Task 1.1

SW1:

```
interface Port-channel12
 switchport trunk encapsulation isl
 switchport mode trunk
!
interface Port-channel13
 switchport trunk encapsulation isl
 switchport mode trunk
!
interface Port-channel14
 switchport trunk encapsulation isl
 switchport mode trunk
!
interface range Fa0/13 - 14
 switchport trunk encapsulation isl
 switchport mode trunk
 channel-group 12 mode on
 no shutdown
!
interface range Fa0/16 - 17
 switchport trunk encapsulation isl
 switchport mode trunk
 channel-group 13 mode on
 no shutdown
!
interface range Fa0/19 - 20
 switchport trunk encapsulation isl
 switchport mode trunk
 channel-group 14 mode on
 no shutdown
```

SW2:

```
interface Port-channel12
 switchport trunk encapsulation isl
 switchport mode trunk
!
interface range Fa0/13 - 14
 switchport trunk encapsulation isl
 switchport mode trunk
 channel-group 12 mode on
 no shutdown
```


SW3:
```
interface Port-channel13
 switchport trunk encapsulation isl
 switchport mode trunk
!
interface range Fa0/13 - 14
 switchport trunk encapsulation isl
 switchport mode trunk
 channel-group 13 mode on
 no shutdown
```

SW4:
```
interface Port-channel14
 switchport trunk encapsulation isl
 switchport mode trunk
!
interface range Fa0/13 - 14
 switchport trunk encapsulation isl
 switchport mode trunk
 channel-group 14 mode on
 no shutdown
```

Task 1.1 Breakdown

The first step in creating a layer 2 EtherChannel is to apply the **`channel-group`** command to the interface. As previously discussed, the *on* mode of the channel will disable the usage of both PAgP and LACP.

Next, configuration that should apply to both the channel interface and the member interfaces should be placed on the *port-channel* interface. In the above example the trunking configuration is shown on both the physical and logical interfaces for clarity. Options configured on the *port-channel* interface will be automatically inherited by the physical member interfaces.

The phase 'traffic sent over these trunk links should be tagged with a VLAN header' implies that ISL trunking must be used. By default only ISL tags all VLANs sent over the trunk link with a VLAN header (remember dot1q uses the native VLAN). However, in certain circumstances such as 802.1q tunneling, the native VLAN can carry a VLAN header by issuing the global command **`vlan dot1q tag native`**. This case will be covered in later labs.

Task 1.1 Verification

Verify that the port-channel is functional, for instance on SW2:

```
Rack1SW2#show etherchannel 12 port-channel
                Port-channels in the group:
                ---------------------------

Port-channel: Po12
------------

Age of the Port-channel   = 00d:00h:18m:19s
Logical slot/port   = 2/12          Number of ports = 2
GC                  = 0x00000000      HotStandBy port = null
Port state          = Port-channel Ag-Inuse
Protocol            =    -

Ports in the Port-channel:

Index   Load   Port     EC state        No of bits
------+------+------+------------------+-----------
  0     00     Fa0/13   On/FEC             0
  0     00     Fa0/14   On/FEC             0

Time since last port bundled:    00d:00h:18m:14s    Fa0/13
```

Verify the Etherchannel trunk encapsulation, for instance on SW2:

```
Rack1SW2#show interfaces port-channel 12 switchport
Name: Po12
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
Administrative Trunking Encapsulation: isl
Operational Trunking Encapsulation: isl
Negotiation of Trunking: On
Access Mode VLAN: 1 (default)
Trunking Native Mode VLAN: 1 (default)
Administrative Native VLAN tagging: enabled
<output omitted>
```

Verify all of the Etherchannel bundles from SW1:

```
Rack1SW1#show etherchannel summary | begin Group
Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------------
12     Po12(SU)         -        Fa0/13(P)   Fa0/14(P)
13     Po13(SU)         -        Fa0/16(P)   Fa0/17(P)
14     Po14(SU)         -        Fa0/19(P)   Fa0/20(P)
```


Task 1.2

SW2:

```
port-channel load-balance dst-mac
```

Task 1.2 Breakdown

By default, all traffic sent over an EtherChannel interface is load balanced based on the source MAC address of the frame. This can sometimes be a problem when a large amount of traffic is coming from the same source, such as a file server, media server, router, etc. If traffic monitoring shows that one interface inside a channel group is highly utilized and the others are not, this usually indicates the above case. To change the load balancing method to be based on the destination MAC address of the frame, use the global configuration command **`port-channel load-balance dst-mac`**.

For this task, traffic from the file server located behind BB2 will be sent across the trunk with the source MAC address of BB2's FastEthernet interface. By default, all of this traffic would use only one of the EtherChannel trunk links since the default is to load balance based on the source MAC address. With destination based load balancing enabled on SW2, this traffic will now be distributed across both links. Since traffic destined to BB2 will have the source MAC address of R1 or R6, this traffic will be load balanced based on the source MAC address and in turn load balanced.

Task 1.2 Verification

Verify the load balancing configuration:

```
Rack1SW2#show etherchannel load-balance
EtherChannel Load-Balancing Operational State (dst-mac):
Non-IP: Destination MAC address
  IPv4: Destination MAC address
  IPv6: Destination IP address
```


Task 1.3

SW2:

```
mac-address-table aging-time 10 vlan 8
mac-address-table aging-time 10 vlan 88
```

Task 1.3 Breakdown

The Content Addressable Memory (CAM) table is where the switch stores learned MAC addresses. This table is used as a "routing" table for the switch, and is used to determine the outgoing interface for a frame. When a unicast frame comes in that does not have an entry in the CAM table, it is treated as a broadcast frame. A broadcast frame is sent out all interfaces except the one it was received on. When the CAM table is full, all excess unicast frames are treated as broadcast frames. When this occurs, it is possible for traffic to leak between VLANs. The **mac-address-table aging-time** determines how long an idle MAC address can remain in the CAM table, and defaults to five minutes. In the above task this value is adjusted to flush inactive entries out of VLANs 8 and 88 after just 10 seconds.

Task 1.3 Verification

Verify the MAC address table aging time for VLANs 8 and 88:

```
Rack1SW2#show mac-address-table aging-time vlan 8
Vlan    Aging Time
----    ----------
   8      10

Rack1SW2#show mac-address-table aging-time vlan 88
Vlan    Aging Time
----    ----------
  88      10
```


Task 2.1

R2:
```
router ospf 1
 area 27 stub no-summary
 network 162.1.27.2 0.0.0.0 area 27
```

SW1:
```
ip routing
!
router ospf 1
 area 27 stub
 network 150.1.7.7 0.0.0.0 area 27
 network 162.1.27.7 0.0.0.0 area 27
```

Task 2.1 Breakdown

The above task dictates that SW1 does not meet specific reachability information about the rest of the network. As previously discussed, an LSA cannot be removed from the OSPF database on a per device basis. Instead, this must be accomplished by defining a stub area.

Since the above task also states that SW2 should only see a default route as generated by R2, it is evident that external or inter-area routing information should not be allowed into OSPF area 27. This requirement immediately eliminates the stub and not-so-stubby area types, as these two do allow inter-area reachability information to enter the area. Therefore, the only two options remaining are a totally stubby area or a not-so-totally-stubby area. As there is no redistribution occurring into OSPF area 27, the totally-stubby area has been chosen in the above sample solution. However as there is no restriction to eliminate a not-so-totally-stubby area, that would also be a valid solution.

Task 2.1 Verification

Verify that area 27 is a stub area:

```
Rack1R2#show ip ospf | begin Area 27
    Area 27
        Number of interfaces in this area is 1
        It is a stub area, no summary LSA in this area
          generates stub default route with cost 1
```

Verify the routing table on SW1:

```
Rack1SW1#show ip route ospf
O*IA 0.0.0.0/0 [110/2] via 162.1.27.2, 00:01:34, Vlan27
```

Task 2.2

R1:

```
router eigrp 200
 metric weights 0 3 1 1 0 0
```

R3:

```
router eigrp 200
 metric weights 0 3 1 1 0 0
```

SW2:

```
router eigrp 200
 metric weights 0 3 1 1 0 0
```


Task 2.2 Breakdown

EIGRP metric calculation uses a composite of four values, bandwidth, delay, load, and reliability. By default EIGRP only uses bandwidth and delay in its metric calculation; however this behavior can be modified by changing the **metric weights** under the EIGRP process.

☠ Pitfall

Metric weighting must match between devices in the EIGRP domain in order to establish adjacency. Therefore if you modify the metric weights parameter on one EIGRP device you must do so on all EIGRP devices in that autonomous system.

The EIGRP metric calculation formula is as follows:

```
(k1 * bandwidth + (k2 * bandwidth)/(256 - load) + k3 * delay) *
(k5/(reliability + k4))
```

The latter half of the calculation is only performed if k5 does not equal 0. The variable definitions of the above formula are as follows:

Bandwidth: The inverse of the lowest bandwidth along the path for the prefix times 2.56 x 10^{12} in bits per second.

Delay: The cumulative interface delay along the entire path of the prefix in tens of microseconds.

Reliability: Reliability of local interface as a fraction of 255.

Load: Load of local interface as a fraction of 255.

The above values can be determined for a prefix as follows:

```
Rack1R6#show ip route eigrp
D    200.0.0.0/24 [90/146432] via 54.1.1.254, 00:00:09, Serial0/0/0
D    200.0.1.0/24 [90/146432] via 54.1.1.254, 00:00:09, Serial0/0/0
D    200.0.2.0/24 [90/146432] via 54.1.1.254, 00:00:09, Serial0/0/0
D    200.0.3.0/24 [90/146432] via 54.1.1.254, 00:00:09, Serial0/0/0
```



```
Rack1R6#show ip route 200.0.0.0
Routing entry for 200.0.0.0/24
  Known via "eigrp 10", distance 90, metric 2297856, type internal
  Redistributing via eigrp 10
  Last update from 54.1.1.254 on Serial0/0/0, 00:00:19 ago
  Routing Descriptor Blocks:
  * 54.1.1.254, from 54.1.1.254, 00:00:19 ago, via Serial0/0/0
     Route metric is 2297856, traffic share count is 1
     Total delay is 25000 microseconds, minimum bandwidth is 1544 Kbit
     Reliability 255/255, minimum MTU 1500 bytes
     Loading 1/255, Hops 1
```

Task 2.2 Verification

Verify the EIGRP metric weights:

```
Rack1SW2#show ip protocols | beg eigrp
Routing Protocol is "eigrp 200"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=3, K2=1, K3=1, K4=0, K5=0
<output omitted>
```

Task 2.3

R4:

```
ip route 150.1.5.5 255.255.255.255 162.1.45.5 111
ip route 162.1.5.0 255.255.255.0 162.1.45.5 111
ip route 162.1.55.0 255.255.255.0 162.1.45.5 111
!
router ospf 1
 redistribute static subnets
```

R5:

```
ip route 0.0.0.0 0.0.0.0 162.1.45.4 111
```

Task 2.3 Breakdown

Static default routes are a very simple and effective way to replace more specific routing information when it is lost. A default route that is only installed in the IP routing table when another route (either dynamically learned or statically configured) is lost is called a *floating static* route.

A floating static route is a route with the same longest match as another route in the IP routing table, but which has a higher administrative distance. Therefore, the floating route will not get installed unless the primary route with the lower administrative distance leaves the IP routing table.

In the above scenario, R5 is learning a default route from R3 via OSPF. OSPF routes have an administrative distance of 110. There has also been a static default route configured on R5 pointing to R4 over the serial link with an

administrative distance on 111. Unless the route with the lower administrative distance (the OSPF route with AD of 110) leaves the IP routing table, the static route will not get installed. This case will occur when R5 loses its connection to the Frame Relay cloud. Therefore, the above configured default route is a simple yet effective backup solution for R5.

In order to maintain full reachability back to R5, R4 has configured three static routes pointing to the directly connected networks of R4. As these routes have an administrative distance of 111 (higher than OSPF), they will not be installed in the routing table unless R4 loses the route through OSPF. Note that these routes must be redistributed into the OSPF domain to ensure that all other devices have a route when the Frame Relay circuit of R5 is down.

Task 2.3 Verification

Shutdown interface Serial0/0/0 on R5 and verify routing table:

```
Rack1R5#show ip route  | begin Gate
Gateway of last resort is 162.1.45.4 to network 0.0.0.0

     162.1.0.0/16 is variably subnetted, 4 subnets, 2 masks
C       162.1.45.4/32 is directly connected, Serial0/1
C       162.1.45.0/24 is directly connected, Serial0/1
C       162.1.55.0/24 is directly connected, FastEthernet0/1
C       162.1.5.0/24 is directly connected, FastEthernet0/0
     150.1.0.0/24 is subnetted, 1 subnets
C       150.1.5.0 is directly connected, Loopback0
S*   0.0.0.0/0 [111/0] via 162.1.45.4
```

Verify connectivity to R5's networks:

```
Rack1R3#ping 150.1.5.5

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.5.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 88/89/92 ms

Rack1R3#ping 162.1.55.5

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 162.1.55.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 88/89/92 ms
```


Task 2.4

R1:

```
key chain RIP
 key 1
  key-string CISCO
!
interface FastEthernet0/0
 ip rip authentication mode md5
 ip rip authentication key-chain RIP
!
router rip
 version 2
 passive-interface default
 network 192.10.1.0
 neighbor 192.10.1.254
 neighbor 192.10.1.6
 no auto-summary
```

R4:

```
router rip
 version 2
 network 204.12.1.0
 no auto-summary
```

R6:

```
key chain RIP
 key 1
  key-string CISCO
!
interface FastEthernet0/0
 ip rip authentication mode md5
 ip rip authentication key-chain RIP
!
router rip
 version 2
 passive-interface default
 no passive-interface Serial0/0/0.1
 network 54.0.0.0
 network 150.1.0.0
 network 192.10.1.0
 network 162.1.0.0
 neighbor 192.10.1.1
 neighbor 192.10.1.254
 no auto-summary
```


Task 2.4 Verification

Verify the RIP configuration on the RIP enabled routers:

```
Rack1R1#show ip protocols | beg rip
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 0 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: rip
  Neighbor(s):
    192.10.1.6
    192.10.1.254
  Default version control: send version 2, receive version 2
  Automatic network summarization is not in effect
  Maximum path: 4
  Routing for Networks:
    192.10.1.0
  Passive Interface(s):
    VoIP-Null0
    FastEthernet0/0
    Serial0/0/0
    Serial0/1
    Virtual-Access1
    Loopback0
  Routing Information Sources:
    Gateway         Distance      Last Update
    192.10.1.254         120      00:00:24
    192.10.1.6           120      00:00:18
  Distance: (default is 120)
```

Make sure RIP updates are authenticated:

```
Rack1R1#debug ip rip
RIP: received packet with MD5 authentication
RIP: received v2 update from 192.10.1.254 on FastEthernet0/0
     205.90.31.0/24 via 0.0.0.0 in 7 hops
     220.20.3.0/24 via 0.0.0.0 in 7 hops
     222.22.2.0/24 via 0.0.0.0 in 7 hops
     RIP: received packet with MD5 authentication
RIP: received v2 update from 192.10.1.6 on FastEthernet0/0
     54.1.1.0/24 via 0.0.0.0 in 1 hops
     150.1.6.0/24 via 0.0.0.0 in 1 hops
     162.1.6.0/24 via 0.0.0.0 in 1 hops
     212.18.0.0/24 via 0.0.0.0 in 2 hops
     212.18.1.0/24 via 0.0.0.0 in 2 hops
     212.18.2.0/24 via 0.0.0.0 in 2 hops
     212.18.3.0/24 via 0.0.0.0 in 2 hops
Rack1R1#undebug all
```


Finally verify that updates are being sent as unicast (note that we still receive multicast from BB2 and suppress null updates at R1):

```
Rack1R1(config)#access-list 100 permit udp any eq 520 any eq 520
Rack1R1#debug ip packet detail 100
Rack1R1#debug ip rip events

IP: s=192.10.1.254 (FastEthernet0/0), d=224.0.0.9, len 132, rcvd 2
      UDP src=520, dst=520
RIP: received v2 update from 192.10.1.254 on FastEthernet0/0
RIP: Update contains 3 routes
IP: tableid=0, s=192.10.1.6 (FastEthernet0/0), d=192.10.1.1
(FastEthernet0/0), routed via RIB
IP: s=192.10.1.6 (FastEthernet0/0), d=192.10.1.1 (FastEthernet0/0), len
212, rcvd 3 UDP src=520, dst=520
RIP: received v2 update from 192.10.1.6 on FastEthernet0/0
RIP: Update contains 7 routes
RIP: sending v2 update to 192.10.1.6 via FastEthernet0/0 (192.10.1.1) -
suppressing null update
RIP: sending v2 update to 192.10.1.254 via FastEthernet0/0 (192.10.1.1)
- suppressing null update
```

Task 2.5

R1:

```
router eigrp 200
 redistribute rip metric 10000 1000 100 1 1500
!
router rip
 version 2
 redistribute eigrp 200 metric 1
```

R3:

```
router eigrp 200
 redistribute ospf 1 metric 10000 1000 100 1 1500
!
router ospf 1
 redistribute eigrp 200 subnets route-map EIGRP_TO_OSPF
!
ip prefix-list VLAN162 seq 5 permit 192.10.1.0/24
!
route-map EIGRP_TO_OSPF permit 10
 match ip address prefix-list VLAN162
```



```
R4:
interface FastEthernet0/0
 ip summary-address rip 162.1.0.0 255.255.0.0
 ip summary-address rip 150.1.0.0 255.255.240.0
!
router ospf 1
 redistribute connected subnets route-map CONNECTED->OSPF
 redistribute rip subnets
 summary-address 30.0.0.0 255.252.0.0
 summary-address 31.0.0.0 255.252.0.0
!
router rip
 version 2
 redistribute ospf 1 metric 1
!
route-map CONNECTED->OSPF permit 10
 match interface Serial0/1 FastEthernet0/0
```

Task 2.5 Verification

Make sure OSPF domain sees only the summaries of BB3's prefixes:

```
Rack1R3#show ip route ospf  | inc ( 30\.| 31\.)
     31.0.0.0/14 is subnetted, 1 subnets
O E2    31.0.0.0 [110/20] via 162.1.0.4, 00:05:13, Serial1/0
     30.0.0.0/14 is subnetted, 1 subnets
O E2    30.0.0.0 [110/20] via 162.1.0.4, 00:05:13, Serial1/0
```

Make sure R4 announces the summary for the internal address space:

```
Rack1R4#debug ip rip

RIP: sending v2 update to 224.0.0.9 via FastEthernet0/0 (204.12.1.4)
RIP: build update entries
      0.0.0.0/0 via 0.0.0.0, metric 1, tag 0
      30.0.0.0/14 via 0.0.0.0, metric 1, tag 0
      31.0.0.0/14 via 0.0.0.0, metric 1, tag 0
      150.1.0.0/20 via 0.0.0.0, metric 2, tag 0
      162.1.0.0/16 via 0.0.0.0, metric 2, tag 0
      192.10.1.0/24 via 0.0.0.0, metric 1, tag 0
```

Note that RIP on R4 sends back to BB3 a default route (originated in OSPF) and summary prefixes.

Verify full internal connectivity using the TCL script below script:

```
foreach i {
150.1.1.1
162.1.13.1
192.10.1.1
150.1.2.2
162.1.0.2
162.1.27.2
162.1.38.3
150.1.3.3
162.1.0.3
```



```
162.1.3.3
162.1.13.3
162.1.45.4
150.1.4.4
162.1.0.4
204.12.1.4
162.1.45.5
162.1.55.5
150.1.5.5
162.1.5.5
162.1.0.5
54.1.1.6
150.1.6.6
162.1.6.6
192.10.1.6
150.1.7.7
162.1.27.7
162.1.88.8
162.1.38.8
150.1.8.8
162.1.8.8
162.1.88.8
} {puts [ exec "ping $i" ] }
```

Note that the above script does not include IP addresses of the links connecting SW2, SW3 and SW4 as those are not advertised into any IGP and are used for BGP peering.

Finally verify reachability to the backbone IGP networks using the TCL script below:

```
foreach i {
204.12.1.254
192.10.1.254
54.1.1.254
31.3.0.1
31.2.0.1
31.1.0.1
31.0.0.1
30.2.0.1
30.3.0.1
30.0.0.1
30.1.0.1
212.18.1.1
212.18.0.1
212.18.3.1
212.18.2.1
222.22.2.1
220.20.3.1
205.90.31.1
} {puts [ exec "ping $i" ] }
```

Task 2.6

R3:

```
router bgp 300
 neighbor 162.1.0.4 remove-private-AS
```



```
 neighbor 162.1.13.1 remove-private-AS
```

SW1:

```
interface Loopback1
 ip address 162.1.7.7 255.255.255.0
!
router bgp 65001
 network 162.1.7.0 mask 255.255.255.0
```

SW2:

```
interface Loopback1
 ip address 162.1.18.8 255.255.255.0
!
router bgp 65002
 network 162.1.18.0 mask 255.255.255.0
```

Task 2.6 Breakdown

Applying for a public BGP AS number requires the justification of the need for the AS number. For networks that do not have their own block of address space, this may not be possible. For this reason, the top 1024 addresses in the BGP AS range are marked as private.

Suppose that your network has multiple connections to the same Internet Service Provider. Due to complex routing policy, you want to run BGP with this upstream provider. However, as this provider is your only connection to the Internet, you are using their address space, and do not have your own provider independent block. In the case the provider can assign you a locally significant AS number in the range of 64512 – 65535. However, as these AS numbers are not valid on the Internet, they must be removed from the AS-Path of routes you are originating when the provider passes them upstream.

This is accomplished by adding the **`remove-private-as`** keyword on to the neighbor statement of the upstream connection. In order for the private AS number to be removed, it must be the only AS in the path. In other words, the private AS must be directly connected to the AS that is trying to remove it.

Task 2.6 Verification

```
Rack1R4#show ip bgp | include 150.1.9.0
 *> 150.1.9.0/24     162.1.0.3                              0 300 i
```

Task 2.7

R5:

```
interface Loopback2
 ip address 162.1.15.5 255.255.255.0
!
router bgp 500
 network 162.1.15.0 mask 255.255.255.0
 neighbor 150.1.4.4 send-community
 neighbor 150.1.4.4 route-map NO_ADVERTISE out
!
ip as-path access-list 1 permit ^$
!
route-map NO_ADVERTISE permit 10
 match as-path 1
 set community no-advertise
route-map NO_ADVERTISE permit 1000
```

Task 2.7 Breakdown

By setting the well known community attributes of no-export, no-advertise, or local-as, how a route is processed by an upstream neighbor can be controlled downstream. In the above task, it asks that network 162.1.15.0/24 be advertised into BGP on R5. This network then gets passed on to R4 via BGP. The requirement also states that R4 should not pass this on. By setting the community value to one of the aforementioned, R4 will not advertise the route on.

Recall the well-known communities:

Well Known Community	Behavior
Internet	All routes belong to this community by default. The Internet community has no special behavior.
No-advertise	Do not advertise to *any* BGP neighbor.
No-export	Do not advertise to any *EBGP* neighbor.
Local-AS	Do not advertise outside of sub-AS. This is a special case of no-export for use inside of a confederation.

Task 2.7 Verification

Verify that R5 actually sends communities to R4:

```
Rack1R5#show ip bgp neighbors 150.1.4.4 | include Comm
  Community attribute sent to this neighbor
```

Verify that R4 does not advertise R5 prefix to any peers:

```
Rack1R4#show ip bgp 162.1.15.0
BGP routing table entry for 162.1.15.0/24, version 18
Paths: (1 available, best #1, table Default-IP-Routing-Table, not
advertised to any peer)
Flag: 0x880
  Not advertised to any peer
  500
    150.1.5.5 (metric 65) from 150.1.5.5 (150.1.5.5)
      Origin IGP, metric 0, localpref 100, valid, external, best
      Community: no-advertise
```

Task 2.8

R4:

```
router bgp 100
 bgp dampening route-map DAMPENING
!
route-map DAMPENING permit 10
 set dampening 15 1000 3000 30
```

Task 2.8 Breakdown

BGP route flap dampening (damping) is the process of suppressing consistently unstable routes from being used or advertised to BGP neighbors. Dampening is (and must be) used to minimize the amount of route recalculation performed in the global BGP table as a whole.

Command Syntax:
bgp dampening [half-life reuse suppress max-suppress-time]

To understand dampening, the following terms must first be defined:

Penalty: Every time a route flaps, a penalty value of 1000 is added to the current penalty. All prefixes start with a penalty of zero.

Half-life: Configurable time it takes the penalty value to reduce by half. Defaults to 15 minutes.

Suppress Limit: Threshold at which a route is suppressed if the penalty exceeds. Defaults to 2000.

Reuse Limit: Threshold at which a suppressed route is unsuppressed if the penalty drops below. Defaults to 750.

Max Suppress: Maximum time a route can be suppressed if it has been stable. Defaults to four times the half-life value.

Each time a route flaps (leaves the BGP table and reappears), it is assigned a penalty of 1000. As soon as this occurs, the penalty of the route starts to decay based on the half-life timer. As the penalty increases, so does the rate of decay. For example, after a single flap, it will take 15 minutes for a prefix to reduce its penalty to 500.

Once the penalty of a prefix exceeds the suppress limit, the prefix is suppressed. A suppressed prefix cannot be used locally or advertised to any BGP peer. Once the penalty decay has resulted in the penalty decreasing below the reuse limit, the prefix is unsuppressed.

Lastly, the max-suppress timer dictates the maximum amount of time a prefix can be suppressed if it has been stable. This value is useful if a number of flaps have occurred in a short period of time, after which the route has been stable.

To enable BGP route flap dampening, simply enter the command **bgp dampening** under the BGP process. It can also be configured with a route-map as shown here, which allows the potential for more granularity, rather than applying to all networks.

Here, we are given the values for reuse, suppress, and max-suppress time. We are not given a value for half life, but it needs to be less than the default of 30 minutes. It needs to be less than 19. Here, we have chosen 15. For the math behind this, see the command reference for the equation:

Max penalty = reuse-limit *2^(maximum suppress time/half time)

Task 2.8 Verification

Verify the dampening parameters:

```
Rack1R4#show ip bgp dampening parameters
 dampening 15 1000 3000 30 (route-map DAMPENING 10)
  Half-life time      : 15 mins       Decay Time       : 370 secs
  Max suppress penalty:  4000         Max suppress time: 30 mins
  Suppress penalty    :  3000         Reuse penalty    : 1000
```

To verify how dampening works, first issue "debug ip bgp updates" on R5. Next do shutdown,and no shutdown four times or more at interface Loopback2. Keep a little time between shutdown/no shutdown long enough, for BGP to send update (watch debug output for control).

```
Rack1R5#conf t
```



```
Rack1R5(config)#interface lo2
Rack1R5(config-if)#shutdown
BGP(0): route 162.1.15.0/24 down
BGP(0): no valid path for 162.1.15.0/24
BGP(0): nettable_walker 162.1.15.0/24 no best path
BGP(0): 150.1.4.4 send unreachable 162.1.15.0/24
BGP(0): 150.1.4.4 send UPDATE 162.1.15.0/24 – unreachable

Rack1R5(config-if)#no shutdown
BGP(0): route 162.1.15.0/24 up
BGP(0): route 162.1.15.0/24 up
BGP(0): nettable_walker 162.1.15.0/24 route sourced locally
BGP(0): 150.1.4.4 send UPDATE (format) 162.1.15.0/24, next 150.1.5.5,
metric 0, path
```

Next look on R4 for any dampened paths:

```
Rack1R4#show ip bgp dampening dampened-paths
BGP table version is 25, local router ID is 150.1.4.4
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          From             Reuse    Path
*d 162.1.15.0/24    150.1.5.5        00:05:49 500 i
```

Task 3.1

R1:

```
interface Serial0/0
 ipv6 address 2001:CC1E:1::1/64
 ipv6 address FE80::1 link-local
 frame-relay map ipv6 2001:CC1E:1::3 113 broadcast
 frame-relay map ipv6 FE80::3 113
```

R2:

```
interface Serial0/0.1 point-to-point
 ipv6 address FEC0:234::2/64
 ipv6 address FE80::2 link-local
```

R3:

```
ipv6 unicast-routing

interface Serial1/0
 ipv6 address FEC0:234::3/64
 ipv6 address FE80::3 link-local
 frame-relay map ipv6 FEC0:234::2 302 broadcast
 frame-relay map ipv6 FEC0:234::4 304 broadcast
 frame-relay map ipv6 FE80::2 302
 frame-relay map ipv6 FE80::4 304
!
interface Serial1/1
 ipv6 address 2001:CC1E:1::3/64
 frame-relay map ipv6 2001:CC1E:1::1 311 broadcast
 ipv6 address FE80::3 link-local
```



```
 frame-relay map ipv6 FE80::1 311

R4:
interface Serial0/0/0
 ipv6 address FEC0:234::4/64
 ipv6 address FE80::4 link-local
 frame-relay map ipv6 FEC0:234::2 403
 frame-relay map ipv6 FEC0:234::3 403 broadcast
 frame-relay map ipv6 FE80::2 403
 frame-relay map ipv6 FE80::3 403
```

Task 3.1 Breakdown

Frame Relay is a non-broadcast multi-access (NBMA) media. This implies that layer 3 to layer 2 resolution is required for multipoint configurations. As of current Cisco IOS releases, Inverse Neighbor Discovery is not supported. Therefore, static layer 3 to layer 2 resolution must be configured with the **`frame-relay map ipv6`** statement. The host portion of the configured site-local networks can be determined by issuing either the **`show ipv6 interface`** command or the **`show ipv6 interface brief`** command.

```
Rack1R1#show ipv6 interface s0/0
Serial0/0 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::2D0:58FF:FE6E:B720
  Global unicast address(es):
    FEC0::13:2D0:58FF:FE6E:B720, subnet is FEC0:0:0:13::/64
<output omitted>

Rack1R3#show frame-relay map
Serial1/1 (up): ipv6 FEC0::13:2D0:58FF:FE6E:B720 dlci 311
(0x137,0x4C70), static,
              broadcast,
              CISCO, status defined, active
<output omitted>
```

Task 3.1 Verification

Note that link-local IPv6 addresses are configured and mapped explicitly. This is needed to configure IPv6 BGP later.

```
Rack1R4#show frame-relay map
<output omitted>
Serial0/0/0 (up): ipv6 FE80::2 dlci 403(0x193,0x6430), static,
              CISCO, status defined, active
Serial0/0/0 (up): ipv6 FE80::3 dlci 403(0x193,0x6430), static,
              CISCO, status defined, active
...
Serial0/0/0 (up): ipv6 FEC0:234::2 dlci 403(0x193,0x6430), static,
              CISCO, status defined, active
Serial0/0/0 (up): ipv6 FEC0:234::3 dlci 403(0x193,0x6430), static,
              broadcast,
              CISCO, status defined, active
<output omitted>
```



```
Rack1R4#ping fec0:234::2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0:234::2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 136/139/140
ms

Rack1R4#ping fec0:234::3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0:234::3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 60/60/60 ms
```

Note that you need to enable ipv6 unicast routing on R3 in order to ping spoke from spoke.

Task 3.2

R1:

```
ipv6 unicast-routing
!
router bgp 200
 neighbor 2001:CC1E:1::3 remote-as 300
!
 address-family ipv6
 neighbor 2001:CC1E:1::3 activate
```

R2:

```
ipv6 unicast-routing
!
router bgp 300
 neighbor FEC0:234::3 remote-as 300
!
 address-family ipv6
 neighbor FEC0:234::3 activate
```

R3:

```
ipv6 unicast-routing
!
router bgp 300
 neighbor 2001:CC1E:1::1 remote-as 200
 neighbor FEC0:234::2 remote-as 300
 neighbor FEC0:234::4 remote-as 100
!
 address-family ipv6
  neighbor 2001:CC1E:1::1 activate
  neighbor FEC0:234::2 activate
  neighbor FEC0:234::4 activate
```

R4:

```
ipv6 unicast-routing
!
router bgp 100
 neighbor FEC0:234::3 remote-as 300
```



```
!
 address-family ipv6
  neighbor FEC0:234::3 activate
```

Task 3.2 Breakdown

The above configuration dictates how to configure Multiprotocol BGP for IPv6. The first step is to issue the BGP **neighbor** command, followed by the destination peer's IPv6 address and AS number. Next, enable IPv6 unicast processing for the neighbor by issuing the **address-family ipv6** command under the BGP process, followed by the **neighbor [ipv6 address] activate** command.

```
Rack1R3#show bgp ipv6 summary
BGP router identifier 162.1.13.3, local AS number 300
BGP table version is 2, main routing table version 2
1 network entries using 133 bytes of memory
1 path entries using 72 bytes of memory
1 BGP path attribute entries using 60 bytes of memory
1 BGP AS-PATH entries using 24 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
BGP using 289 total bytes of memory
BGP activity 1/0 prefixes, 1/0 paths, scan interval 60 secs

Neighbor  V  AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down State/PfxRcd
FEC0::13:2D0:58FF:FE6E:B720
          4  200     24      23        2    0    0 00:19:52        1
```

To check the status of the MBGP peering, issue the **show bgp ipv6 summary** command. In the above output, R3 has formed a MBGP peering relationship with R1 using the destination address FEC0::13:2D0:58FF:FE6E:B720, and is learning one IPv6 prefix.

```
Rack1R3#show bgp ipv6
BGP table version is 2, local router ID is 162.1.13.3
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 2001:CC1E:1:1::/64
                    FEC0::13:2D0:58FF:FE6E:B720
                                             0             0 200 i
```

Note that in the above output, the prefix 2001:CC1E:1:1:/64 has been learned via BGP per the **show bgp ipv6** output, and has an associated next-hop value of FEC0::13:2D0:58FF:FE6E:B720.


```
Rack1R3#show ipv6 route FEC0::13:2D0:58FF:FE6E:B720
IPv6 Routing Table - 5 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
    O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
C   FEC0:0:0:13::/64 [0/0]
     via ::, Serial1/1
```

When a recursive lookup is performed on this next-hop value, per the above **show ipv6 route FEC0::13:2D0:58FF:FE6E:B720** output, the outgoing interface is seen to be Serial1/1, which is a multipoint interface running Frame Relay. However, when traffic is encapsulated on the interface the layer 2 address is resolved per the link-local address of the next-hop interface, not the global unicast address. This can be seen by the below **show ipv6 route bgp** output.

```
Rack1R3#show ipv6 route bgp
IPv6 Routing Table - 5 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
    O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
B   2001:CC1E:1:1::/64 [20/0]
     via FE80::2D0:58FF:FE6E:B720, Serial1/1
```

This implies that layer 3 to layer 2 resolution via the **frame-relay map ipv6** command must be configured for the remote link-local address FE80::2D0:58FF:FE6E:B720.

```
Rack1R1#show ipv6 int brief
Ethernet0/0                [up/up]
    FE80::230:19FF:FE69:81A0
    2001:CC1E:1:1:230:19FF:FE69:81A0
Serial0/0                  [up/up]
    FE80::2D0:58FF:FE6E:B720
    FEC0::13:2D0:58FF:FE6E:B720
<output omitted>
```

Quick Note
Global unicast address

Quick Note
Link-local address

```
Rack1R3#debug ipv6 packet
IPv6 unicast packet debugging is on
Rack1R3#debug frame-relay packet
Frame Relay packet debugging is on
Rack1R3#ping
Protocol [ip]: ipv6
Target IPv6 address: 2001:CC1E:1:1:230:19FF:FE69:81A0
Repeat count [5]: 1
Datagram size [100]:
Timeout in seconds [2]:
Extended commands? [no]:
Type escape sequence to abort.
Sending 1, 100-byte ICMP Echos to 2001:CC1E:1:1:230:19FF:FE69:81A0,
timeout is 2 seconds:
```



```
IPv6: SAS picked source FEC0::13:201:96FF:FE63:96C0 for
2001:CC1E:1:1:230:19FF:FE69:81A0
IPv6: nexthop FE80::2D0:58FF:FE6E:B720,
IPV6: source FEC0::13:201:96FF:FE63:96C0 (local)
      dest 2001:CC1E:1:1:230:19FF:FE69:81A0 (Serial1/1)
      traffic class 0, flow 0x0, len 100+0, prot 58, hops 64,
originating
Serial1/1:Encaps failed--no map entry link 79(IPV6)
IPv6: Encapsulation failed.
Success rate is 0 percent (0/1)
```

☞ Quick Note
Encapsulation fails because R3 does not have a layer 3 to layer 2 mapping for the next-hop address FE80::2D0:58FF:FE6E:B720s

```
Rack1R3#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1R3(config)#interface s1/1
Rack1R3(config-if)#frame-relay map ipv6 FE80::2D0:58FF:FE6E:B720 311
Rack1R3(config-if)#do ping 2001:CC1E:1:1:230:19FF:FE69:81A0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:CC1E:1:1:230:19FF:FE69:81A0,
timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/17/20 ms
```

Task 3.2 Verification

Verify IPv6 BGP peering status:

```
Rack1R3#show bgp ipv6 unicast summary  | begin Neighbor
Neighbor    V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down State/PfxRcd
2001:CC1E:1::1
            4 200   5       6        7    0    0 00:01:19        0
FEC0:234::2 4 300  14      15        7    0    0 00:09:31        0
FEC0:234::4 4 100  15      13        7    0    0 00:03:22        6
```


Task 3.3

R1:

```
router bgp 200
 !
 address-family ipv6
 network 2001:CC1E:1:1::/64
```

R2:

```
router bgp 300
!
 address-family ipv6
 network 2001:CC1E:1:2::/64
```

R3:

```
router bgp 300
 !
 address-family ipv6
 network 2001:CC1E:1::/64
 network 2001:CC1E:1:3::/64
 network FEC0:234::/64
```

R4:

```
router bgp 100
 !
 address-family ipv6
  network 2001:204:12:1::/64
```

Task 3.3 Verification

Verify that the prefixes are advertised:

```
Rack1R4#show bgp ipv6 unicast
BGP table version is 13, local router ID is 150.1.4.4
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
<output omitted>
*> 2001:CC1E:1::/64 FEC0:234::3              0             0 300 i
*> 2001:CC1E:1:1::/64
                    FEC0:234::3                            0 300 200 i
*> 2001:CC1E:1:2::/64
                    FEC0:234::3                            0 300 i
*> 2001:CC1E:1:3::/64
                    FEC0:234::3              0             0 300 i
*> FEC0:234::/64    FEC0:234::3              0             0 300 i
```


Task 3.4

R3:

```
router bgp 300
 !
 address-family ipv6
 neighbor 2001:CC1E:1::1 unsuppress-map UNSUPPRESS
 neighbor FEC0:234::2 unsuppress-map UNSUPPRESS
 aggregate-address 2001:CC1E:1::/62 summary-only
!
route-map UNSUPPRESS permit 10
```

Task 3.4 Verification

Verify that R4 receives only the summarized prefixes from R3:

```
Rack1R4#show bgp ipv6 unicast neighbors FEC0:234::3 routes  | begin Net
   Network          Next Hop            Metric LocPrf Weight Path
*> 2001:CC1E:1::/62 FEC0:234::3              0             0 300 i
*> FEC0:234::/64    FEC0:234::3              0             0 300 i

Total number of prefixes 2
```

Verify that R2 receives all prefixes (including the summary):

```
Rack1R2#show bgp ipv6 unicast neighbors FEC0:234::3 routes | begin Net
   Network          Next Hop            Metric LocPrf Weight Path
<output omitted>
*>i2001:CC1E:1::/64 FEC0:234::3              0    100      0 i
*>i2001:CC1E:1::/62 FEC0:234::3              0    100      0 i
*>i2001:CC1E:1:1::/64
                    2001:CC1E:1::1           0    100      0 200 i
*>i2001:CC1E:1:3::/64
                    FEC0:234::3              0    100      0 i
*>iFEC0:234::/64    FEC0:234::3              0    100      0 i
```


Task 5.1

R1:

```
interface Loopback0
 ip pim sparse-dense-mode
!
ip pim send-rp-discovery Loopback0 scope 16
ip pim rp-announce-filter rp-list 25 group-list 26
ip pim rp-announce-filter rp-list 50 group-list 51
!
access-list 25 permit 150.1.3.3
access-list 26 permit 239.0.0.0 0.255.255.255
access-list 50 permit 150.1.5.5
access-list 51 deny 224.0.0.0 1.255.255.255
access-list 51 deny 239.0.0.0 0.255.255.255
access-list 51 permit 224.0.0.0 15.255.255.255
```

R3:

```
interface Loopback0
 ip pim sparse-dense-mode
!
ip pim send-rp-announce Loopback0 scope 16 group-list 50
!
access-list 50 permit 239.0.0.0 0.255.255.255
```

R5:

```
interface Loopback0
 ip pim sparse-dense-mode
!
ip pim send-rp-announce Loopback0 scope 16 group-list 50
!
access-list 50 permit 226.0.0.0 1.255.255.255
access-list 50 permit 228.0.0.0 3.255.255.255
access-list 50 permit 232.0.0.0 3.255.255.255
access-list 50 permit 236.0.0.0 1.255.255.255
access-list 50 permit 238.0.0.0 0.255.255.255
```

Task 5.1 Breakdown

In previous labs, there was not a requirement to have the mapping agent control which specific groups were mapped with which candidate RPs (cRP). The requirement is given to have R1 map only specific multicast groups to R3 and R5. This will require the use of the **`ip pim rp-announce-filter`** command on the mapping agent (MA). The rp-announce filter will need to match the send-rp-announce filter used by the cRPs. If the groups requested by the RP do not match the mapping agent's filters, the cRPs requests that do not match will be discarded.

Example: If the cRP asks for 228.0.0.0/8 and 229.0.0.0/8, but the MA is allowing only 228.0.0.0/8, then the MA will filter the 229.0.0.0/8 and the cRP will be the RP for just 228/8. If the cRP asks for say 224.0.0.0/4 (all multicast groups), but the MA is only allowing 228.0.0.0/8, then the cRP's 224.0.0.0/4 announcement will be filtered by the MA and the cRP will not be the RP for any groups.

The logic of the access-list 51 used is as follows. The first step in finding out how to match all these addresses is to write them out in binary:

```
226 11100010
227 11100011
228 11100100
229 11100101
230 11100110
231 11100111
232 11101000
233 11101001
234 11101010
235 11101011
236 11101100
237 11101101
238 11101110
```

From the above addresses, it's evident that it's very close to a complete range, but is missing the 224, 225, and 239. If these numbers were included, then the range would be a contiguous block of 16 addresses (224 to 239). This address block would be matched as:

```
access-list 51 permit 224.0.0.0 15.255.255.255
```

However, now it overlaps 224, 225, and 239. These three can be removed with deny statements:

```
access-list 51 deny 224.0.0.0 0.255.255.255
access-list 51 deny 225.0.0.0 0.255.255.255
access-list 51 deny 239.0.0.0 0.255.255.255
access-list 51 permit 224.0.0.0 15.255.255.255
```

The total list is four lines now. Since we want the minimum number of lines, let's try to consolidate some of the deny statements. To do this we write out the three denied addresses in binary:

```
224 - 11100000
225 - 11100001
239 - 11101111
```

From the above address space, it is easily seen that 224 and 225 can be consolidated in one line (only the least significant bit is different), but the 239 cannot. Therefore these can be rewritten as:

```
     224 - 11100000
     225 – 11100001
Wildcard - 00000001

     239 - 11101111
Wildcard - 00000000
```

These groupings result in the following list:


```
access-list 51 deny 224.0.0.0 1.255.255.255
access-list 51 deny 239.0.0.0 0.255.255.255
access-list 51 permit 224.0.0.0 15.255.255.255
```

Now we have three lines. This may be the least amount of lines, but let's use just permit statements to be sure. Remember the addresses are as follows:

```
226 11100010
227 11100011
228 11100100
229 11100101
230 11100110
231 11100111
232 11101000
233 11101001
234 11101010
235 11101011
236 11101100
237 11101101
238 11101110
```

Now let’s group them together in ranges that can be checked without overlap:

```
226 11100010
227 11100011

228 11100100
229 11100101
230 11100110
231 11100111

232 11101000
233 11101001
234 11101010
235 11101011

236 11101100
237 11101101

238 11101110
```

This gives us five permit statements.

```
226 11100010
227 11100011
230 11100110
231 11100111

228 11100100
229 11100101
236 11101100
237 11101101

232 11101000
```



```
                                233 11101001
                                234 11101010
                                235 11101011

                                238 11101110
```

This gives us four. Better, but still not three. There's no way to get under four statements just by using permits. Therefore the correct answer for this section should be:

```
access-list 51 deny 224.0.0.0 1.255.255.255
access-list 51 deny 239.0.0.0 0.255.255.255
access-list 51 permit 224.0.0.0 15.255.255.255
```

Task 5.1 Verification

Below is the output from the **debug ip pim auto-rp** command on R1, with R5 configured to request all groups (no group-list on R5's **ip pim send-rp-announce** command), and R1 is configured with the **ip pim rp-announce-filter** shown above.

```
Rack1R1#debug ip pim auto-rp
PIM Auto-RP debugging is on
Rack1R1#
 Auto-RP: Received RP-announce, from 150.1.5.5, RP_cnt 1, ht 181
 Auto-RP: Filtered 224.0.0.0/4 for RP 150.1.5.5
Rack1R1#
```

As we can see, R5 request all multicast groups. This request does not match R1's filters and in turn was discarded (filtered). Now, we will add the *group-list* option to R5's **ip pim send-rp-announce** command that matches R1's **ip pim rp-announce-filter** *group-list*.

```
Rack1R1#
 Auto-RP: Received RP-announce, from 150.1.5.5, RP_cnt 1, ht 181
 Auto-RP: Update (226.0.0.0/7, RP:150.1.5.5), PIMv2 v1
 Auto-RP: Update (228.0.0.0/6, RP:150.1.5.5), PIMv2 v1
 Auto-RP: Update (232.0.0.0/6, RP:150.1.5.5), PIMv2 v1
 Auto-RP: Update (236.0.0.0/7, RP:150.1.5.5), PIMv2 v1
 Auto-RP: Update (238.0.0.0/8, RP:150.1.5.5), PIMv2 v1

Rack1R1#show ip pim rp mapping
PIM Group-to-RP Mappings
This system is an RP-mapping agent (Loopback0)

Group(s) 226.0.0.0/7
  RP 150.1.5.5 (?), v2v1
    Info source: 150.1.5.5 (?), via Auto-RP
         Uptime: 00:03:13, expires: 00:02:47
Group(s) 228.0.0.0/6
  RP 150.1.5.5 (?), v2v1
    Info source: 150.1.5.5 (?), via Auto-RP
```



```
         Uptime: 00:03:13, expires: 00:02:45
Group(s) 232.0.0.0/6
  RP 150.1.5.5 (?), v2v1
    Info source: 150.1.5.5 (?), via Auto-RP
         Uptime: 00:03:14, expires: 00:02:44
Group(s) 236.0.0.0/7
  RP 150.1.5.5 (?), v2v1
    Info source: 150.1.5.5 (?), via Auto-RP
         Uptime: 00:03:14, expires: 00:02:45
Group(s) 238.0.0.0/8
  RP 150.1.5.5 (?), v2v1
    Info source: 150.1.5.5 (?), via Auto-RP
         Uptime: 00:03:15, expires: 00:02:44
```

Task 5.2

R1:

```
interface FastEthernet0/0
 ip pim neighbor-filter 75
!
access-list 75 deny   192.10.1.254
access-list 75 permit any
```

R3:

```
interface FastEthernet0/0
 ip multicast boundary 51
!
access-list 51 deny   239.0.0.0 0.255.255.255
access-list 51 permit 224.0.0.0 15.255.255.255
```

R1, R2, R3, R5, SW2:

```
ip pim spt-threshold infinity group-list 52
!
access-list 52 permit 239.0.0.0 0.255.255.255
```


Task 5.2 Breakdown

To restrict PIM neighbor relationship, the **ip pim neighbor-filter** interface command was used for this section. This will not allow BB2 to become a PIM neighbor with R1, but will still allow clients on the FastEthernet segment to join any multicast group as this command only affects PIM neighbor relationships and not IGMP. This configuration can be verified by using the **show ip pim neighbors** command.

Although other methods can be used the control multicast traffic, the preferred method is to use the **ip multicast-boundary** interface command. The command requires an access-list that defines what groups are deny or permitted out the interface.

With multicasting, there normally is more than one receiver and in turn there can be more than one path the packets take through the network to reach the various receivers. These paths are commonly referred to as branches on the multicast tree. There are two types of multicast trees that can be formed in regards to this section's configuration. The default tree type is a source based tree. A source based tree is rooted at the source of the multicast stream. The tree is built using the least cost path between the source and destination or destinations. This is sometimes referred to a shortest path tree. The second type of tree, the one that this configuration will actually use, is called a shared tree. With a shared tree all multicast packets are sent to the RP and then down to the receivers.

It is interesting to note how routers perform the RPF check in regards to the two different types of trees. With a source based tree, the RPF check is done against the source of the multicast packets. With a shared tree, the RPF check is done against the RP and not against the source of the multicast stream. Using a shared tree, as opposed to the default of a source based tree, could possibly be used to workaround an issue with an RPF failure where static mroutes could not be used and the unicast routing could also not be easily changed.

For this task, the spt threshold was modified. By default, after traffic is received, there is a switchover from the shared tree via the RP to the shortest path tree. By setting the threshold to infinity, there will never be a switchover to the shortest path tree. Alternatively, another solution would be to configure bidirectional PIM for these groups, which would require configuring **ip pim bidir-enable** on the multicast routers, and configuring R3's advertisement with the bidir keyword.

Task 5.2 Verification

Try adding R6's interface Fa0/0 IP address to access-list 75:

```
Rack1R1#show ip access-lists 75
Standard IP access list 75
    10 deny   192.10.1.254
    15 deny   192.10.1.6 (3 matches)
```



```
    20 permit any (3 matches)
```

Next issue the following debug commands:

```
Rack1R1#debug ip pim
Rack1R1#debug ip pim hello

PIM(0): Send periodic v2 Hello on FastEthernet0/0
PIM(0): v2, PIM neighbor 192.10.1.6 explicitly denied on
FastEthernet0/0
```

Verify the multicast boundary:

```
Rack1R3#show ip multicast interface e0/0
Ethernet0/0 is up, line protocol is up
  Internet address is 162.1.38.3/24
  Multicast routing: enabled
  Multicast switching: fast
  Multicast packets in/out: 12771/43
                       51
  Multicast TTL threshold: 0
  Multicast Tagswitching: disabled

Rack1R3#show ip access-lists 51
Standard IP access list 51
    10 deny   239.0.0.0, wildcard bits 0.255.255.255
    20 permit 224.0.0.0, wildcard bits 15.255.255.255 (24 matches)
```


Join interface Fa0/0 at R5 to group 239.5.5.5, and then ping 239.5.5.5 from R1. Now check the multicast routing table at R5:

```
Rack1R5#show ip mroute
IP Multicast Routing Table
Flags: D - Dense, S - Sparse, B - Bidir Group, s - SSM Group, C -
Connected,
       L - Local, P - Pruned, R - RP-bit set, F - Register flag,
       T - SPT-bit set, J - Join SPT, M - MSDP created entry,
       X - Proxy Join Timer Running, A - Candidate for MSDP
Advertisement,
       U - URD, I - Received Source Specific Host Report,
       Z - Multicast Tunnel, z - MDT-data group sender,
       Y - Joined MDT-data group, y - Sending to MDT-data group
Outgoing interface flags: H - Hardware switched, A - Assert winner
 Timers: Uptime/Expires
 Interface state: Interface, Next-Hop or VCD, State/Mode

(*, 239.5.5.5), 00:01:08/00:02:25, RP 150.1.3.3, flags: SCL
  Incoming interface: Serial0/0/0, RPF nbr 162.1.0.3
  Outgoing interface list:
    FastEthernet0/0, Forward/Sparse-Dense, 00:01:08/00:02:25
<output omitted>
```

Note that the 'J' flag is not set for group 239.5.5.5 and there are no SPT entries in multicast routing table of R5. Next, join e0/0 of R5 to group 226.5.5.5 and ping the group from R1 again. Now, verify the multicast routing table at R5 to compare outputs:

```
Rack1R5#show ip mroute 226.5.5.5
IP Multicast Routing Table

<output omitted>

(*, 226.5.5.5), 00:00:20/stopped, RP 150.1.5.5, flags: SJCL
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Sparse-Dense, 00:00:20/00:02:39

(150.1.1.1, 226.5.5.5), 00:00:13/00:02:52, flags: T
  Incoming interface: Serial0/0/0, RPF nbr 162.1.0.3
  Outgoing interface list:
    FastEthernet0/0, Forward/Sparse-Dense, 00:00:13/00:02:46
<output omitted>
```

Note the 'J' flag and SPT entry with the 'T' flag set.

Task 6.1

R4:

```
interface FastEthernet0/0
 ip access-group IN_ACL in
 ip access-group OUT_ACL out
!
ip access-list extended IN_ACL
 permit icmp any any echo-reply
 permit tcp any eq telnet any established
 permit tcp any any eq bgp
 permit tcp any eq bgp any
 permit udp any any eq rip
 evaluate MY_REFLECT
!
ip access-list extended OUT_ACL
 permit tcp any any reflect MY_REFLECT
 permit udp any any reflect MY_REFLECT
 permit icmp any any reflect MY_REFLECT
```

Task 6.1 Breakdown

This task requires that ping and telnet packets be permitted to return. Since traffic originated by the router itself will not be reflected, the following static entries are needed:

```
ip access-list extended IN_ACL
 permit icmp any any echo-reply
 permit tcp any eq telnet any established
```

These access-list entries will allow telnet and ICMP echo replies inbound, even if they were not reflected. A workaround for this can be to policy route all ICMP and telnet traffic originated by the router out a loopback interface. When this occurs, the traffic will be reflected and in turn can by evaluated. Here is an example:

```
interface FastEthernet0/0
 ip access-group IN_ACL in
 ip access-group OUT_ACL out
!
ip access-list extended IN_ACL
 evaluate MY_REFLECT
ip access-list extended OUT_ACL
 permit tcp any any reflect MY_REFLECT
 permit udp any any reflect MY_REFLECT
 permit icmp any any reflect MY_REFLECT
```



```
Rack1R4#ping 204.12.1.254

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 204.12.1.254, timeout is 2 seconds:

ICMP: dst (204.12.1.4) administratively prohibited unreachable sent to
204.12.1.254.
ICMP: dst (204.12.1.4) administratively prohibited unreachable sent to
204.12.1.254.
ICMP: dst (204.12.1.4) administratively prohibited unreachable sent to
204.12.1.254.
ICMP: dst (204.12.1.4) administratively prohibited unreachable sent to
204.12.1.254.
ICMP: dst (204.12.1.4) administratively prohibited unreachable sent to
204.12.1.254.
Success rate is 0 percent (0/5)
Rack1R4#
```

As we can see from the output of the **debug ip icmp** command, the echo replies were not allowed to return. R4 denied the replies and in turn sent an administratively prohibited message to the source of the reply (BB3). Of course in a real network, we normally do not want the router telling the source that a packet was denied by an access-list and disable these replies by using the **no ip unreachables** command under the interfaces.

The workaround will involve policy routing the ICMP echo requests and telnet packets out the loopback 0 interface.

```
route-map LOCAL_TRAFFIC permit 10
 match ip address ORIGINATED
 set interface Loopback0
!
ip access-list extended ORIGINATED
 permit icmp any any
 permit tcp any any eq telnet
```

This configuration will allow the packets originated by the router itself to be reflected.

```
Rack1R4#ping 204.12.1.254

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 204.12.1.254, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/9 ms
Rack1R4#
```


To further verify that the policy routing is working, below is the same ping with the **debug ip policy** command enabled.

```
Rack1R4#debug ip policy
Policy routing debugging is on
Rack1R4#ping 204.12.1.254 repeat 1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 204.12.1.254, timeout is 2 seconds:
!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/9/12 ms
Rack1R4#
IP: s=204.12.1.4 (local), d=204.12.1.254, len 100, policy match
IP: route map LOCAL_TRAFFIC, item 10, permit
IP: s=204.12.1.4 (local), d=204.12.1.254 (Loopback0), len 100, policy
routed
IP: local to Loopback0 204.12.1.254
Rack1R4#
```

Alternatively, this section could be completed using CBAC.

Task 6.1 Verification

Verify that R4 can ping and telnet to BB3:

```
Rack1R4#ping 204.12.1.254

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 204.12.1.254, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/8 ms

Rack1R4#telnet 204.12.1.254
Trying 204.12.1.254 ... Open

BB3>exit

[Connection to 204.12.1.254 closed by foreign host]
```

Verify that RIP and BGP are OK:

```
Rack1R4#show ip route rip
     31.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
R       31.3.0.0/16 [120/1] via 204.12.1.254, 00:00:25, FastEthernet0/0
R       31.2.0.0/16 [120/1] via 204.12.1.254, 00:00:25, FastEthernet0/0
R       31.1.0.0/16 [120/1] via 204.12.1.254, 00:00:25, FastEthernet0/0
R       31.0.0.0/16 [120/1] via 204.12.1.254, 00:00:25, FastEthernet0/0
<output omitted>
```



```
Rack1R4#show ip bgp summary | begin Neighbor
Neighbor    V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down  State/PfxRcd
2001:204:12:1::254
            4 54    46      52       0    0    0 00:40:55 (NoNeg)
150.1.5.5   4 500  195     190      26    0    0 02:59:24        1
162.1.0.3   4 300  192     190      26    0    0 02:59:36        5
204.12.1.254
            4 54   191     191      26    0    0 02:59:18       10
FEC0:234::3 4 300   66      70      26    0    0 00:35:48        0
```

Verify that internal routers can still ping and telnet to BB3:

```
Rack1R3#ping 204.12.1.254

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 204.12.1.254, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 60/62/64 ms

Rack1R3#telnet 204.12.1.254
Trying 204.12.1.254 ... Open

BB3>exit

[Connection to 204.12.1.254 closed by foreign host]
```

And finally, verify that BB3 can not initiate connections to the internal routers:

```
BB3>show ip route 150.1.3.3
Routing entry for 150.1.0.0/20
  Known via "rip", distance 120, metric 2
  Redistributing via rip
  Last update from 204.12.1.4 on FastEthernet0, 00:00:13 ago
  Routing Descriptor Blocks:
  * 204.12.1.4, from 204.12.1.4, 00:00:13 ago, via FastEthernet0
      Route metric is 2, traffic share count is 1


BB3>telnet 150.1.3.3
Trying 150.1.3.3 ...
% Destination unreachable; gateway or host down
```


Task 6.2

SW1:

```
vlan access-map ICMP_FILTER 10
 action drop
 match ip address 100
vlan access-map ICMP_FILTER 20
 action forward
vlan filter ICMP_FILTER vlan-list 162
!
access-list 100 permit icmp 205.90.31.0 0.0.0.255 any echo
```

☞ Quick Note

The sequence of the **action** and **match** commands within a **vlan filter** are irrelevant. The IOS will always show the **action** before the **match** irrespective to the order they are entered.

Task 6.2 Breakdown

A VLAN access-list, commonly referred to as a VACL, uses the same basic logic as a route-map. A VACL contains sequence numbers along with *match* and *action* criteria like a route-map with its *match* and *set* criteria. In a VACL, the match can be an IP or MAC access-list. The action will always be forward of drop. The action will either forward the traffic that is matched, or drop the traffic that is matched. If the **`vlan access-map`** sequence does not contain a match statement, all traffic will match.

This means that the logic of the access-list is important in that traffic that you want to deny will need to be permitted in your access-list. By permitting the traffic in your access-list, it will match and in turn the action will be taken. Of course, you could reverse the logic and match traffic that you want to permit and use the action of *forward* on it. Then create another **`vlan access-map`** sequence that just has an *action drop* and no match statement. Here is an example:

```
vlan access-map ICMP_FILTER 10
 action forward
 match ip address 100
vlan access-map ICMP_FILTER 20
 action drop
vlan filter ICMP_FILTER vlan-list 162
!
access-list 100 deny icmp 205.90.31.0 0.0.0.255 any echo
access-list 100 permit ip any any
```

☠ Pitfall

Because VACLs are compiled to improve performance, changes made to an access-map will not take affect till the **vlan filter** is removed and reapplied. Be sure to remove and then reapply the **vlan filter** command whenever changes are made to the **vlan access-map** commands.

In the above configuration, the access-list logic is reversed. Here access-list 100 denies ICMP sourced from 205.90.31.0/24 and permits all other IP traffic. Since the ICMP traffic from 205.90.31.0/24 is not 'positive' in the access-list, it will not match and in turn not get forwarded. The ICMP traffic will match **vlan access-map** sequence number 20, since there is not a match statement. This will, in turn, cause the ICMP traffic to be dropped.

Although this configuration appears fine, it will cause problems. The problem is that access-list 100 does not permit ARP. The IP FastEthernet and ARP FastEthernet types are not the same. ARP will not match **vlan access-map** sequence number 10 but will match 20, which has a default action of drop. This configuration will appear to work for a few hours or until a reboot, assuming the devices have already created an ARP entry before the **vlan filter** was applied. Someone could leave the lab with the assumption that their configurations are correct, but once the devices are reloaded and need to ARP, everything that relies on ARP will stop working (routing protocols, ping, etc). Here is an example:

```
Rack1R1#show arp
Protocol  Address   Age (min)  Hardware Addr   Type   Interface
Internet  192.10.1.254     1   00e0.1ece.4a68  ARPA   FastEthernet0/0
Internet  192.10.1.1       -   00d0.586e.b720  ARPA   FastEthernet0/0
Rack1R1#ping 192.10.1.254

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.10.1.254, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
```

Now the ARP cache is cleared, so that R1 will need to ARP for BB2. Then, we try to ping from R1 (192.10.1.1) to BB2 (192.10.1.254) again.

```
Rack1R1#clear arp
Rack1R1#show arp
Protocol  Address   Age (min)  Hardware Addr   Type   Interface
Internet  192.10.1.1       -   00d0.586e.b720  ARPA   FastEthernet0/0
Rack1R1#ping 192.10.1.254

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.10.1.254, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
Rack1R1#
```

As we can see this configuration broke ARP. To permit ARP, a MAC access-list will need to be created that matches the ARP FastEthernet type. A new **vlan filter** sequence will also need to be added. Do not forget to remove the **vlan filter** and reapply it.


```
mac access-list extended PERMIT_ARP
 permit any any 0x806 0x0
!
!
vlan access-map ICMP_FILTER 10
 action forward
 match ip address 100
vlan access-map ICMP_FILTER 15
 action forward
 match mac address PERMIT_ARP
vlan access-map ICMP_FILTER 20
 action drop
vlan filter ICMP_FILTER vlan-list 162
!
access-list 100 deny icmp 205.90.31.0 0.0.0.255 any echo
access-list 100 permit ip any any

Rack1SW2#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1SW2(config)#no vlan filter ICMP_FILTER vlan-list 162
Rack1SW2(config)#vlan filter ICMP_FILTER vlan-list 162
Rack1SW2(config)#^Z
Rack1SW2#

Rack1R1#ping 192.10.1.254

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.10.1.254, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 1/3/4 ms
Rack1R1#
```

Task 6.2 Verification

Verify that the VLAN filter is applied:

```
Rack1SW1#show vlan filter
VLAN Map ICMP_FILTER is filtering VLANs:
  162
```

Verify that R1 can still ping addresses in the 205.90.31.0/24 network

```
Rack1R1#ping 205.90.31.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 205.90.31.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/8 ms
```

To verify that the filter works add another entry to access-list 100:

```
access-list 100 permit icmp 205.90.31.0 0.0.0.255 any echo-reply
```


And try ping again:

```
Rack1R1#ping 205.90.31.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 205.90.31.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)

Rack1R1#ping 192.10.1.6

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.10.1.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
```

Task 7.1

R6:

```
access-list 25 permit 192.10.1.101
access-list 25 deny   any log
!
snmp-server community CISCORO RO 25
snmp-server community CISCORW RW 25
snmp-server trap-source Loopback0
snmp-server location San Jose, CA US
snmp-server contact CCIE Lab R6
snmp-server chassis-id 556-123456
snmp-server host 192.10.1.101 traps CISCOTRAP
```

Task 7.1 Breakdown

Although this is a relatively standard SNMP configuration, the task requiring to configure logging could be difficult to interpret. This key is the word “logged”. Even though the router could notify the NMS via an SNMP trap by using the **snmp-server enable traps snmp authentication** command, the tasked requested that the failed attempts be logged. To *log* the failed attempts, an access-list entry was used that denied all other IP addresses and logged them. Of course, in a real network, a remote syslog server would also be configured or at least **logging buffered** would have been enabled.

Task 7.1 Verification

Verify basic SNMP configuration:

```
Rack1R6#show snmp
Chassis: 556-123456
Contact: CCIE Lab R6
Location: San Jose, CA US
<output omitted>
SNMP logging: enabled
    Logging to 192.10.1.101.162, 0/10, 0 sent, 0 dropped.
```

Verify that logging is configured for access-list 25:


```
Rack1R6#show ip access-lists 25
Standard IP access list 25
    10 permit 192.10.1.101
    20 deny   any log
```


Task 7.2

R4 and R5:

```
logging trap notifications
logging origin-id hostname
logging facility sys10
logging source-interface Loopback0
logging 192.10.1.101
```

Task 7.2 Breakdown

By default, syslog messages do not include the hostname of the device in them. In a real network, this can be very useful when viewing the syslog messages on the syslog server itself. The **logging origin-id** command enables the hostname, IP address, or even (as of 12.3[1]), a user defined string in the log messages.

Task 7.2 Verification

Verify that logging is configured and the origin-id is set:

```
Rack1R5#show logging
Syslog logging: enabled (11 messages dropped, 1 messages rate-limited,
<output omitted>

    Trap logging: level notifications, 78 message lines logged
        Logging to 192.10.1.101 (udp port 514, audit disabled, link
up), 4 message lines logged, xml disabled,
              filtering disabled

Rack1R5#show running-config | include origin-id
logging origin-id hostname
```


Task 7.3

R6:

```
ip name-server 192.10.1.100
!
ip domain-lookup
!
line con 0
 transport preferred none
```

Task 7.3 Breakdown

Although this could be considered a Stupid Router Trick (SRT), it is a very useful command to have enabled in a lab environment or even real network when DNS is being used. The **`transport preferred none`** line command will stop the router from trying to resolve a mistyped command via DNS.

The router is technically attempting to resolve the string entered via DNS as it's trying to telnet to a device with that particular name. Once the **`transport preferred none`** command is enabled, you will need to use the **`telnet`** exec mode command to telnet to another device.

Task 7.4

R6:

```
enable secret level 2 CISCO
!
privilege interface level 2 ip access-group
privilege interface all level 2 encapsulation
privilege configure level 2 hostname
privilege configure level 2 interface
privilege exec level 2 show run
```

Just because the show run command is moved to a lower level, it doesn't mean a user at that level can see the entire configuration. A user will only be able to see the items that they have the appropriate privilege level for. Make sure to test!! You may want to add an access-list to an interface to verify that it shows up in the output of show run.

☑ Verification

```
Rack1R6>enable 2
Password:
Rack1R6#show privilege
Current privilege level is 2
Rack1R6#show run
Building configuration...

Current configuration : 260 bytes
!
!
```



```
hostname Rack1R6
!
boot-start-marker
boot-end-marker
!
!
!
!
!
interface Loopback0
!
interface FastEthernet0/0
!
interface Serial0/0/0
 encapsulation frame-relay
!
interface Serial0/0/0.1 multipoint
!
interface FastEthernet0/1
 ip access-group 101 in
!
!

End
```


Task 8.1

R1:

```
policy-map SHAPE_384K
 class class-default
  shape average 384000 38400 12800
  shape adaptive 256000
!
interface Serial0/0
 service-policy output SHAPE_384K
```

Task 8.1 Breakdown

The following values on R1 can be inferred from this task's description:

AR (port speed) = 512000 bps
CIR (average rate) = 384000bps
Tc (interval length) = 100 ms

The following values are then calculated based on the given values and the below formula:

Bc = CIR * Tc/1000
Bc = 38400 bits

Be = (AR – CIR) * Tc/1000
Be = 12800 bits

Task 8.1 Verification

Verify the shaping parameters:

```
R1#show policy-map interface serial 0/0

 Serial0/0

  Service-policy output: SHAPE_384K

    Class-map: class-default (match-any)
      10 packets, 797 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Traffic Shaping
           Target/Average   Byte   Sustain   Excess    Interval
Increment
             Rate           Limit  bits/int  bits/int  (ms)
(bytes)
           384000/384000    6400   38400     12800     100       4800

        Adapt  Queue     Packets   Bytes     Packets   Bytes
Shaping
        Active Depth                         Delayed   Delayed   Active
        -      0         6         745       0         0         no
```

Task 8.2

R3:

```
interface Serial1/0
 frame-relay map ip 162.1.0.4 304 broadcast rtp header-compression
passive connections 15
```

R4:

```
interface Serial0/0/0
 frame-relay map ip 162.1.0.3 403 broadcast rtp header-compression
connections 15
```

Task 8.2 Breakdown

RTP header compression allows the reduction of the header inside an RTP packet to be reduced from 40 bytes to 2 – 5 bytes. RTP compression is best used on low speed links for real time traffic with small data payloads, such as VoIP. To configure RTP on a serial link, use the **`ip rtp header-compression`** command. For Frame Relay links, RTP compression can be configured on a per VC basis as in the above example. The **`passive`** keyword of the RTP statement means that the router will not start sending RTP compressed headers unless RTP headers are received.

Task 8.2 Verification

Verify the per-VC configured RTP header compression:

```
Rack1R4#show frame-relay map

<output omitted>

Serial0/0/0 (up): ip 162.1.0.3 dlci 403(0x193,0x6430), static,
              broadcast,
              CISCO, status defined, active
              RTP Header Compression (enabled), connections: 15
```

Task 8.3

R2:

```
ip cef
!
class-map match-all SQL
 match protocol sqlserver
!
!
 policy-map SQL_POLICY
  class SQL
   shape average 256000
   shape max-buffers 2048
!
interface Serial0/0.1 point-to-point
 service-policy output SQL_POLICY
```

Task 8.3 Breakdown

The above task uses Network Based Application Recognition to match SQL traffic (TCP port 1433). As SQL traffic leaves the serial interface, it is shaped to 256Kbps. The shaping buffers are modified to allow up to 2048 packets to sit in the shaping queue while waiting to be sent out. A large shaping queue is advantageous for delay insensitive data traffic for which you do not want to be dropped.

The above configuration is known as Generic Traffic Shaping. GTS uses the same principles and calculations as does Frame Relay Traffic Shaping, but does not adaptively shape, and is supported on non-Frame Relay interfaces

Task 8.4

SW1:

```
mls qos
!
! Enable VLAN-based QoS on physical interfaces
!
interface range Fa0/13 - 21
 mls qos vlan-based

!
! Access-Lists
!
ip access-list extended TCP
 permit tcp any any
!
ip access-list extended UDP
 permit udp any any
!
mac access-list extended IPX
 permit any any 0x8137 0x0

!
! First-level class-maps
!
class-map TCP
 match access-group name TCP
!
class-map UDP
 match access-group name UDP
!
class-map IPX
 match access-group name IPX

mls qos map policed-dscp 0 to 8

!
! For 2nd level policy you can only match input interfaces
!
class-map TRUNKS
 match input-interface FastEthernet 0/13 -  FastEthernet 0/21

!
! Second-level policy-map may only police, but not mark
!
policy-map INTERFACE_POLICY_TCP
 class TRUNKS
  police 1000000 16000 exceed policed-dscp-transmit

!
! Second-level policy-map may only police, but not mark
!
policy-map INTERFACE_POLICY_UDP
 class TRUNKS
  police 512000 16000 exceed drop

!
```



```
! Second-level policy-map may only police, but not mark
!
policy-map INTERFACE_POLICY_IPX
 class TRUNKS
  police 128000 16000 exceed drop

!
! 1st level policy-map may only mark, not police
! VAN aggregate policing is not possible in the 3560
!
policy-map VLAN_POLICY
 class TCP
  set dscp 0
  service-policy INTERFACE_POLICY_TCP
!
! Nothing is told about UDP marking and we need to set something.
! Thus we just select the default dscp value of zero.
!
 class UDP
  set dscp 0
  service-policy INTERFACE_POLICY_UDP
 class IPX
  set dscp cs2
  service-policy INTERFACE_POLICY_IPX
!
interface Vlan 162
 service-policy input VLAN_POLICY
```

