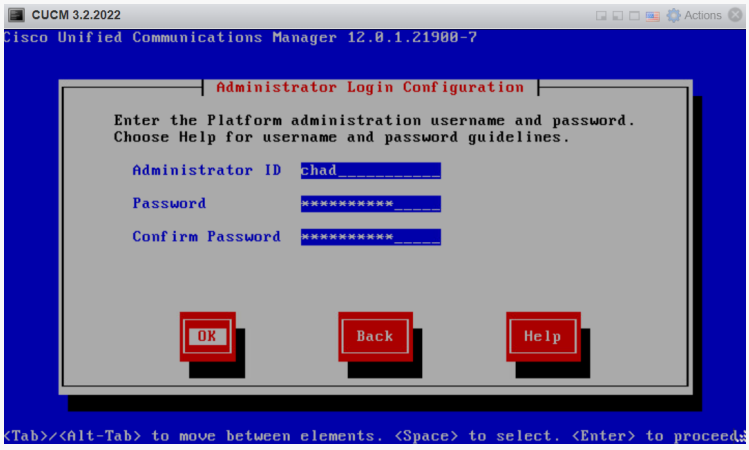
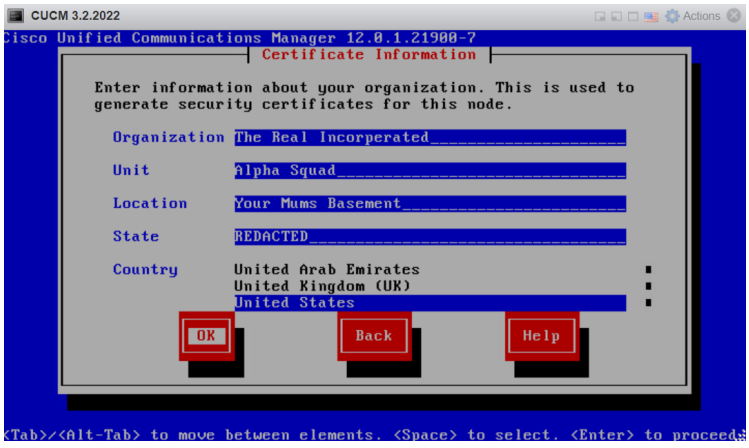
Instruction	Reference
Navigate to `Virtual Machines/Create VM`	
`Create` a new virtual machine	
`Name` the VM and use *Red Hat Enterprise Linux 5* as the `Guest OS`	

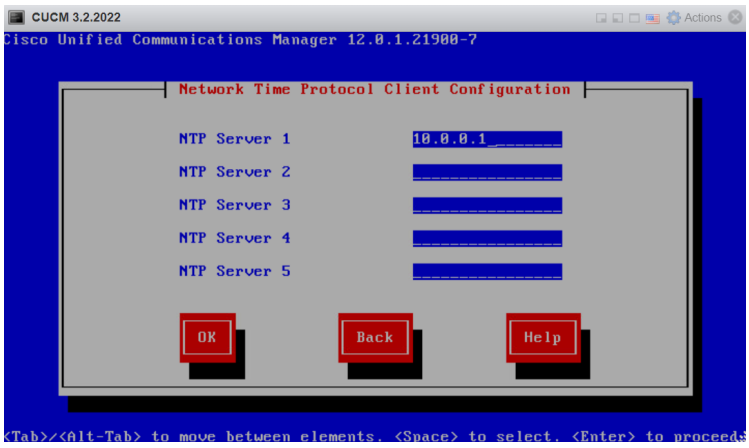
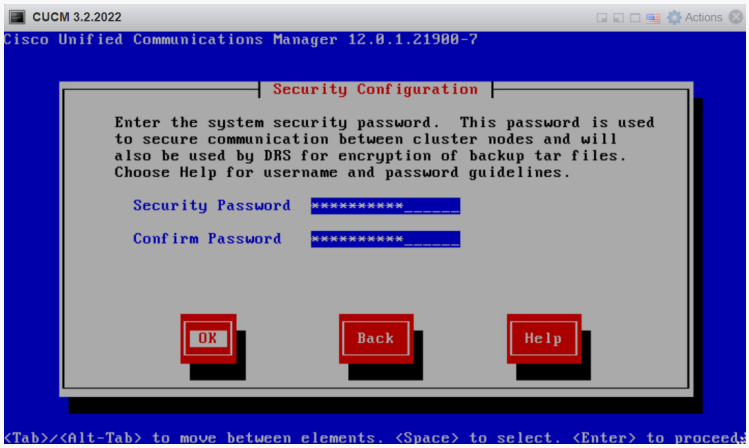
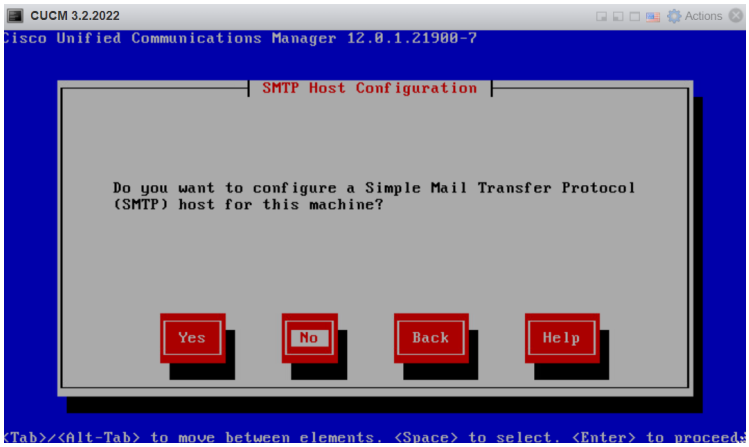
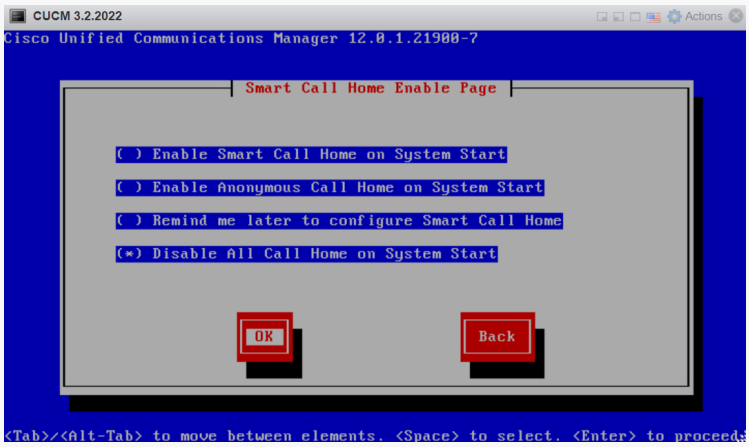
Instruction	Reference
Allocate reasonable amounts of `Memory` and `Storage` [1] Make sure the `CD/DVD Drive` boots the CUCM ISO	

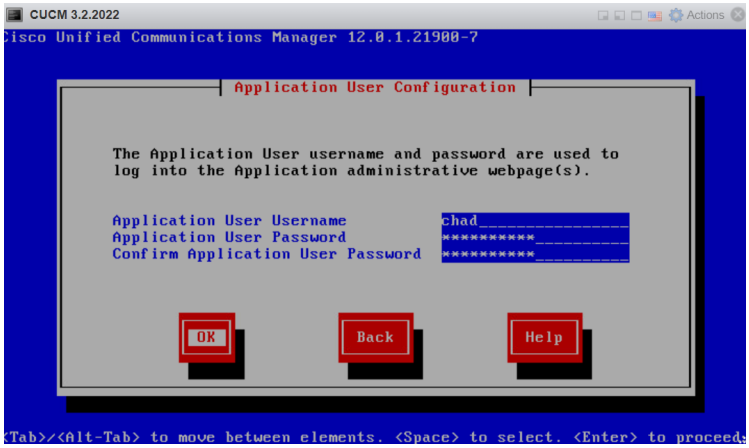
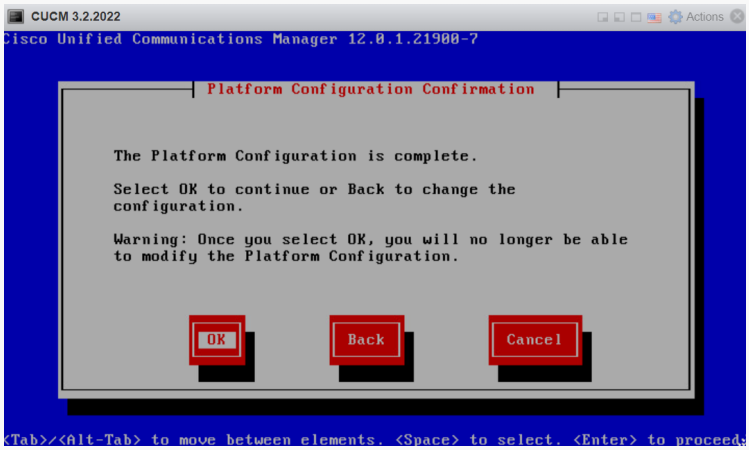
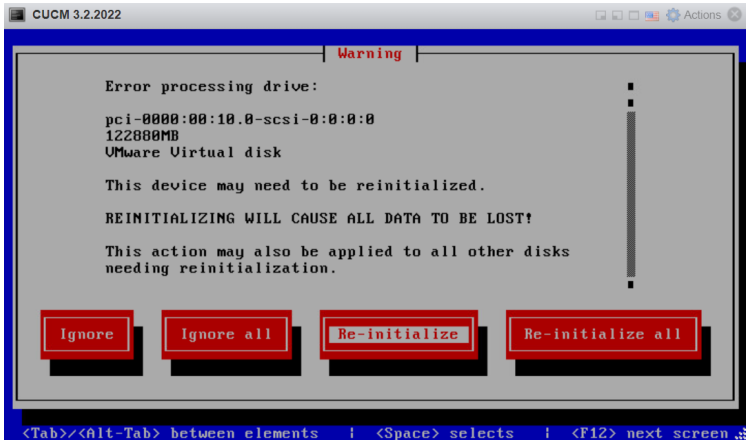
First time Installation

Instruction	Reference
Launch the new VM, Select `OK`	
Select `OK`	
Select `Proceed`	
Select `No`	

Instruction	Reference
Select your `Timezone`	
Select `No`	
Select `No`	
Enter `Static DHCP` information	

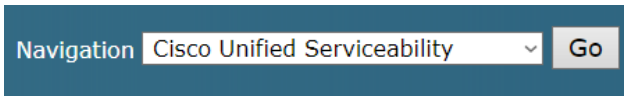
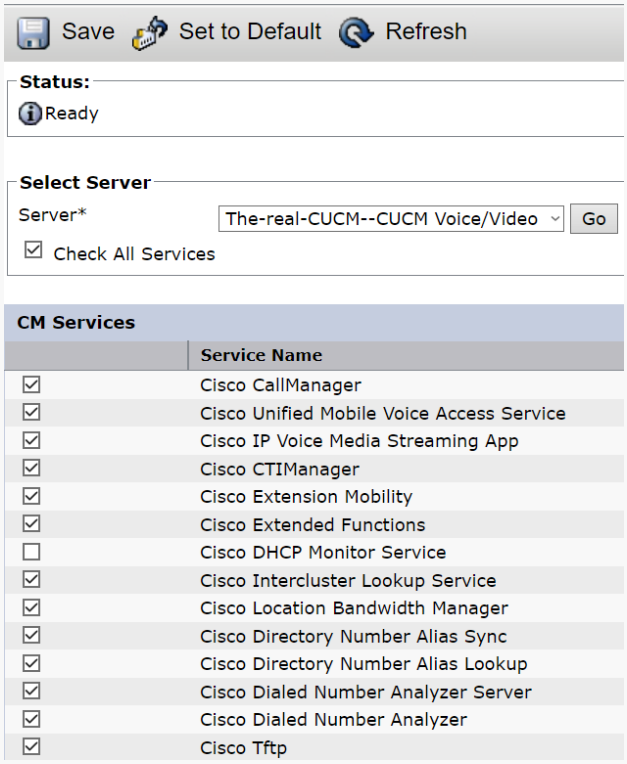
Instruction	Reference
Select `No`	
Enter `Administrative Login` credentials	
Enter `Certificate` credentials	
Select `Yes`	

Instruction	Reference
Enter `NTP Server(s)`	
Enter `Security Password`	
Select `No`	
Select `Disable All`	

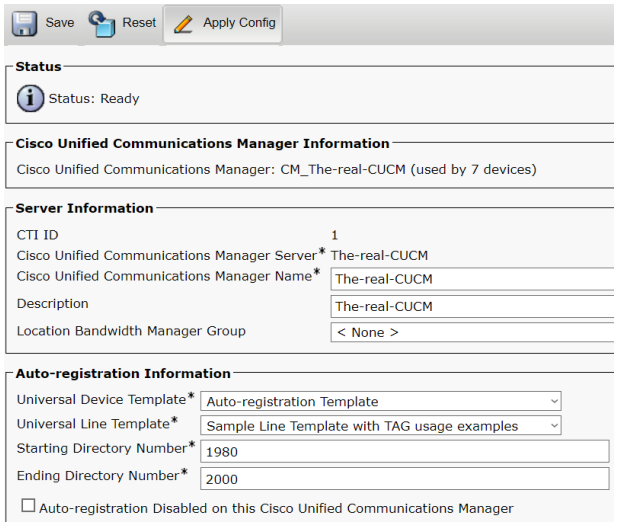
Instruction	Reference
Enter `User` Credentials	
Select `OK`	
Select `Re-initialize`	

Registering Cisco IP Phones

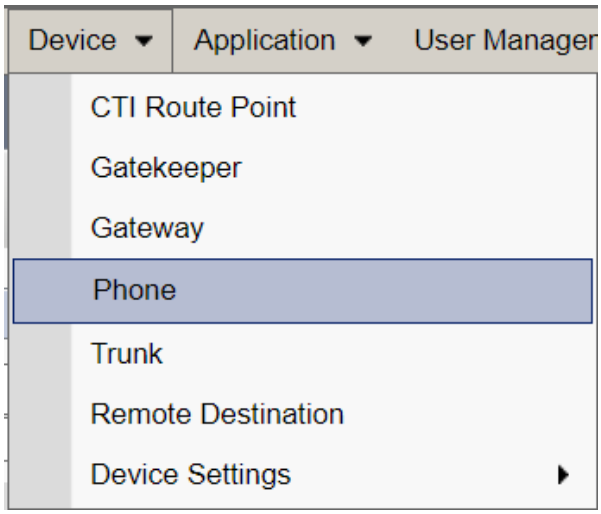
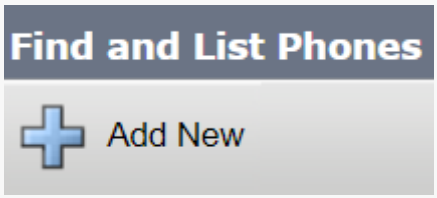
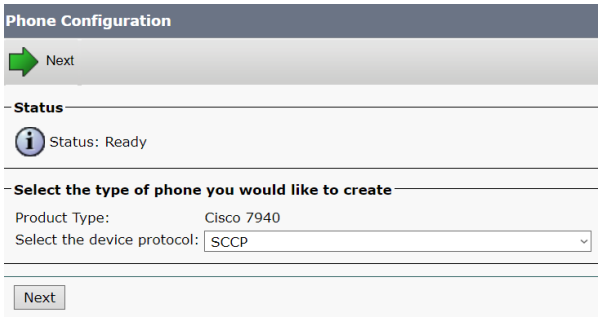
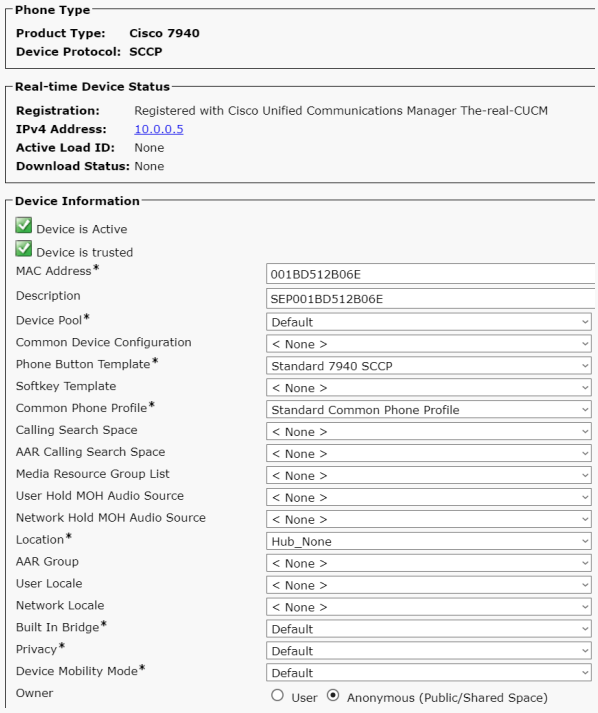
1. Activating Services

Instruction	Reference
Log in to the CUCM using the Static IP in a web browser Navigate to `Cisco Unified Serviceability`	
`Activate` all Services except DHCP	

2. Enabling Auto-Registration

Instruction	Reference
Navigate to `Cisco Unified CM Administration`	
Navigate to `System/Cisco Unified CM`	
Enable `Auto-Registration` and fill out the required fields	

3. Creating Phone Profiles

Instruction	Reference
Navigate to `Device/Phone`	
`Add` a New Phone	
Enter your *specific* model phone and use `SCCP`	
Enter your phone's `mac address` and other required fields Set the Owner to `Anonymous`	

Instruction	Reference
Click `Save`, then add a dial number if necessary Restart previous steps for other phone	

1. The base allocations are insufficient for the CUCM (An error will occur during the boot). Personally, I've found **8 GB** of Memory and **100 GB** of Storage to work well, though I haven't tested other values rigorously.

Securing a Cisco router with FreeRADIUS

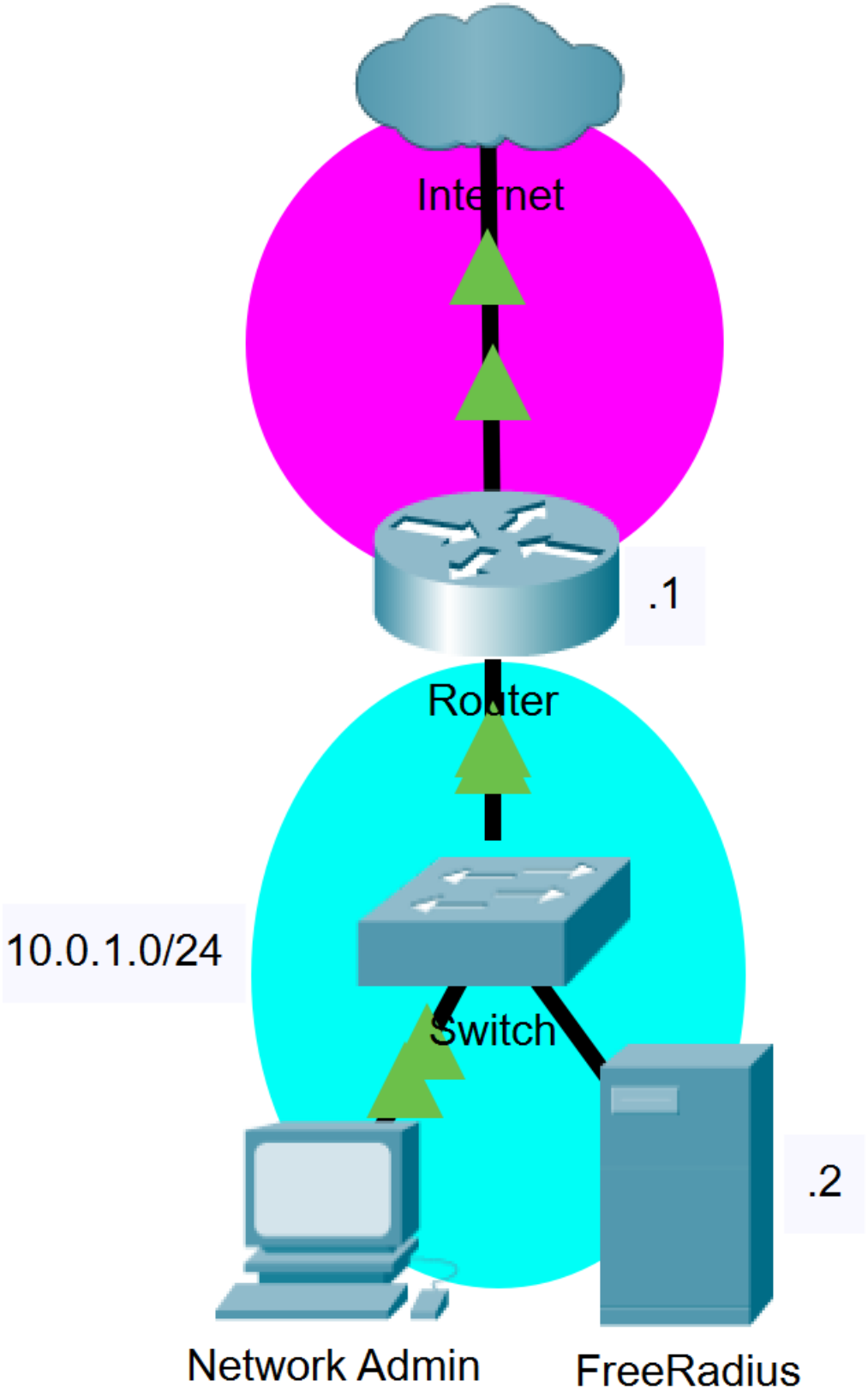
By Gabriel Rosas, in cooperation with Harsha Bhat

This guide will teach you how to secure a Cisco router using Free Radius on a Raspberry Pi. While this guide is intended for configuring Free Radius on a Raspberry Pi (Raspbian), the method should work on other Linux distributions.

Prerequisites

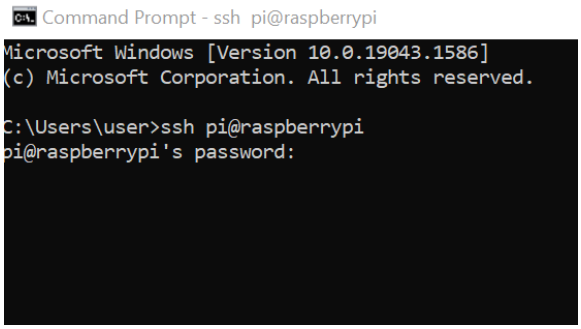
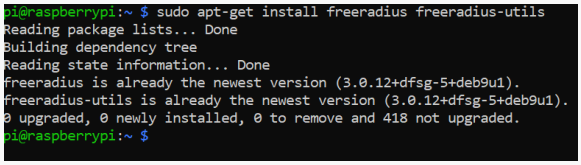
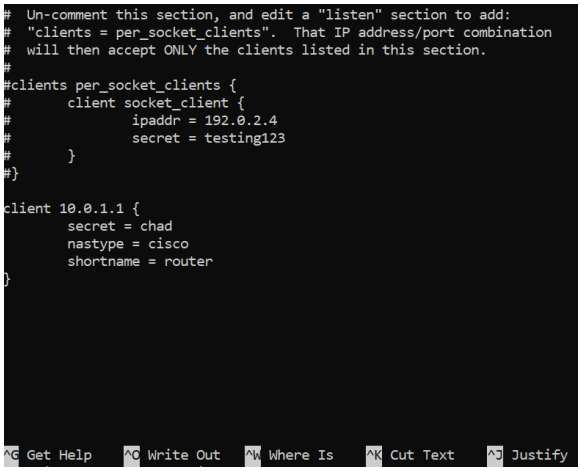
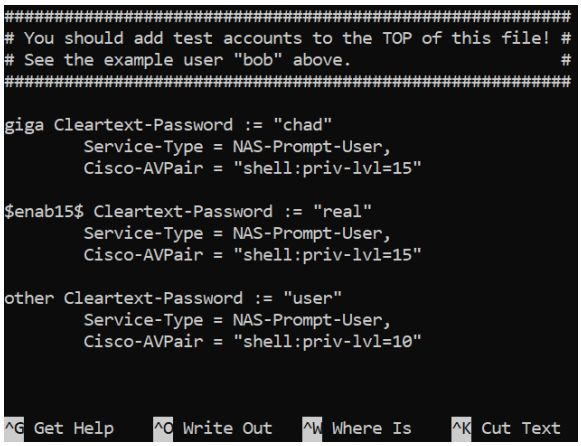
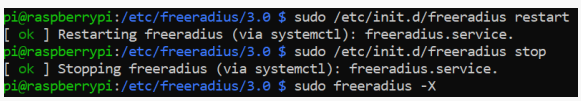
- Have SSH working on the Raspberry Pi.

Topology



Installing FreeRADIUS

Step	Reference

Step	Reference
Log in to your raspberry pi. (Or on Linux, open a terminal).	
Install freeRADIUS using the command `sudo apt-get install freeradius freeradius-utils`.	
Navigate to `/etc/freeradius/3.0`. We will be editing the `clients.conf` and `users` files.	
Edit `clients.conf` using your preferred text editor.	
Scroll to the bottom of the file. Add the client (in my case, the IP of the Cisco router). For example, `client <ip> {}`. Within the braces, define a `secret`, `nastype`, and `shortname`.	
Save `clients.conf` and open `users`.	
Scroll to the bottom of the file. Add the line `<username> Cleartext-Password := "password"` for each unique user. Add the line, `Service-Type = NAS-Prompt-User,` and another for permission `Cisco-AVPair = "shell:priv-lvl=<0-15>"` per user. This will act as an account for an admin to log in with. For enable mode, use `$enab15$` for the username.	
Ensure freeRADIUS is working, then launch debug mode: `sudo freeradius -X`	

If everything went according to plan, we should have a FreeRADIUS server with two accounts and an `enable` password. It is time to add the complementary commands on the router.

Configuring AAA on the router

First we will define a new AAA model, set a banner and fail message, and state we want radius to act as the default login server. We also set the enable mode to use our radius server.

```
aaa new-model
aaa authentication attempts login 5
aaa authentication banner `Stop mortal. Speak thine Words of Entry.`
aaa authentication fail-message `You have failed. Begone.`
aaa authentication login default group radius
aaa authentication enable default group radius
```

Finally, we define the radius server that the router should use. The IP should be the radius server and the key is the one we defined in `clients.conf`.

```
radius server PI_RADIUS
 address ipv4 <ip> auth-port 1812 acct-port 1813
 timeout 30
 retransmit 3
 key chad
```

And that's it! Hopefully everything is working as expected.

Securing a Cisco router with TACACS+ on Windows Server 2019

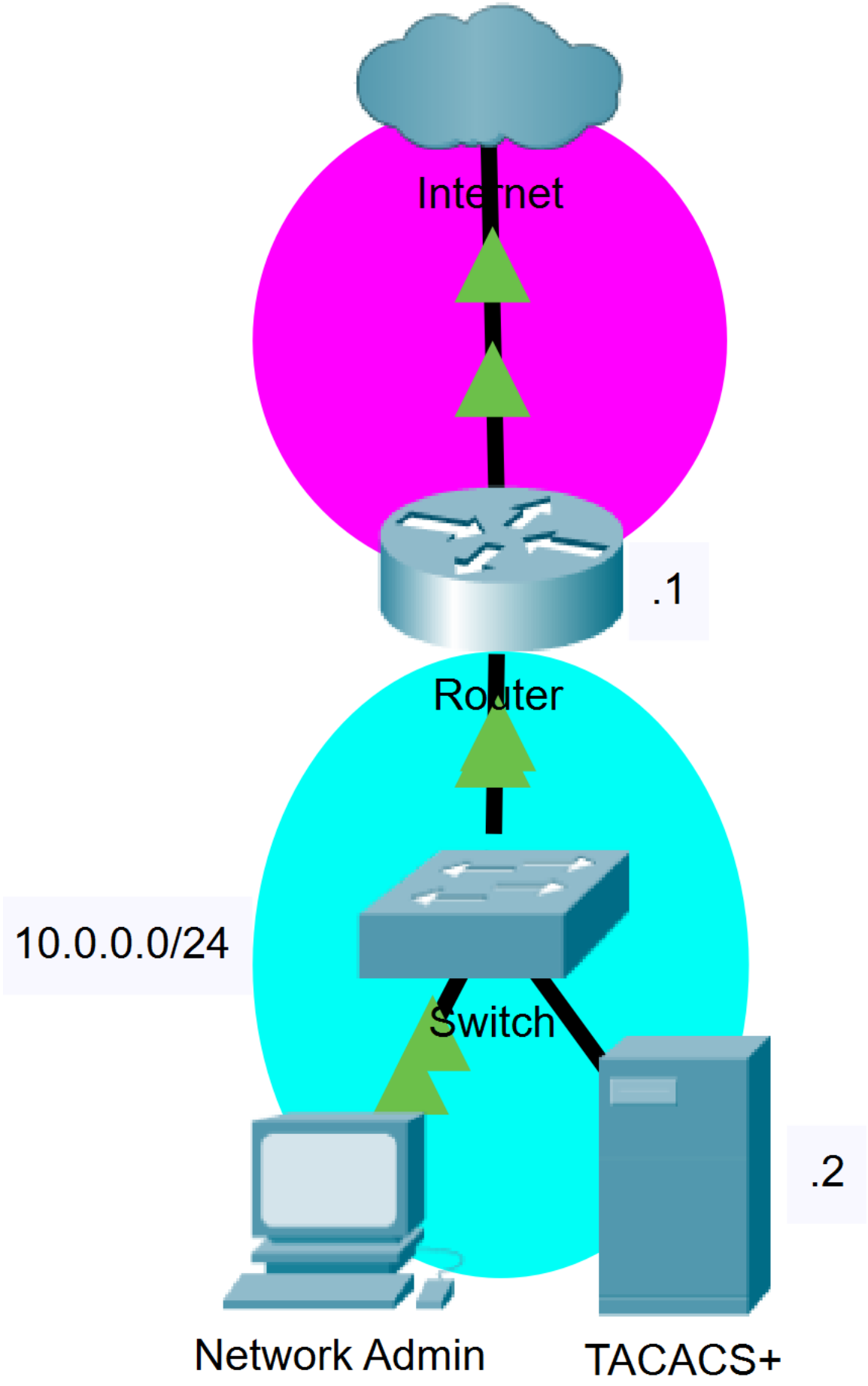
By Gabriel Rosas

This guide will teach you how to secure a Cisco router with TACACS+ running on Windows Server 2019. I used a VirtualBox VM of Windows Server 2019 in this guide, but it should work the same regardless of VM.

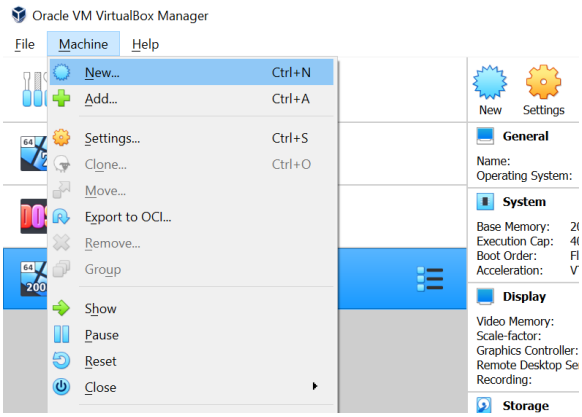
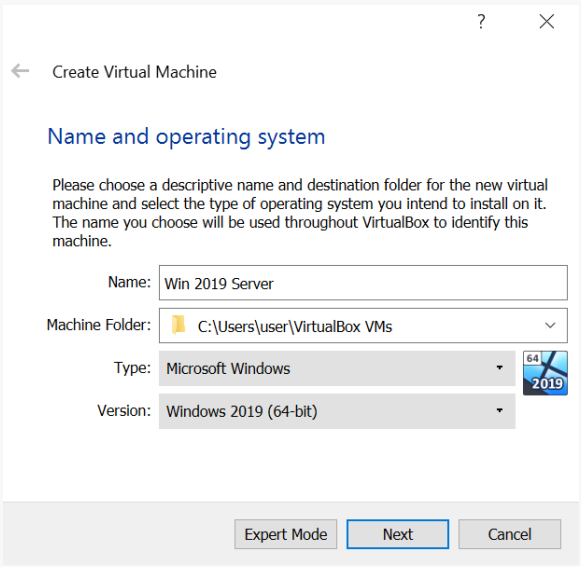
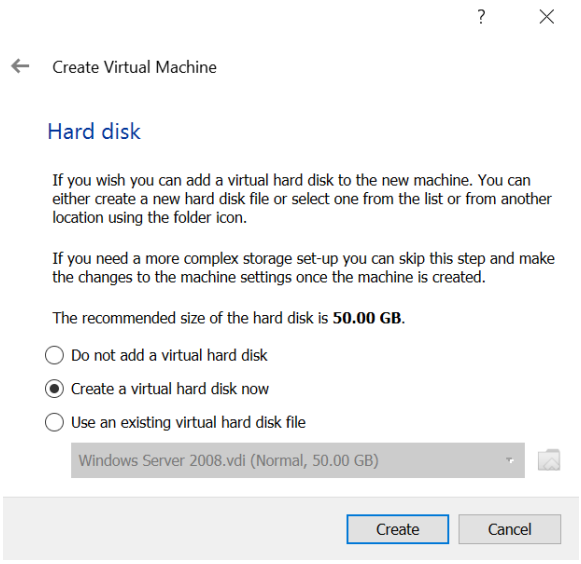
Prerequisites

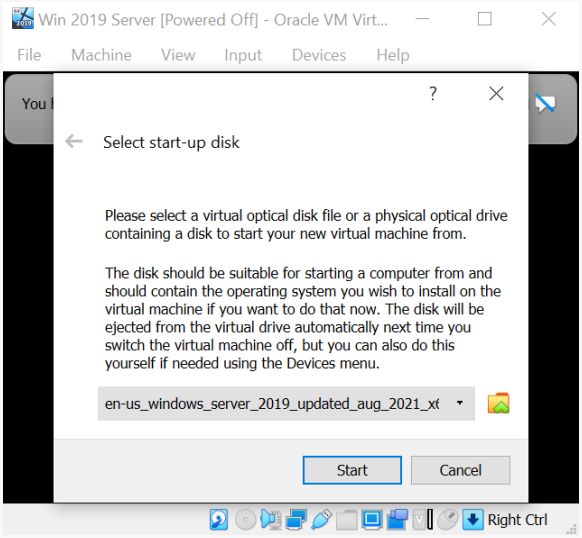
- None

Topology

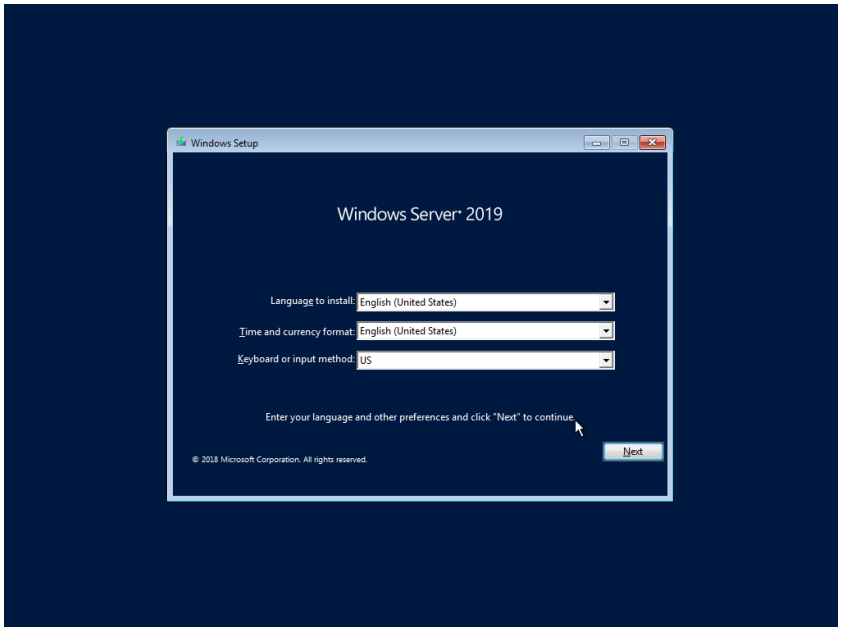


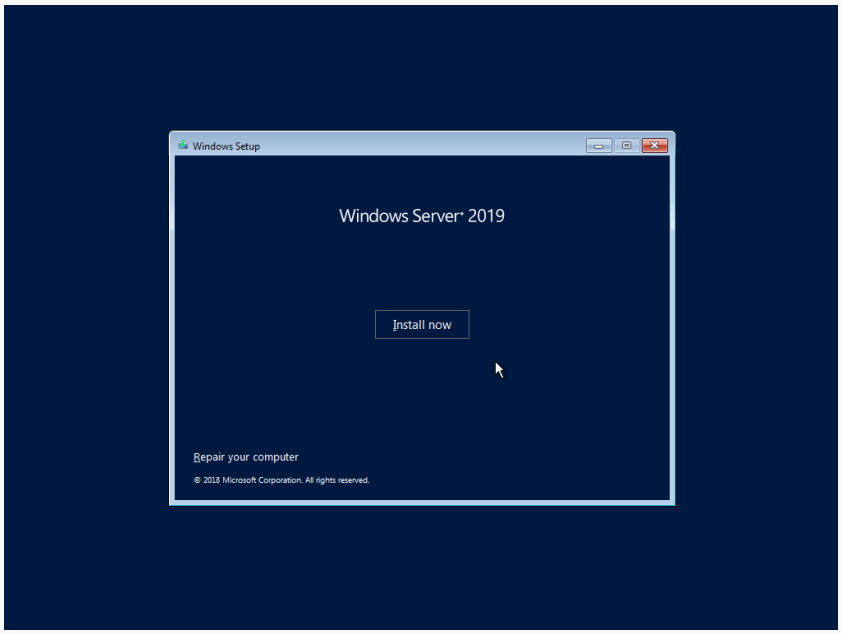
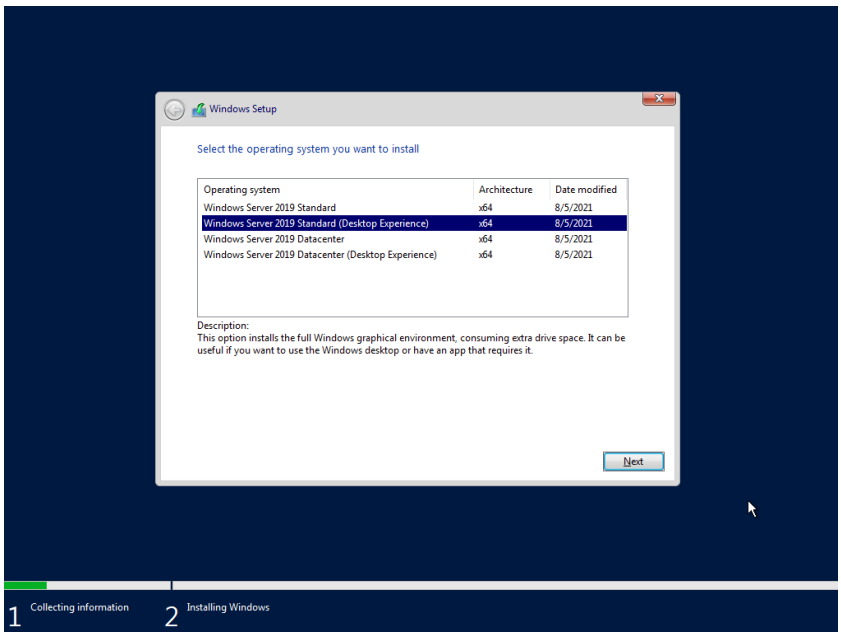
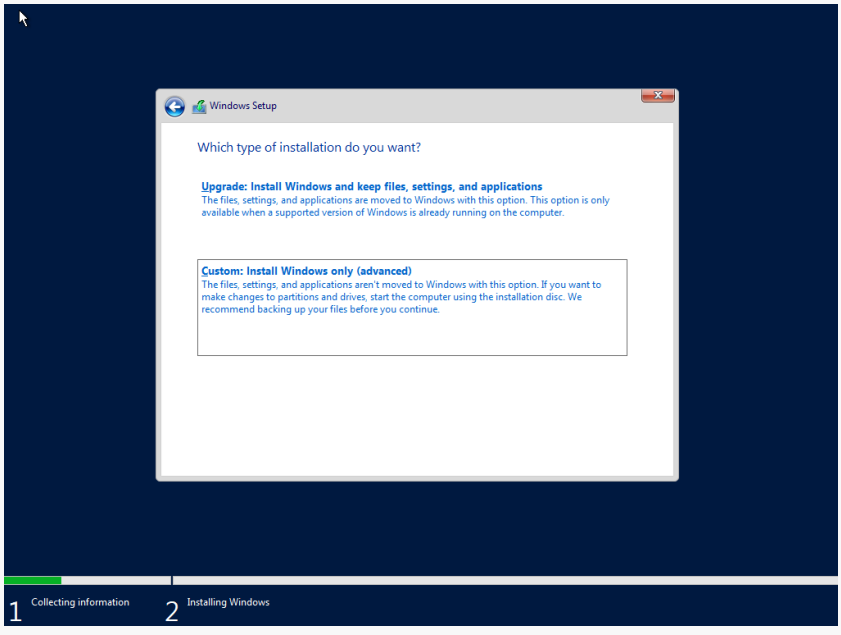
Setting up a VirtualBox VM

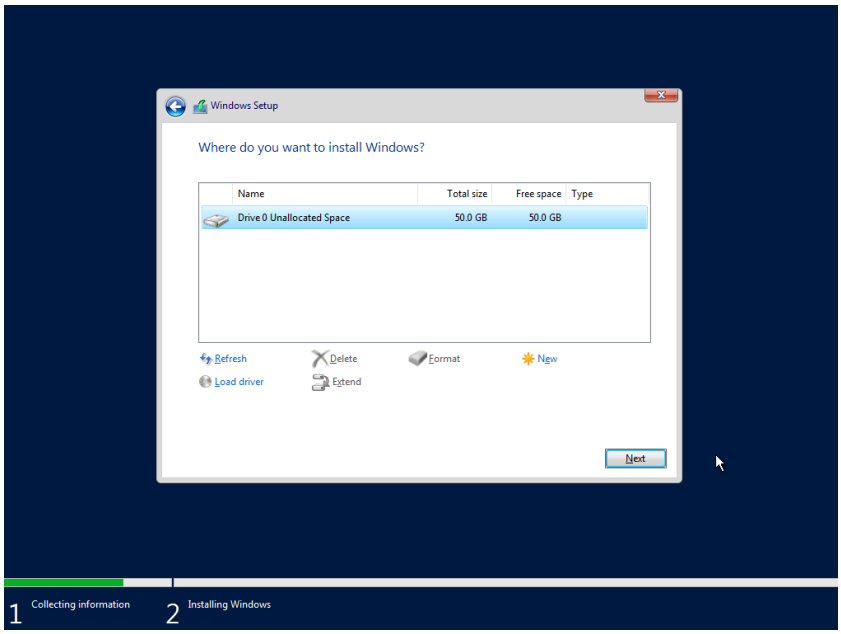
Step	Reference
Install and open VirtualBox. Navigate to `Machine>New` to create a new machine.	
Name and select the version.	
Create a Disk Drive and allocate a reasonable amount of space (50GB is more than enough).	

Step	Reference
After the disk has been created, launch the VM. A prompt should appear. Enter the Windows Server ISO file location.	

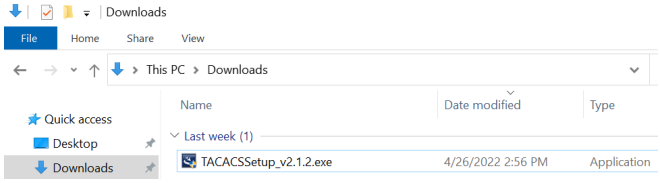
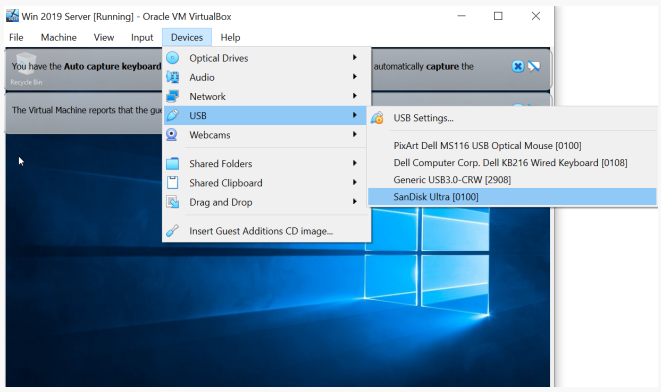
Installing Windows Server

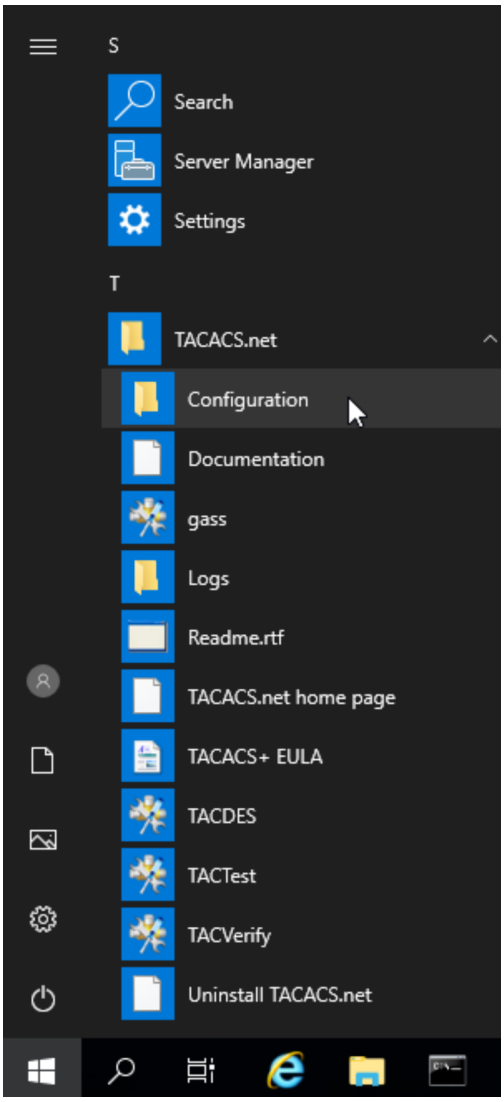
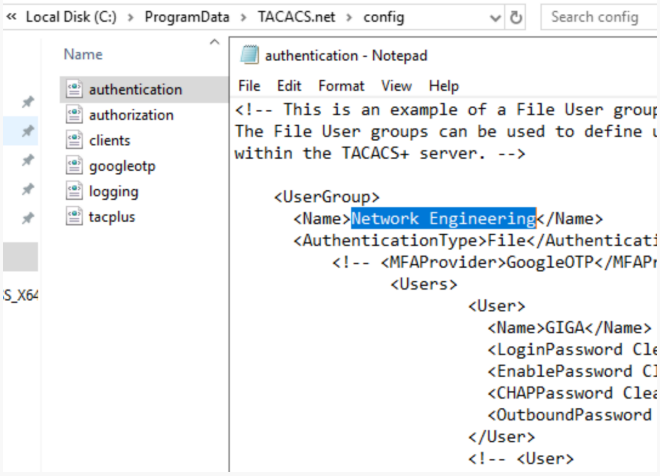
Step	Reference
Set language preferences. Navigate with `TAB` and `ENTER`.	

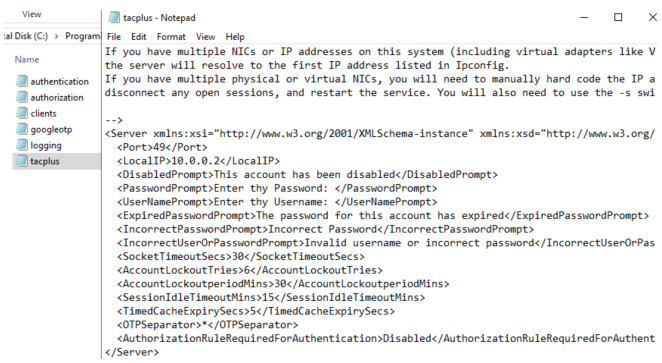
Step	Reference
Hit `Install Now`.	
Choose either of the Standard versions. I chose the Graphical (Desktop) version.	
Do a custom install.	

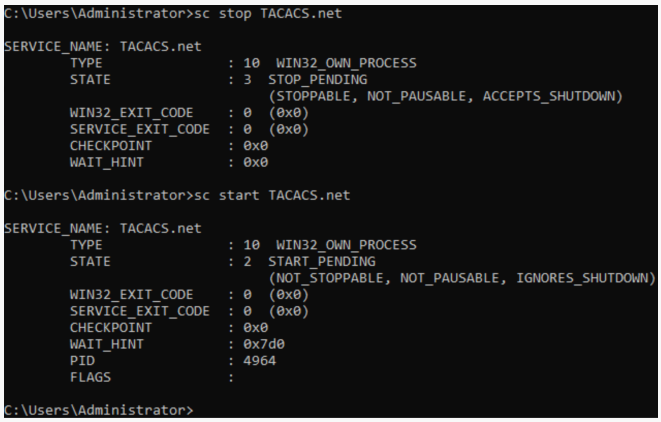
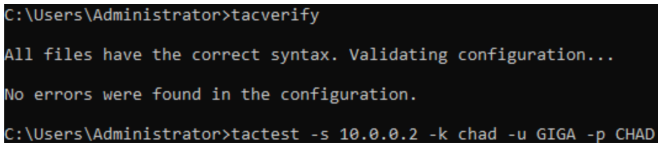
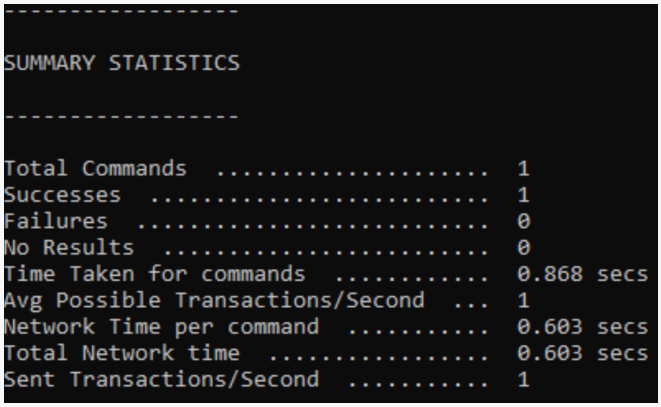
Step	Reference
Select the partition available and continue. The install should now complete.	

Installing and running a basic TACACS+ Service

Step	Reference
Download the TACACS+ service from [TACACS.net]().	
To transfer a file from the host to the guest machine, I recommend using a USB drive. For USB 3.0 support, install and run the VirtualBox Extension pack found here. Run the TACACS exe. You should be prompted to enter a key. Remember that key, we will need it later.	

Step	Reference
TACACS.net services should now be installed. Navigate to the TACACS configuration folder.	
There will be a couple `xml` files in the `TACACS` folder. We will edit `authentication`, `authorization`, `tacplus`. Start by opening `authentication.xml` and locate `UserGroup`.	

Step	Reference
Much like HTML, information is embedded within editable UML tags. Users are defined like `<User> ... </User>` with internal tags containing information about the user. Feel free to change the `<Name>`, `<LoginPassword>`, and `<EnablePassword>` tags to fit the username and passwords you desire. Note that comments are in the form: `<!-- comment here -->`. The users may be commented out, and you may have to uncomment them.	
Open `authorization.xml` and locate the `<UserGroups>` section. You should see a UserGroup called "Network Engineer", exactly like that in `authentication.xml`. There are a lot of interesting configurable features here, though we will only focus on permitting all commands. In the `<Shell>` tag, and add `<Permit>.*</Permit>` to allow the user to enter any command.	
Finally, open up `tacplus.xml`. Change `<LocalIP>` to contain the IP of the TACACS+ Server (most likely the local machine). Feel free to edit any of the prompts. We need to restart the TACACS+ service for this specific change to take effect.	

Step	Reference
In a command prompt, enter `sc stop TACACS.net` to stop the service and `sc start TACACS.net` to start it.	
We can use `tacverify` to check our files for syntax errors and test our TACACS+ service using `tactest`. I recommend running both these commands to make sure everything is working.	
Tactest example: `tactest -s <IP> -k <Server key> -u <Username> -p <Password>`. A success should look something like this:	

That concludes the TACACS+ server-side configuration. Now lets move on to the router.

Basic TACACS+ configuration on the Cisco Router

Define a new AAA model and use tacacs+ for authentication.

```
aaa new-model
aaa authentication login default group tacacs+
aaa authentication enable default group tacacs+
```

Use `tacacs server <name>` to create a new tacacs server. Add an IP and key. You defined the key when you ran the tacacs exe, though you can find it in `clients.xml` if you forgot.

```
tacacs server <Arbitrary Name>
 address ipv4 <IP>
 key <Server Key>
```

And that's it! The router should now request authentication and authorization from the TACACS+ Server.