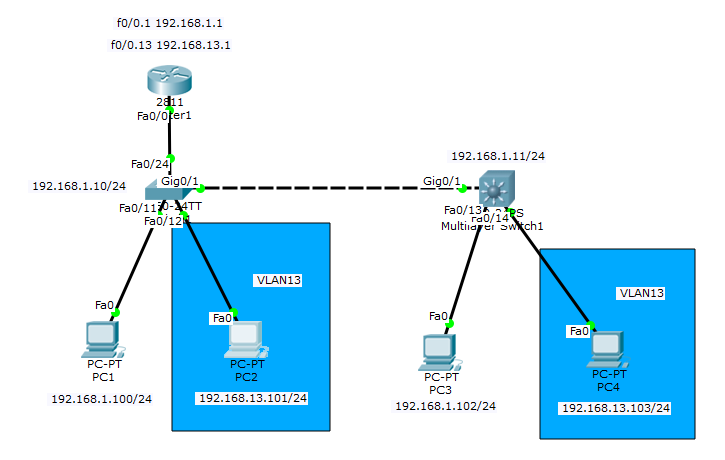
1. VLANs, VTP, and IEEE 802.1q (NOTE: Make sure you accomplish this ***without*** going into VLAN Database mode and show the syntax for both 2960 and 3560 switches!)

To configure the question above I used 2 switches, a router, and 4 PCs. VLANs are making groups of PCs into a subnetwork so the PCs in one VLAN can only communicate with those in the same VLAN group. To accomplish the question at hand we assigned IPs to each computer, and interface needed as well as used sub interfaces on f0/0 of the router so we can have multiple IP addresses pointed towards that one port. The VTP and IEEE 802.1q comes into play when configuring the switches. The 2960 switch used was configured as the server in the VTP configuration. Additioanlly, the 2960 switch does not needed to be configured to 802.1q because it has the configuration already on it. The 3560 switch needed the IEEE configuration added and was treated as the VTP client in this topology.



|  |  |
| --- | --- |
| Basic Commands | Description |
| Router>enable | Enters a user into privileged mode |
| Router#Configure Terminal | Enters a privileged user into the configuration mode |
| Router(config)#hostname Router1 | Sets the hostname of the Router to Router1 |
| Router(config)#enable secret cisco | Sets an encrypted password when a user tries to enter Privileged mode |
| Router(config)#line con 0 | Enters into the configuring of the lines when booting into CLI |
| Router(config-line)#exec-timeout 0 0 | Sets the timeout to never timeout |
| Router(config-line)#password cisco | Sets the password at the base line in CLI |
| Router(config-line)#logging synchronous | This prevents every logging output from interrupting your console session |
| Router(config-line)#login | Enables a login prompt at base line of CLI |
| Router(config)#line vty 0 4 | Enters into the configuring of lines 0-4 in TELNET |
| Router(config-line)#exec-timeout 60 0 | Sets the timeout to 60 minutes |
| Router(config)#int f0/0.1 | Enters selected physical interface or subinterface |
| Router(config-if)# [ip address] [Subnet Mask] | Sets the ip address of the interface selected |
| Router(config-if)# no [ip address] | Deletes current ip address or ip address selected |
| Router(config-if)# no shutdown | Turns on the selected interface |
| VTP Commands | **Descriptions** |
| Switch(config)# vtp mode [server/client] | Sets the VTP mode |
| Switch(config)# vtp domain [INETLAB] | Sets the VTP domain |
| Switch(config)# vtp password cisco | Sets the VTP password |
| VLAN Commands | **Description** |
| Switch(config-if)# switchport mode access | Enables device to carry traffic in a VLAN |
| Switch(config-if)#switchport access vlan # | Sets the interface to specified VLAN |
| Switch(config)int vlan # | Enters specified VLAN |
| Switch(config-if)#ip address [ip address] [subnet mask] | Sets ip address of VLAN |
| Switch(config-if)#no shutdown | Enables the VLAN |
| Switch(config)#vlan # | Enters VLAN Group |
| Switch(config-vlan)#name BLUE | Sets VLAN name to BLUE |
| IEEE 802.1q | **Description** |
| Switch(config-if)switchport trunk encapsulation dot1q | Sets Trunking encapsulation to 802.1q standards |
| Switch(config-if)switchport mode trunk | Enables the current interface to trunk |

|  |  |
| --- | --- |
| Show Commands | Explanation |
| Switch#show ip int brief | Shows current interfaces enabled and IP addresses set to those interfaces |
| Switch#show vlan | Shows table of VLANs, current interfaces in each, and names |
| Switch#show run | Shows current running configuration for the device |
| Ping [ip address] | Pings selected ip address |

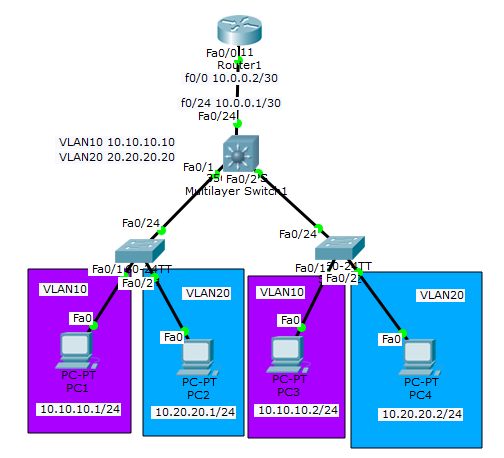
Using these show commands I was able to verify each interface had the correct IP address as well as each VLAN had the correct interfaces associated with them. Ping was the final reveal if the VLANs worked. Pinging from PC1 to PC3 shows connectivity, but pinging from PC1 to PC2 or PC4 should not show connectivity and vice versa for PC2 and PC4.

**Debug Syntax:**

**Show run:** Show run was a huge debugging, one of my errors was I didn’t put an interface into trunking mode which lead to the whole network not being able to trunk. Once that was resolved it worked like a charm.

**Show vlan:** Show vlan was used in debugging if my vlans were created and named properly as well as making sure the interfaces I wanted were shown in the VLAN.

1. InterVLAN Routing with a Router-on-a-stick and SVIs

In this problem, we are asked to configure a “Router on a stick” that has interVLAN routing and SVIs. The point of a VLAN is to have connectivity to other end devices in the same VLAN and those only. InterVLAN routing makes it so if PC1 is in VLAN11 and PC3 is in VLAN13 they can still connect to each other. In addition to this, to configure an SVI I utizilied a 3560 switch because of its multilayer capabilities. SVI makes it so a switch can route connections. This is not capable on a 2960 switch because it is strictly layer 2, which means we need a 3560 switch to configure an SVI on.

|  |  |
| --- | --- |
| SVI Commands | Description |
| Switch3(config)ip routing | Enables ip routing on the multilayer switch |
| Switch3(config)#int f0/24 | Enters interface going to router |
| Switch3(config-if)#no switchport | Turns off switchport |
| Switch3(config-if)#ip address [ip address] [subnet mask] | Sets IP address for interface going to router |
| Switch3(config)#int vlan # | Enters interface of specified VLAN |
| Switch3(config-if)#ip address [ip address] [subnet mask] | Sets VLAN IP address |

|  |  |
| --- | --- |
| Show Commands | Explanation |
| Switch#show vlan | Shows table of VLANs, current interfaces in each, and names |
| Switch#show int vlan # | Shows table of current config on a VLAN # |
| Switch#show run | Shows current running configuration for the device |
| Ping [ip address] | Pings selected ip address |

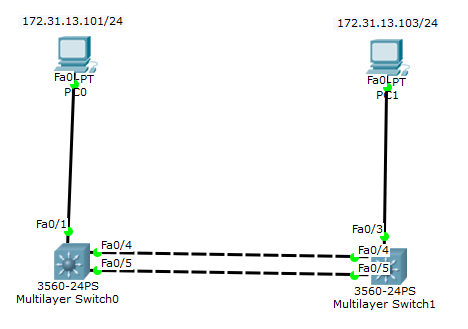
To test validity of this topology the main goal is to be able to ping from 1 vlan end device to another vlan device. In addition to that, the SVI will be doing the routing rather than the router. I used the above show commands to check each device and VLAN. Finally to test, I pinged from VLAN10 to VLAN20 using PC1🡪PC3 and PC4🡪PC2.

**Debug Syntax:**

**Show vlan:** I used this command a lot to see if my VLANs were grouped correctly and to make sure the correct interfaces were assigned to the groups they needed to be.

**Show run:** This was a major one in seeing if the configurations went through because a big problem was that I could not see what the sub interfaces were configured to, using show run I could see the sub interfaces and how they were configured so I could make changes accordingly.

1. First Hop Redundancy Protocol with HSRP using Link Aggregation between two 3560 multilayer switches. Demonstrate your knowledge of LACP and PAgP.

In this problem we are focusing on HSRP and the channel protocols LACP and PAgP. To complete this I used 2 3560 switches and 2 PCs. The end goal of this is to be able to ping to each PC. To get to this goal we utilized First Hop Redundancy Protocols LACP and PAgP. Using these protocols and HSRP we achieved full connectivity.

|  |  |
| --- | --- |
| PAGP Commands | Description |
| Switch(config-if)#channel-protocol pacp | Declares the protocol |
| Switch(config-if)#channel-group # mode desirable | Command assigns the interface to an etherchannel and declares what protocol will be used. |
| Switch(config)#int port-channel # | Creates a portchannel is one does not exist, and enters into that channel |
| Switch(config-if)#switchport trunk encap dot1q | Assigns 802.1q to portchannel |
| Switch(config-if)#switchport mode trunk | Sets etherchannel to trunking |
| Switch(config-if)#switchport native vlan # | Sets the native vlan to # specified |
| LACP Commands | **Description** |
| Switch(config-if)#channel-protocol lacp | Declares the protocol |
| Switch(config-if)#channel-group # mode active | Command assigns the interface to an etherchannel and declares what protocol will be used |
| Switch(config)#spanning-tree vlan # root [primary/secondary] | Enables the spanning tree to be primary or secondary based on the switch being configured. |
| HSRP Commands | **Description** |
| Switch(config)#ip routing | Enables routing on switch |
| Switch(config)int vlan # | Enters interface of the VLAN |
| Switch(config-if)#ip address [ip address] | Sets ip address for interface |
| Switch(config-if)#standby [vlan #] ip [ip address] | Sets IP address for standy, default gateway |
| Switch(config-if)#standby [vlan #] priority [105/115] | Sets standby priority by number listed |
| Switch(config-if)#standby [vlan #] preempt | Enables the HSRP with the highest priority to immediately become the active router |

|  |  |
| --- | --- |
| HSRP, LACP, PAGP Show Commands | Explanation |
| Switch#Show etherchannel sum | Shows summary of all etherchannels on the switch and state of devices in the channels. |
| Switch#Show standy | Shows status of standby for HSRP |
| Switch#Show int trunk | Shows what portchannel or interfaces are trunking |

To see the end result of this problem I had to be able to ping from PC1 to PC2. Through this we could swap out PAGP or LACP interchangeably with a simple change in commands both giving full connectivity. I used all the commands above to make sure my configuration was working and proper in state. All were used and the ping was successful after a few tweaks were made.

**Debug Syntax**

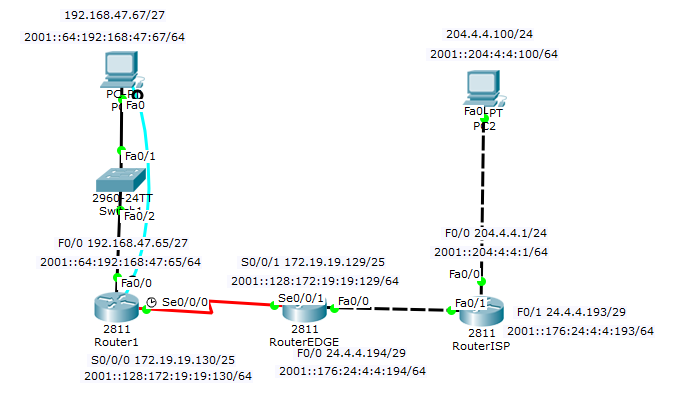
**Ping:**  ultimate goal here was to be able to ping to the other PC, we could have used either PAGP or LACP. To ping, it tests if we can or not and if not we can trouble shoot accordingly.

**Show etherchannel sum:** This command allowed me to see what the etherchannels had assigned and what state the devices in them were in. This allowed me to make changes to the interfaces in the etherchannels.

**Sh run:** Show run was used to make sure my switches were matching and the configuration was the same. This helps a lot with the fact if one of the VLANs is off, or IPs are different the connectivity could be broken.

1. IPv4 and IPv6 Static routing (NOTE: With AND Without the use of a default static route.)

The topic in this question is IPv4 and IPv6 Static routing. This is the method of having connectivity between computers and routers so different networks can be reached. These static routes make sure the routers can connect to networks that aren’t the ones they are in so a PC can essentially ping to a different network than its own. A default static route is used for any connection that doesn’t have a learned or static route it gets pushed to the default static route. To accomplish this, we needed 2 separate networks with a standard network containing a PC, switch, router and another network that contained a router in the middle as well as a router and PC. This enables us to have default static routes between the 2 routers and PC shown on the right of the topology and static routes to the router in the middle known as RouterEDGE. This topology enables us to use everything asked in the question above.



|  |  |
| --- | --- |
| IP Routing Commands | Explanation |
| Router(config)#Ipv6 unicast routing | Enables ipv6 addressing on the router at hand. |
| Router(config)#Ip route [dest. Network] [dest. SM] [next hop address] | Sets a static route using the syntax listed |
| Router(config)#Ipv6 route [dest address/SM] [Outgoing Int] | Sets ipv6 route using the syntax listed |
| Router(config)#Ip route 0.0.0.0 0.0.0.0 24.4.4.193 | Sets a default static route to address listed in last octet of syntax |
| Router(config)#Ipv6 route ::/0 [outgoing int or ip address] | Sets default static route to interface listed. |

|  |  |
| --- | --- |
| Ip Routing Show Commands | Explanation |
| Router#Show ip route | Shows current routes on the router |
| Router#Show ipv6 route | Shows current ipv6 routes on the router |

Using these show commands you are will see all the routes for the router you are configuring. On here you should see the S marker for a static route listing the network it is reaching with that static route. Additionally, you should see S\* for a default static. In the topology from above the Edge and ISP router both have default static routes. To check for functionality, I would ping address where my static routes were set to make sure I have full connectivity.

**Debug Syntax:**

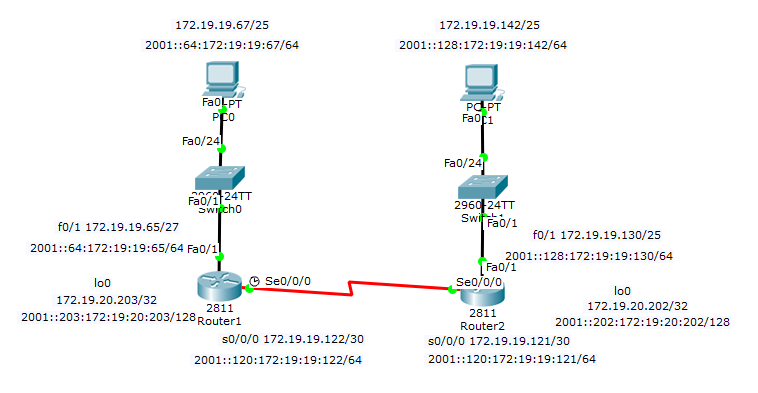
**Ping**: Pinging in this situation is very useful being static routes are mainly used for connectivity between networks. If my PC was not able to ping my Edge router then the static routes I configured was not correct. Using this command I was able to troubleshoot and see a mistake in my ip address I typed in when configuring routing.

**Tracert**: Tracert is a command that is used to show the flow of connectivity through a ping. It can show where a connection is being dropped which is very helpful when configuring static routing. Using this command a user can see where the connection goes bad and debug their static routes starting there.

In addition to these two, the two show commands were also used in debugging to see if there were typos in address or to see what was routed and what was not.

1. EIGRP and EIGRP for IPv6 (NOTE: Make sure you utilize VLSM in your topology and configuration without the use of automatic summarization.)

In this problem, we move away from static routing and into dynamic routing. I used 2 PCs for pinging, 2 routers to configure EIGRP on, and 2 switches in between my PCs and routers. EIGRP is a dynamic routing protocol meaning you just have to advertise the networks that you are connecting to you and the protocol connects you to other routers using the same protocol and autonomous number. The syntax and enabling for ipv6 is a little different but still very similar. To test the final product, a user must be able to ping from PC0 to PC1 using both IPv4 addresses and IPv6 addresses.



|  |  |
| --- | --- |
| EIGRP Commands | Description |
| Router(config)#ipv6 unicast-routing | Enables IPv6 addressing |
| Router(config)#router eigrp # | Enters eigrp routing protocol using a # |
| Router(config-router)#router-id [ip address] | Sets router ID for the eigrp protocol |
| Router(config-router)#network [ip address] [WC mask] | Sets network for routing protocol |
| Router(config-router)#no auto-summary | Turns off auto summarization |
| Router(config)#ipv6 router eigrp # | Enters ipv6 eigrp |
| Router(config-router)#no shutdown | Turns on ipv6 eigrp |
| Router(config)#int [interface] | Enters specified interface |
| Router(config-if)# ipv6 eigrp # | Enables eigrp 307 for IPv6 |

|  |  |
| --- | --- |
| EIGRP Show Commands | Explanation |
| Router#show ip route | Shows IP routing for ipv4 |
| Router#show ipv6 route | Shows IP routing for ipv6 |
| Router#show run | Shows current running config |

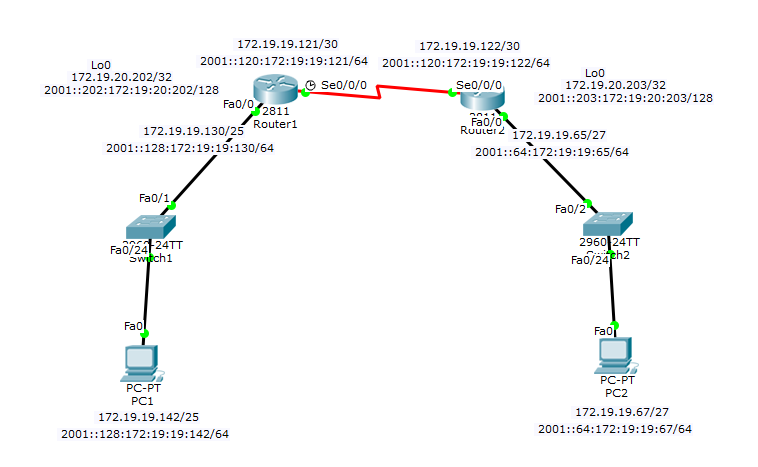
In this problem, the main show commands used was show Ip/ipv6 route. This shows the routing protocol and routes for the router. Using this we could determine what needed to be added or taken away. Additionally, show run was used to make sure both routers configurations were close to matching with some IPs changed. To finally test the functionality of EIGRP I pinged from PC0 to PC1 using both IPv4 and IPv6 addresses

**Debug Syntax:**

**Show ip/ipv6 route:** I use this command to troubleshoot what routes were missing and what routes were bad.

**Show Run:** Using show run I was able to look at the current config for EIGRP and use that information to remove networks that were wrong. This was very helpful in bringing up connectivity.

1. OSPFv2 and OSPFv3 (NOTE: Make sure you utilize loopback interfaces within your topology and configuration…and make sure you specify why you are utilizing them.)

In this problem, we are once again using another dynamic routing protocol, OSPF. OSPFv2 is used for IPv4 and OSPFv3 is used for IPv6. In this problem I used 2 PCs, 2 switches, and 2 routers. On the routers I implemented a loopback address. I used this for the router id in OSPF. The end goal of this problem is to be able to ping from PC1 to PC2 using the OSPF protocol.

|  |  |
| --- | --- |
| OSPF Commands | Description |
| Router(config)#ipv6 unicast-routing | Enables IPv6 addressing |
| Router(config)#router ospf # area # | Enters ospf routing protocol using a # and area # |
| Router(config-router)#router-id [ip address] | Sets router ID for the ospf protocol |
| Router(config-router)#network [ip address] [WC mask] area # | Sets network for routing protocol |
| Router(config)#ipv6 router ospf # area # | Enters ipv6 ospf |
| Router(config)#int [interface] | Enters specified interface |
| Router(config-if)# ipv6 ospf # area # | Enables ospf for IPv6 on selected interface |

|  |  |
| --- | --- |
| OSPF Show Commands | Explanation |
| Router#show ip route | Shows IP routing for ipv4 |
| Router#show ipv6 route | Shows IP routing for ipv6 |
| Router#show run | Shows current running config |

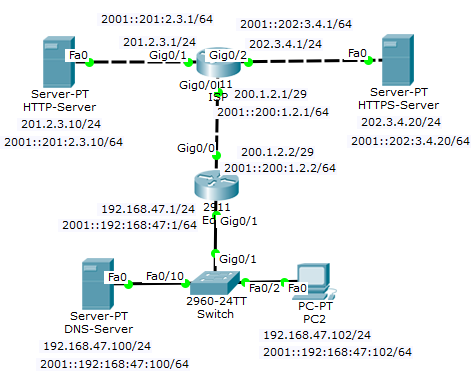
In this problem, the main show commands used was show Ip/ipv6 route. This shows the routing protocol and routes for the router. Using this we could determine what needed to be added or taken away. Additionally, show run was used to make sure both routers configurations were close to matching with some IPs changed. To finally test the functionality of EIGRP I pinged from PC0 to PC1 using both IPv4 and IPv6 addresses. This is very similar to EIGRP because they are both dynamic routing protocols.

**Debug Syntax:**

**Show ip/ipv6 route:** I use this command to troubleshoot what routes were missing and what routes were bad.

**Show Run:** Using show run I was able to look at the current config for OSPF and use that information to remove networks that were wrong. This was very helpful in bringing up connectivity.

1. Extended ACLs and Extended Named ACLs for IPv4 and IPv6 (NOTE: how, where and why you applying these…)

Extended ACLs and Extended Named ACLs are used for controlling traffic to and from servers. This is used in firewalls a lot because it can control bad traffic and cancel the request or let the request through if it is thorough. The technologies used in the example are servers, an Edge router, an ISP router, and a PC with a switch. The servers are used for sending and holding requests. For example, if a pc is trying to reach a webpage held on Server 2 it needs to go through server 1 and send out a request to access server 2 going through the Edge and ISP router. The switch makes it easy for a pc to be added to the scheme. The routers are where the ACLs are going to be stored, they are the ones that process and either deny or permit traffic through to the servers.

|  |  |
| --- | --- |
| ACL Commands | Description |
| Router(config)#ip access-list extended [name] | Creates an extended named ACL |
| Router(config)#permit tcp [source address] 0.0.0.0 [dest. Address] 0.0.0.0 | Adds a permit component to the access list being configured, which permits traffic from source to desination |
| Router(config)#deny ip any any | Denys any ip address that was not listed in the ACL |
| Router(config)#int [interface] | Enters interface specified |
| Router(config-if)#ip access-group [name/num] [in/out] | Applies an ACL to an interface facing either in or out. |
| Router(config)#ipv6 access-list [100-199] | Creates an Extended Numbered ACL for IPv6 |
| Router(config)#[permit/deny] [source ipv6 address] [dest. Ipv6 address] | Creates a statement in the ACL for an IPv6 address |
| Router(config)#int [interface] | Enters specified interface |
| Router(config-if)#no switchport | Changes from layer 2 mode to layer 3 mode. |
| Router(config-if)#ipv6 traffic-filter [ACL num/name] [in/out] | Applies the ipv6 access list to an interface facing in or out |

|  |  |
| --- | --- |
| ACL Show Commands | Explanation |
| Router#show access-lists | Shows all ACLs on router |
| Router#show ipv6 access-list | Shows all IPv6 ACLs on router |
| Router#show run | Shows current running config |

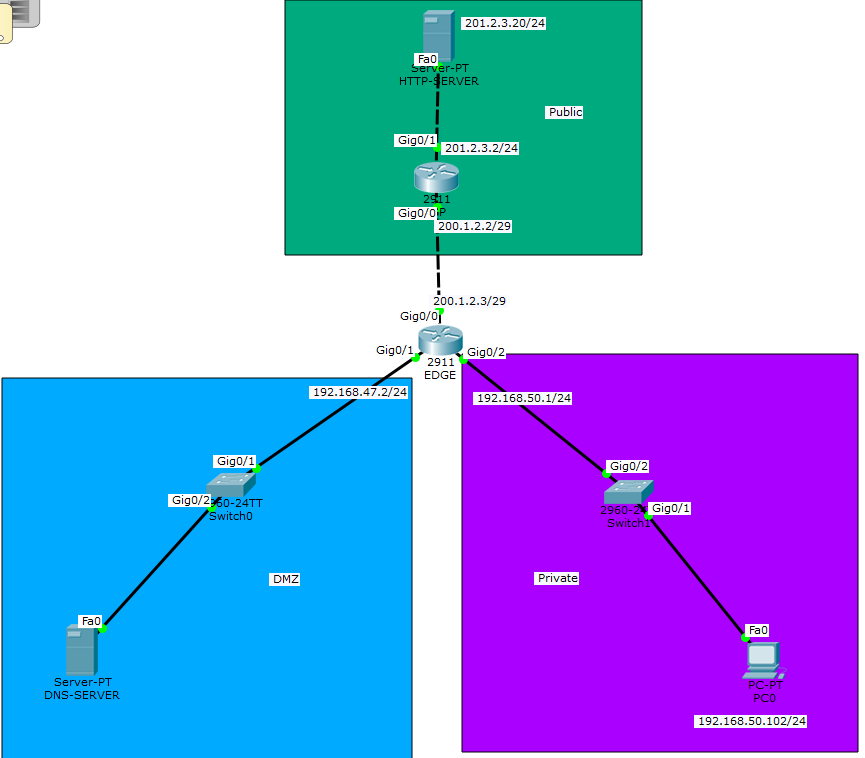
In this problem, the end goal was controlling traffic to a server. In my topology I configured it so the DNS Server could access the HTTP server but could not access the HTTPS server. In addition to that, I also configured that PC2 could access the HTTPS server but not the HTTP server. Using these 2 ACL configurations I was able to test the functionality by trying to access a webpage from an end device point towards one of the HTTP servers.

**Debug Syntax:**

**Show Run:** Show run was used to see the ACL and the addresses I put into those groups. This was very useful in debugging because I could see if I messed on accessing an address and be able to change the access list.

**Webpage:** I used the webpage function in trying to access the servers that I was supposed to be able to access and ones that I was not supposed to access. This was used as a final debugging step to see if I had to go back and change anything.

1. Static NAT and Dynamic NAT/PAT for IPv4 (NOTE: Make sure you show a configuration with static NAT for a web server in the DMZ, then PAT using a pool of multiple addresses for internal hosts connecting to the Internet. You should also demonstrate your knowledge of using just the interface IP Address when this is the only available address presented to you.)

In this problem we are asked to configure a static NAT and dynamic PAT for IPv4. To accomplish this, we had a DMZ network of a switch and server connected to an Edge router. We also had a private network which consisted of a switch and PC connected to the Edge router. Finally we had a public network with a router and server that was also connected to the edge router. For the DMZ we configured a static NAT without the knowledge of the end devices IP, which had us use the g0/1 of the edge router. This was configured as an inside network as well. For the PAT we configured a pool of address from the public network that would be accessible from the private network. We initialized this using PAT commands. With this configuration we were able to complete this problem. This is my favorite problem on the whole project, it made me understand the whole concept of PAT, NAT, pooling, and how the actual internet would work.

|  |  |
| --- | --- |
| NAT/PAT Commands | Description |
| Router(config)#ip access-list standard [name] | Creates an Access list for the PRIVATE NETWORK |
| Router(config)#permit tcp [source address] 0.0.0.0 [dest. Address] 0.0.0.0 | Adds a permit component to the access list being configured, which permits traffic from source to desination |
| Router(config)#deny any | Denys any ip address that was not listed in the ACL |
| Router(config)#int [interface] | Enters interface specified |
| Router(config-if)#ip access-group [name/num] [in/out] | Applies an ACL to an interface facing either in or out. |
| Router(config)#ip nat pool [name] [start address] [end address] netmask [SM] | Creates a pool of addresses for NAT |
| Router(config)#ip nat inside source list [name] pool [name] overload | Combines a pool and access list to then have an overload function to allow the address to have multiple addresses tied to it. |
| Router(config)#ip nat inside source static [source address] [dest. Address] | Static NAT command |
| Router(config)#int [interface] | Enters specified interface |
| Router(config-if)#ip nat [inside/outside] | Sets if the interface is an inside or outside interface. There can only be 1 outside. |

|  |  |
| --- | --- |
| NAT/PAT Show Commands | Explanation |
| Router#show ip nat translations | Shows nat translations on selected router |
| Router#show run | Shows current running config |

In this problem we are configuring a DMZ, private network, and public network. Using the commands above we are able to validate that the configurations went through and are functioning by being able to ping from end device to end device. In addition to this we should be able to see in nat translations the changes when we ping through the Edge router.

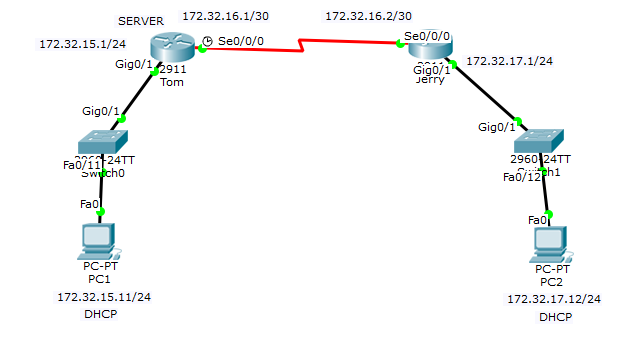
**Debug Syntax:**

**Show ip nat translations:** I used this debug command to check if my nat configurations went through and were functioning. To check this I looked if the local was filled or even if there was a table that comes when issued. This seemed a little buggy after I would ping through edge and there would be some syntax that was bugged. Other than that this command was very useful.

**Ping:** I would use pings to ping through the edge router to test my nat translations.

1. DHCP Server on the Router and a local DHCP client and a remote DHCP client (ie: using DHCP Relay.)

In this problem, we are faced with the problem of configuring DHCP servers and clients to that DHCP server. In the topology below we configured a DHCP server, local DHCP client, and remote DHCP client. I used 2 routers, 2 switches, 2 PCs. I used 1 router for the server which has a local connection to the DHCP local client. The other router and switch form a separate network that house a client as well. Since it is a different network it is going to use DHCP Relay to make it a remote client. Once finished, we had a server, a local DHCP client and remote DHCP client using DHCP Relay.



|  |  |
| --- | --- |
| DHCP Commands | Description |
| Tom(config)#ip dhcp excluded-address [ip address start] [ip address finish] | Sets the excluded addresses that DHCP can’t use to auto assign. |
| Tom(config)#ip dhcp pool [name] | Creates a DHCP pool |
| Tom(dhcp-config)# network [network address] [subnet mask] | Sets the network for the DHCP pool |
| Tom(dhcp-config)#default router [ip address] | Sets the default-router for the DHCP pool |
| Tom(config)#router ospf 1 | Initializes OSPF |
| Tom(config-router)#network [ip address] [WC mask] [area #] | Sets network to route to |
| Jerry(config)int g0/1 | Enters interface |
| Jerry(config-if)ip helper-address [ip address] | Routes all DHCP requests to address listed |

|  |  |
| --- | --- |
| DHCP Show Commands | Explanation |
| Router#show ip dhcp pool | Shows the DHCP pools configured on the router |
| Router#show ip dhcp binding | Shows the DHCP binding on the router |
| Ping [ip address] | Checks connectivity to address |

To show that the configuration worked I actually didn’t use any of the commands above first. I went into a PC and went into IP configuration and issued a DHCP request for the IP config. If the request was successful and set the IP address to the correct IP and Default gateway then I knew that it worked. To double check, I pinged from PC1 to PC2 using the DHCP configuration given and got connectivity.

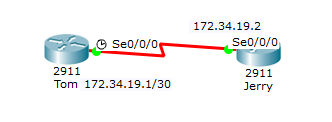
**Debug Syntax:**

**Show ip dhcp pool:** I used this command to check to see if my IP Pools were correctly configured.

**Show Run:** I used this a lot to make sure the excluded addresses for the config were correct and starting and ending at the address I wanted.

1. PPP using PPP CHAP

In this problem we are asked to use PPP chap which is an authentication protocol which uses a username based on the hostname and a standard password. To solve this problem I used 2 routers with a DCE connection and configured PPP using chap. Chap is significant because its based on a opposite hostname and standardized password.



|  |  |
| --- | --- |
| PPP Chap Commands | Description |
| Router(config)#username [username] password [cisco] | Sets a username and password for PPP |
| Router(config)#int s0/0/0 | Enters Interface listed |
| Router(config)#encapsulation ppp | Sets PPP on the interface |
| Router(config)#ppp authentication chap | Specifies for chap to be used in PPP |

|  |  |
| --- | --- |
| PPP Show Commands | Explanation |
| Router(config)#debug ppp authentication | Enables debugging for the current ppp authentication |
| Router(config)#debug ppp negotiation | Enables debugging of negotiation between the PPP process. |

Using these commands we were able to verify PPP Chap was working with the output shown in the CLI.

**Debug Syntax**

**Show Run:** Checked show run multiples to make sure hostnames and passwords were okay and to make sure the configurations set were on the right physical interface.

**Debug commands:** Using the 2 debug commands above we were able to see if our configurations were taken based on the output in the CLI