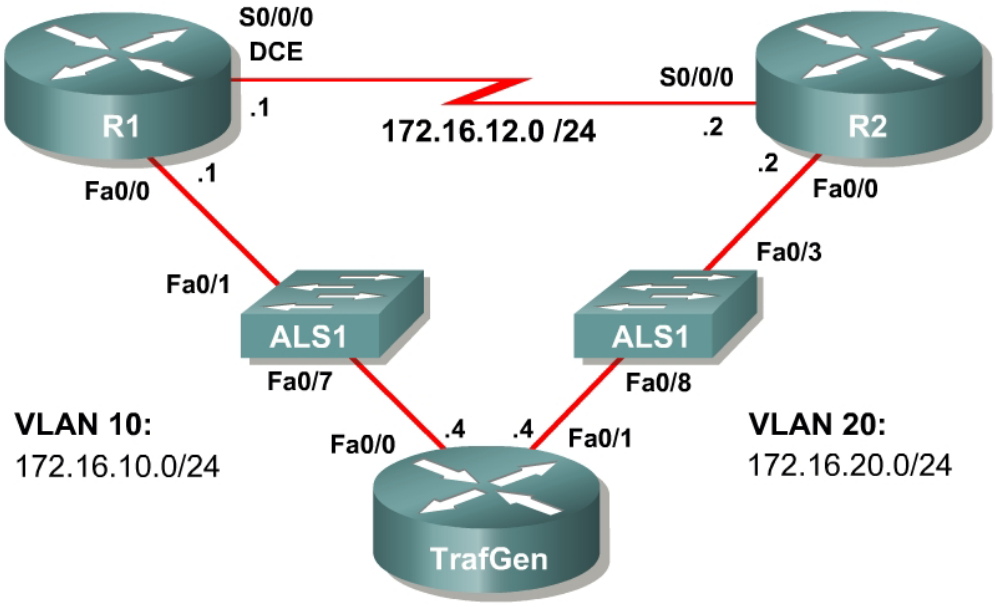
Lab 4.4 Comparing Queuing Strategies

Learning Objectives

- Implement FIFO, WFQ, CQ, and PQ queuing strategies
- Compare queuing strategies using the NQR tool

Topology Diagram



Scenario

This lab is designed as an integration lab to help you assess and recall skills learned in Labs 4.1 and 4.2. You will use some of the packet analysis tools available in the Pagent toolset to compare different queuing strategies and their impact on end-to-end quality of service (QoS). The four different queuing strategies that will be configured in this lab are first-in, first-out (FIFO), weighted fair queuing (WFQ), custom queuing (CQ), and priority queuing (PQ).

This is an investigative lab, so be sure to tweak the queuing strategies to ameliorate the results of your configurations. Compare results with classmates and contrast the configurations that provide those results.

Typically, commands and command output will only be shown if they have not been implemented in preceding labs, so it is highly recommended that you complete the previous labs to ensure knowledge of the queuing strategies and their configurations.

Preparation

This lab relies on the Basic Pagent Configuration, which you should have created in Lab 3.1: Preparing for QoS.

Prior to beginning this lab, configure R4 and the switch according to the Basic Pagent Configuration. You may easily accomplish this on R4 by loading the *basic-ios.cfg* file from flash memory into the NVRAM, and reloading.

```
TrafGen# copy flash:basic-ios.cfg startup-config
Destination filename [startup-config]?
[OK]
2875 bytes copied in 1.456 secs (1975 bytes/sec)
TrafGen# reload
Proceed with reload? [confirm]
```

On the switch, load the *basic.cfg* file into NVRAM and reload the device.

```
ALS1# copy flash:basic.cfg startup-config
Destination filename [startup-config]?
[OK]
2875 bytes copied in 1.456 secs (1975 bytes/sec)
ALS1# reload
Proceed with reload? [confirm]
```

Unlike Labs 4.1 and 4.2, this lab will use the NQR tool in the Pagent toolset rather than the TGN traffic generator. Do not load the TGN traffic generator configuration.

In addition, add the Fast Ethernet 0/3 interface on the switch to VLAN 20 since R2 will be the exit point from the network topology in this lab.

```
ALS1# configure terminal
ALS1(config)# interface fastethernet 0/3
ALS1(config-if)# switchport access vlan 20
ALS1(config-if)# switchport mode access
```

Step 1: Configure Addressing and Routing

Configure all IP addresses shown in the topology diagram and use a clock rate of 800 kbps on the serial link between R1 and R2. Set the informational bandwidth parameter appropriately on the serial interfaces.

Configure EIGRP AS 1 to include all networks shown in the diagram.

```
R1(config)# interface fastethernet 0/0
R1(config-if)# ip address 172.16.10.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# interface serial 0/0/0
R1(config-if)# bandwidth 800
R1(config-if)# ip address 172.16.12.1 255.255.255.0
R1(config-if)# clockrate 800000
R1(config-if)# no shutdown
R1(config-if)# router eigrp 1
R1(config-router)# network 172.16.0.0
```



```
R2(config)# interface fastethernet 0/0
R2(config-if)# ip address 172.16.20.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# interface serial 0/0/0
R2(config-if)# bandwidth 800
R2(config-if)# ip address 172.16.12.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# router eigrp 1
R2(config-router)# network 172.16.0.0
```

Step 2: Create NQR Configuration for Testing Purposes

Traffic generated from NQR, the traffic generation component of Pagent, requires almost all header fields to be hardcoded. Since the packets will be generated over Ethernet, you need to set the destination MAC address of the packets so that they are not broadcast. Remember that this is only the destination for the first hop, not the final destination MAC address. Use the **show interfaces** command to discover the value of the 48-bit MAC address.

Example:

```
R1# show interfaces fastethernet0/0
FastEthernet0/0 is up, line protocol is up
  Hardware is MV96340 Ethernet, address is 0019.0623.4380 (bia 0019.0623.4380)
<OUTPUT OMITTED>
```

Use the MAC address on R1 as the Layer 2 destination of the NQR stream you will configure next.

On R4, issue the **nqr** command in privileged EXEC mode to enter NQR configuration mode. Then, copy and paste the NQR configuration shown below into a text editor, such as Notepad, and replace the $R1_MAC$ field with the MAC address you displayed in the output of the **show interfaces fastethernet 0/0** command. Then, copy and paste that configuration into the TrafGen router.

```
fastethernet0/0
add tcp
send 1000
rate 60
length random 200 to 1000
l2-dest $R1_MAC$
l3-src 172.16.10.4
l3-dest 172.16.20.4
l4-dest 23
fastethernet0/1 capture
add clone-of 1
l4-dest 21
add clone-of 1
l4-dest 119
add clone-of 1
l4-dest 22
add clone-of 1
l4-dest 6000
```

To begin NQR testing, issue either the **start send** command in NQR configuration mode or the **nqr start send** command from privileged EXEC

mode. Time will pass, and then the router will inform you when all packets have been sent. There is no need to stop the streams since they will stop on their own.

Finally, issue the **show pkt-seq-drop-stats**, **show delay**, and **show jitter** NQR commands to display drop/resequencing, delay, and jitter statistics, respectively. Example output is shown below, although this type of output will not be shown again later in the lab. Record all statistics by copying and pasting them into a text editor such as Notepad. Record a baseline reading for your current topology.

```
R4(NQR:OFF,Fa0/0:5/5)# start send
R4(NQR:SEND,Fa0/0:5/5)#

   Send process complete.

R4(NQR:WAIT,Fa0/0:5/5)#
R4(NQR:OFF,Fa0/0:5/5)# show pkt-seq-drop-stats

Summary of packet sequence/drop stats of traffic streams
  ts#   template interface      sent     recvd   dropped   out-of-seq  max-seq
  1     TCP      Fa0/0.10*      1000       625       375          271       28
  2     TCP      Fa0/0.10*      1000       637       363          271       30
  3     TCP      Fa0/0.10*      1000       638       362          254       30
  4     TCP      Fa0/0.10*      1000       598       402          265       29
  5     TCP      Fa0/0.10*      1000       604       396          267       28

R4(NQR:OFF,Fa0/0:5/5)# show delay-stats

Summary of delay-stats of traffic streams
 ts#    template interface    min-delay    max-delay    avg-delay  stdev-delay
  1     TCP      Fa0/0.10*     0.013646     0.433202     0.355633     0.047306
  2     TCP      Fa0/0.10*     0.012966     0.426203     0.352435     0.048258
  3     TCP      Fa0/0.10*     0.008824     0.436855     0.357987     0.046055
  4     TCP      Fa0/0.10*     0.028379     0.448521     0.361942     0.049450
  5     TCP      Fa0/0.10*     0.015277     0.457674     0.363785     0.046969

R4(NQR:OFF,Fa0/0:5/5)# show jitter-stats

Summary of jitter-stats of traffic streams
 ts#    template interface   min-jitter   max-jitter   avg-jitter stdev-jitter
  1     TCP      Fa0/0.10*     0.000063     0.204891     0.033416     0.034363
  2     TCP      Fa0/0.10*     0.000098     0.190365     0.034329     0.034809
  3     TCP      Fa0/0.10*     0.000015     0.172803     0.033511     0.032503
  4     TCP      Fa0/0.10*     0.000047     0.223152     0.035887     0.034892
  5     TCP      Fa0/0.10*     0.000070     0.165289     0.035484     0.031709
```

Step 3: Test FIFO Queuing

This lab will compare four different queuing types. The first type is the most basic, FIFO queuing.

Configure FIFO queuing on the serial interface on R1. Recall that disabling all other queuing strategies on an interface will enable FIFO queuing.

Notice that the scenario the authors have designed overpowers all of the queuing mechanisms implemented because there is simply much more traffic

than the bandwidth of the serial link. If you had this ratio of legitimate traffic to bandwidth in a production network, then queuing strategies would not solve the problem. It would be necessary to obtain additional bandwidth.

Step 4: Test Weighted Fair Queuing

Enable WFQ on the serial interface. Run the NQR streams again using **nqr start send** and compare the results of the **show** commands.

```
R1(config)# interface serial0/0/0
R1(config-if)# fair-queue
```

Is there a significant difference between the statistics using WFQ and FIFO in this scenario?

No.

The streams from NQR are generated in something similar to a round-robin fashion with the same number of packets for each stream. The result is that many of the same packets will be forwarded by WFQ as by FIFO, but this is only by the construction of the streams on TrafGen. In real networks, many traffic patterns are bursty, unlike this simulation. To understand what is meant by bursty traffic patterns, think of loading a web page. You type in a URL and there is a burst of traffic as the text and the graphics load. Then while you read the web page, there is no additional traffic being sent across the network. Then you click on a link, and another burst of traffic traverses the network.

What effect does the function of the NQR generator have on your results?

Since the packets are being generated in a round-robin fashion, WFQ and FIFO perform very similarly so WFQ does not have much of an advantage over FIFO.

Provide a circumstance in which you would expect a different result from FIFO?

If the generated flows were more mixed than in the current scenario, you would expect lower latency with WFQ than with FIFO.

Step 5: Test Custom Queuing

Configure custom queuing (CQ) on R1's serial interface. Place each traffic stream in its own queue but do not customize any parameters of it. (The port numbers configured for the NQR streams are TCP ports 23, 21, 119, 22, and 6000) Run the NQR streams and compare results as you did before.


```
R1(config)# queue-list 1 protocol ip 1 tcp telnet
R1(config)# queue-list 1 protocol ip 2 tcp ftp
R1(config)# queue-list 1 protocol ip 3 tcp nntp
R1(config)# queue-list 1 protocol ip 4 tcp 22
R1(config)# queue-list 1 default 5
R1(config)# interface serial0/0/0
R1(config-if)# custom-queue-list 1
```

Contrast the results for the CQ test with those of the previous queuing strategies.

Delay, jitter, and dropped packets were all worse (higher) than with the other queuing strategies. Because the queues are services in a round-robin fashion, delay increases over FIFO and WFQ while packets sit in the queue.

Try making one of the queues have a size of 10000. How does this affect all of the traffic flows?

```
R1(config)# queue-list 1 queue 1 byte-count 10000
```

The affected flow does not lose any packets and has lower jitter and delays than the other flows. The other ones have more lost packets, as well as having higher delays and jitter as packets wait for the large queue to reach its byte count.

Step 6: Test Priority Queuing

Configure priority queuing (PQ) on R1 on the serial interface facing R2. Assign one of the application protocols in use to the high priority queue, one to the medium queue, one to the normal queue, and make the low priority queue the default queue. Run the NQR streams and compare results as you did before.

```
R1(config)# priority-list 1 protocol ip high tcp telnet
R1(config)# priority-list 1 protocol ip medium tcp ftp
R1(config)# priority-list 1 protocol ip normal tcp 22
R1(config)# priority-list 1 default low
R1(config)# interface serial0/0/0
R1(config-if)# no custom-queue-list 1
R1(config-if)# priority-group 1
```

How does the packet loss with PQ compare to that of previous queuing strategies?

The higher priority streams get no packet loss and very low delay and jitter. There is nearly full loss on the lower priority streams, and high delay and jitter (when there is enough data for statistics).

What would happen if you put all the streams in the high priority queue?

This would effectively make the interface use FIFO queuing.

Final Configurations

```
R1# show run
!
hostname R1
!
interface FastEthernet0/0
 ip address 172.16.10.1 255.255.255.0
 no shutdown
!
interface Serial0/0/0
 ip address 172.16.12.1 255.255.255.0
 priority-group 1
 clock rate 800000
 no shutdown
!
router eigrp 1
 network 172.16.0.0
 no auto-summary
!
queue-list 1 protocol ip 1 tcp telnet
queue-list 1 protocol ip 2 tcp ftp
queue-list 1 protocol ip 3 tcp nntp
queue-list 1 protocol ip 4 tcp 22
queue-list 1 default 5
queue-list 1 queue 1 byte-count 10000
priority-list 1 protocol ip high tcp telnet
priority-list 1 protocol ip medium tcp ftp
priority-list 1 protocol ip normal tcp 22
priority-list 1 default low
!
end
```

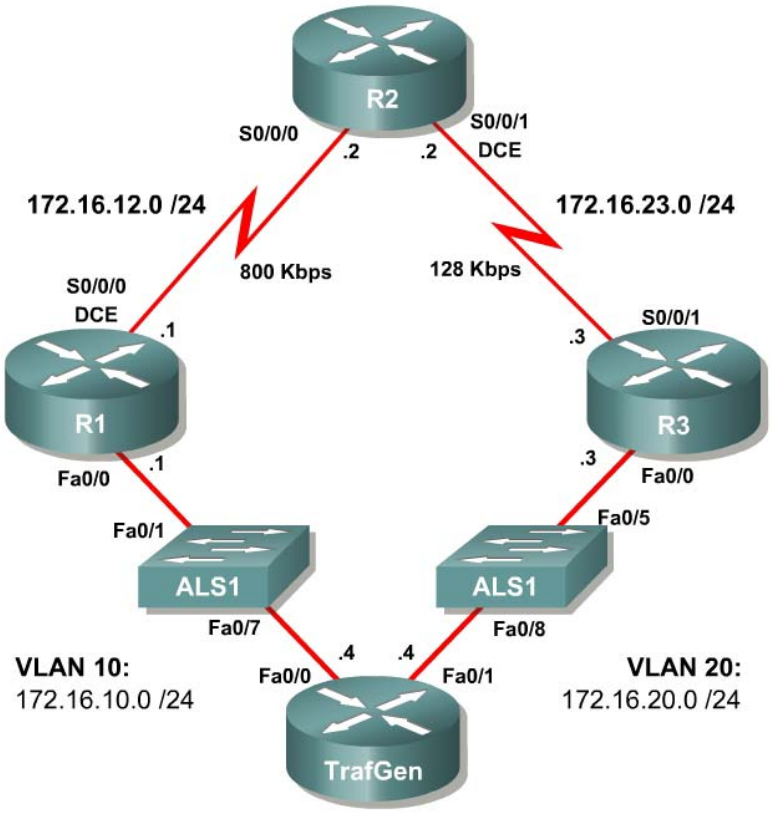
```
R2# show run
!
hostname R2
!
interface FastEthernet0/0
 ip address 172.16.20.2 255.255.255.0
 no shutdown
!
interface Serial0/0/0
 ip address 172.16.12.2 255.255.255.0
 no shutdown
!
router eigrp 1
 network 172.16.0.0
 no auto-summary
!
end
```


Lab 4.5 Class-based Queuing and NBAR

Learning Objectives

- Utilize NBAR for protocol detection
- Mark IP Precedence
- Allocate bandwidth using the Modular QoS Command-Line Interface
- Configure CBWFQ and LLQ queuing strategies

Topology Diagram



Scenario

In this lab, you will implement classification using Network-based Application Recognition (NBAR) and the Modular QoS CLI (MQC) to configure quality of service (QoS) on R1 and R2. You will configure both class-based marking and class-based queuing algorithms.

Preparation

This lab uses the Basic Pagent Configuration for TrafGen and the switch to generate and facilitate lab traffic in a stream from TrafGen to R1 to R2. Prior to beginning this lab, configure TrafGen (R4) and the switch according to the Basic Pagent Configuration in Lab 3.1: Preparing for QoS. You can accomplish this on R4 by loading the *basic-ios.cfg* file from flash memory into the NVRAM and reloading.

```
TrafGen# copy flash:basic-ios.cfg startup-config
Destination filename [startup-config]?
[OK]
2875 bytes copied in 1.456 secs (1975 bytes/sec)
TrafGen# reload
Proceed with reload? [confirm]
```

On the switch, load the *basic.cfg* file into NVRAM and reload the device.

```
Switch# copy flash:basic.cfg startup-config
Destination filename [startup-config]?
[OK]
2875 bytes copied in 1.456 secs (1975 bytes/sec)
TrafGen# reload
Proceed with reload? [confirm]
```

On TrafGen, instruct TGN to load the *basic-tgn.cfg* file and to start generating traffic.

```
TrafGen> enable
TrafGen# tgn load-config
TrafGen# tgn start
```

In addition, add the Fast Ethernet 0/5 interface on the switch to VLAN 20 since R3 will be the exit point from the network topology in this lab.

```
Switch# configure terminal
Switch(config)# interface fastethernet 0/5
Switch(config-if)# switchport access vlan 20
Switch(config-if)# switchport mode access
```

Step 1: Configure the Physical Interfaces

Configure all of the physical interfaces shown in the diagram. Set the clock rate on the serial link between R1 and R2 to 800000, the clock rate of the serial link between R2 and R3 to be 128000, and use the **no shutdown** command on all interfaces. Set the informational bandwidth parameter on the serial interfaces.

```
R1(config)# interface fastethernet 0/0
R1(config-if)# ip address 172.16.10.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# interface serial 0/0/0
R1(config-if)# bandwidth 800
R1(config-if)# ip address 172.16.12.1 255.255.255.0
R1(config-if)# clock rate 800000
```



```
R1(config-if)# no shutdown

R2(config)# interface serial 0/0/0
R2(config-if)# bandwidth 800
R2(config-if)# ip address 172.16.12.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# interface serial 0/0/1
R2(config-if)# bandwidth 128
R2(config-if)# ip address 172.16.23.2 255.255.255.0
R2(config-if)# clock rate 128000
R2(config-if)# no shutdown

R3(config)# interface fastethernet 0/0
R3(config-if)# ip address 172.16.20.3 255.255.255.0
R3(config-if)# no shutdown
R3(config-if)# interface serial 0/0/1
R3(config-if)# bandwidth 128
R3(config-if)# ip address 172.16.23.3 255.255.255.0
R3(config-if)# no shutdown
```

Issue the **show interfaces serial 0/0/0 | include Queueing** command on R1 to verify that the queuing strategy is Weighted Fair Queuing (WFQ).

```
R1# show interface serial0/0/0 | include Queueing
  Queueing strategy: weighted fair
```

If you see “fifo” as the queuing type, use the interface-level command **fair-queue** on the serial interface.

Step 2: Configure EIGRP AS 1

Configure routing between R1, R2 and R3 using Enhanced Interior Gateway Routing Protocol (EIGRP). Include the entire 172.16.0.0/16 major network in AS 1 and disable automatic summarization.

```
R1(config)# router eigrp 1
R1(config-router)# no auto-summary
R1(config-router)# network 172.16.0.0

R2(config)# router eigrp 1
R2(config-router)# no auto-summary
R2(config-router)# network 172.16.0.0

R3(config)# router eigrp 1
R3(config-router)# no auto-summary
R3(config-router)# network 172.16.0.0
```

Verify that the number of packets counted is increasing on the outbound interface of R3. Use the **show interfaces fastethernet 0/1** command**.** Issue the command twice to make sure the number of packets output has changed. If the number is not increasing, troubleshoot Layers 1, 2, and 3 connectivity and the EIGRP topology.

Step 3: Configure NBAR Protocol Discovery

NBAR is an IOS QoS feature that allows QoS decisions to be made based on individual protocols. Access control lists (ACLs) can be used to classify traffic based on headers for Layers 1 through 4 of the OSI model. NBAR, on the other hand, allows classification based on the upper layers of the OSI model—Layers 4 through 7. Since it does not rely on TCP/UDP port numbers at Layer 4, it can be used to identify traffic from applications that have dynamic port assignments. One standard feature of NBAR, known as protocol discovery, allows you to dynamically learn which application protocols are in use on your network. NBAR Protocol Discovery can also record and display the most used protocols.

For this lab, configure NBAR Protocol Discovery on the Fast Ethernet 0/0 interface on R1. The only IP traffic leaving the interface will be EIGRP Hello packets, so the majority of packets you should expect to see will be in the inbound direction. The protocols that protocol discovery shows heavy inbound traffic for are the protocols that traffic generation was configured for. To enable protocol discovery, use the interface-level command **ip nbar protocol-discovery**.

```
R1(config)# interface fastethernet0/0
R1(config-if)# ip nbar protocol-discovery
```

After protocol discovery has been enabled for a minute or two, you can see the information it has collected by using the command **show ip nbar protocol-discovery**. This command displays statistics globally for every interface in which NBAR protocol discovery is enabled. The protocols will be ranked based on traffic usage per interface. Notice that ingress and egress traffic is separated as it is in the output of the **show interfaces** command.

```
R1# show ip nbar protocol-discovery

 FastEthernet0/0
                            Input                     Output
                            -----                     ------
   Protocol                 Packet Count              Packet Count
                            Byte Count                Byte Count
                            5min Bit Rate (bps)       5min Bit Rate (bps)
                            5min Max Bit Rate (bps)   5min Max Bit Rate (bps)
   ------------------------ ------------------------  ------------------------
   ssh                      47691                     0
                            37214753                  0
                            800000                    0
                            800000                    0
   xwindows                 46638                     0
                            36235048                  0
                            797000                    0
                            797000                    0
   pop3                     47549                     0
                            37165341                  0
                            796000                    0
                            796000                    0
   smtp                     47112                     0
                            36874672                  0
```



```
                              794000                    0
                              794000                    0
   http                       47099                     0
                              36687939                  0
                              791000                    0
                              791000                    0
   ntp                        44401                     0
                              34670597                  0
                              770000                    0
                              770000                    0
   ftp                        45142                     0
                              35185881                  0
                              767000                    0
                              767000                    0
   telnet                     44322                     0
                              34652510                  0
                              762000                    0
                              762000                    0
   eigrp                      0                         17
                              0                         1258
                              0                         0
                              0                         0
<OUTPUT OMITTED>
```

NBAR uses a preconfigured set of port numbers, which it references during protocol discovery and normal classification operation. Issue the **show ip nbar port-map** command to view the protocol-to-port mappings. This command can also come in handy if you need to find out a well-known port number for an application and do not have access to outside resources. Existing protocol mappings can be modified and custom protocols can be defined, but those NBAR features are outside of the scope of this lab.

```
R1# show ip nbar port-map
port-map bgp                      udp 179
port-map bgp                      tcp 179
port-map bittorrent               tcp 6881 6882 6883 6884 6885 6886 6887 6888
6889
port-map citrix                   udp 1604
port-map citrix                   tcp 1494
port-map cuseeme                  udp 7648 7649 24032
port-map cuseeme                  tcp 7648 7649
port-map dhcp                     udp 67 68
port-map directconnect            tcp 411 412 413
port-map dns                      udp 53
port-map dns                      tcp 53
port-map edonkey                  tcp 4662
port-map exchange                 tcp 135
port-map fasttrack                tcp 1214
port-map finger                   tcp 79
port-map ftp                      tcp 21
port-map gnutella                 udp 6346 6347 6348
port-map gnutella                 tcp 6346 6347 6348 6349 6355 5634
port-map gopher                   udp 70
port-map gopher                   tcp 70
port-map h323                     udp 1300 1718 1719 1720 11720
port-map h323                     tcp 1300 1718 1719 1720 11000 - 11999
<OUTPUT OMITTED>
```

According to best QoS practices, where should packets be marked?

Mark as close as possible to the source, but not so close to the edge that the marking is made on an untrusted device.

What is a trust boundary in terms of classification and marking?

A trust boundary is a delineation of where markings will be honored and where they will not.

Step 4: Classify and Mark Packets

The Modular QoS CLI (MQC) allows someone to create QoS policies on a router in a modular and easy-to-understand format. When creating QoS policies using MQC, there are normally three configuration tasks:

1. Define traffic classes and the method of classification. Classes of traffic are defined in class maps using match statements. The match criterion can be an access list, NBAR-recognized protocol, QoS marking, packet size, and so forth.

2. Create a QoS policy to provision network resources for any traffic classes created in Step 1. A QoS policy maps QoS actions, such as marking, queuing, shaping, policing, or compression, to selected classes.

3. Finally, the policy is applied to an interface directionally, in either the inbound or outbound direction.

Certain policy-map commands can only be applied in a specific direction. For instance, queuing strategies can only be applied in the outbound policies. The router sends an error message to the console if a queuing policy is applied to an interface in the inbound direction, because this is an impossible configuration option.

On R1, you will create a QoS policy to mark an IP Precedence based on the application-layer protocol of the packets. The 3-bit IP Precedence field is part of the legacy Type of Service (ToS) byte on IP packets. Internet standards later converted this byte to the differentiated services (DiffServ) byte which contained the 6-bit differentiated services code point (DSCP) field. The three bits of the IP Precedence field map to the three high-order bits of the DSCP field for backwards-compatibility. For instance, WFQ does not look at the three low-order bits in the DSCP field, but does set weights for each flow based on the three high-order bits of the ToS/DS byte that are used for the IP Precedence

You will apply this QoS policy outbound on R1's Serial 0/0/0 interface.

Begin by implementing the first task: classification. Create traffic classes using NBAR for protocol recognition.

Class-maps are defined with the global configuration command **class-map** [*match-type*] *name*. The optional *match-type* argument can be set to either **match-any** or the default, **match-all**. This argument defines whether all of the successive match statements must be met in order for traffic to be classified into this class, or if only one is necessary.

Once in the class-map configuration mode, matching criteria can be defined with the **match** *criteria* command. To view all the possibilities of what can be matched on, use the **?** command. Choose to use NBAR for classification using the **match protocol** *name* command.

Create three traffic classes:

Critical: EIGRP or Network Time Protocol (NTP) traffic. These protocols are used for network control.

Interactive: Telnet, SSH, and XWindows traffic. These protocols are used for remote administration.

Web: HTTP, POP3, and SMTP traffic. These protocols are used for web and email access.

When creating these traffic classes, should you use the **match-any** or the **match-all** keyword?

You should employ the **match-any** keyword so that more than one protocol can be selected for each traffic class.

The classes created must match with the match-any mode so that any of the protocols listed can be matched. Obviously, it would be impossible for a packet to be two protocols at once.

```
R1(config)# class-map match-any critical
R1(config-cmap)# match ?
  access-group         Access group
  any                  Any packets
  class-map            Class map
  cos                  IEEE 802.1Q/ISL class of service/user priority values
  destination-address  Destination address
  discard-class        Discard behavior identifier
  dscp                 Match DSCP in IP(v4) and IPv6 packets
  flow                 Flow based QoS parameters
  fr-de                Match on Frame-relay DE bit
  fr-dlci              Match on fr-dlci
  input-interface      Select an input interface to match
  ip                   IP specific values
  mpls                 Multi Protocol Label Switching specific values
```



```
  not                  Negate this match result
  packet               Layer 3 Packet length
  precedence           Match Precedence in IP(v4) and IPv6 packets
  protocol             Protocol
  qos-group            Qos-group
  source-address       Source address
  vlan                 VLANs to match
R1(config-cmap)# match protocol eigrp
R1(config-cmap)# match protocol ntp
R1(config-cmap)# class-map match-any interactive
R1(config-cmap)# match protocol telnet
R1(config-cmap)# match protocol ssh
R1(config-cmap)# match protocol xwindows
R1(config-cmap)# class-map match-any web
R1(config-cmap)# match protocol http
R1(config-cmap)# match protocol pop3
R1(config-cmap)# match protocol smtp
```

You can verify created class-maps with the command **show class-map**.

```
R1# show class-map
 Class Map match-any critical (id 1)
   Match protocol eigrp
   Match protocol ntp

 Class Map match-any class-default (id 0)
   Match any

 Class Map match-any interactive (id 2)
   Match protocol telnet
   Match protocol ssh
   Match protocol xwindows

 Class Map match-any web (id 3)
   Match protocol http
   Match protocol pop3
   Match protocol smtp
```

The next task will be to define the QoS policy in a policy map. Create a policy map in global configuration mode using the **policy-map** *name* command. Segment the policy map by traffic class by issuing the **class** *name* command. The names of the classes will be the same as the class maps you created above. Additionally, there is the built-in class “class-default,” which matches any traffic not included in any other class.

```
R1(config)# policy-map markingpolicy
```

At the class configuration prompt, you can use various commands that will affect traffic of that class (use **?** to see what is available). To modify packets, use the command **set** *property value*. Create a new policy named “markingpolicy” and set the IP Precedence for matched packets as follows:

Critical: Set the IP Precedence to Network Control, represented by the value 7.

Interactive: Set the IP Precedence to Critical, represented by the value 5.

Web: Set the IP Precedence to Flash, represented by the value 3.

All other traffic: Set the IP Precedence of all other traffic to Routine, represented by the value 0. This value is the default value for IP Precedence.

There are different names for each value (these can be found out with the **?** command, and this is shown in the following output for reference).

```
R1(config-pmap)# class critical
R1(config-pmap-c)# set precedence ?
  <0-7>           Precedence value
  cos             Set packet precedence from L2 COS
  critical        Set packets with critical precedence (5)
  flash           Set packets with flash precedence (3)
  flash-override  Set packets with flash override precedence (4)
  immediate       Set packets with immediate precedence (2)
  internet        Set packets with internetwork control precedence (6)
  network         Set packets with network control precedence (7)
  priority        Set packets with priority precedence (1)
  qos-group       Set packet precedence from QoS Group.
  routine         Set packets with routine precedence (0)

R1(config-pmap-c)# set precedence 7
R1(config-pmap-c)# class interactive
R1(config-pmap-c)# set precedence 5
R1(config-pmap-c)# class web
R1(config-pmap-c)# set precedence 3
R1(config-pmap-c)# class class-default
R1(config-pmap-c)# set precedence 0
```

Verify the policy map configuration using the **show policy-map** command.

```
R1# show policy-map
  Policy Map markingpolicy
    Class critical
      set precedence 7
    Class interactive
      set precedence 5
    Class web
      set precedence 3
    Class class-default
      set precedence 1
```

Finally, apply the configuration outbound towards R2 with the interface-level command **service-policy** *direction name*.

```
R1(config)# interface serial 0/0/0
R1(config-if)# service-policy output markingpolicy
```

Once a policy map is applied to an interface, you can use an extended form of the **show policy-map** command by issuing the **show policy-map interface** *interface-name* command. This will give you detailed information and statistics on policy maps applied to an interface.

```
R1# show policy-map interface serial0/0/0
 Serial0/0/0

  Service-policy output: markingpolicy
```



```
Class-map: critical (match-any)
  13822 packets, 10617832 bytes
  5 minute offered rate 264000 bps, drop rate 0 bps
  Match: protocol eigrp
    5 packets, 320 bytes
    5 minute rate 0 bps
  Match: protocol ntp
    13817 packets, 10617512 bytes
    5 minute rate 264000 bps
  QoS Set
    precedence 7
      Packets marked 13822

Class-map: interactive (match-any)
  44974 packets, 34630670 bytes
  5 minute offered rate 830000 bps, drop rate 0 bps
  Match: protocol telnet
    15300 packets, 11765411 bytes
    5 minute rate 289000 bps
  Match: protocol ssh
    14451 packets, 11209788 bytes
    5 minute rate 270000 bps
  Match: protocol xwindows
    15223 packets, 11655471 bytes
    5 minute rate 282000 bps
  QoS Set
    precedence 5
      Packets marked 44984

Class-map: web (match-any)
  44600 packets, 34404320 bytes
  5 minute offered rate 857000 bps, drop rate 0 bps
  Match: protocol http
    13688 packets, 10530109 bytes
    5 minute rate 269000 bps
  Match: protocol pop3
    14513 packets, 11240708 bytes
    5 minute rate 290000 bps
  Match: protocol smtp
    16399 packets, 12633503 bytes
    5 minute rate 312000 bps
  QoS Set
    precedence 3
      Packets marked 44620

Class-map: class-default (match-any)
  13745 packets, 10547088 bytes
  5 minute offered rate 261000 bps, drop rate 0 bps
  Match: any
  QoS Set
    precedence 0
      Packets marked 13743
```

If a BGP packet with an IP precedence marking of 3 enters the Fast Ethernet 0/0 interface on R1 and is destined for R2, into which traffic class will the packet be classified?

Into the class-default class.

What IP precedence will the packet be assigned at the egress port?

The BGP packet will be assigned IP Precedence 0 for Routine Traffic.

Step 5: Shape Traffic and Queue with CBWFQ and LLQ

One of the QoS actions that can be performed in a policy map is shaping. Shaping limits traffic for a traffic class to a specific rate and buffers excess traffic. Policing, a related concept drops the excess traffic. Thus, the purpose of shaping is to buffer traffic so that more traffic is sent than if you policed at the same rate because not only will the traffic conforming to the policy be sent, but also buffered excess traffic when permitted.

Policing and shaping can each be configured within a policy map as a QoS action for a specific traffic class, or you can nest policy maps to create an aggregate shaper or policer. Multiple QoS actions can be taken on a specific class of traffic so you could use shaping in conjunction with marking or compression, or various other actions. Keep this in mind for the remaining labs

The first task in creating the QoS policy is to enumerate classes. This time, use uncreative names such as “prec7” and “prec5” for packets with IP Precedences 7 and 5, respectively. Create classes like this for IP Precedences 0, 3, 5, and 7—the in Module 4.

In this circumstance, however, you will view the class-based shapers in conjunction with low-latency queuing (LLQ). There are two class-based queuing tools, class-based weighted fair queuing (CBWFQ) and low-latency queuing (LLQ). CBWFQ is similar to custom queuing (CQ) in that it provisions an average amount or percent of bandwidth to a traffic class. However, the classification mechanism in class-based tools is much more powerful because it can also use NBAR to discover application protocols and even application protocol parameters, such as the URL in an HTTP request. LLQ is a simple improvement on CBWFQ, adding the ability to designate some classes as priority traffic and ensure that they are sent before others.

On R2, create a policy map to be applied on its Serial 0/0/1 interface. This policy map will be used to shape traffic based on markings by R1.possibilities for marking from the last step. To match on IP Precedence in a class definition, use the **match precedence** *precedence* command, where the *precedence* argument is the value or representative name. You must reclassify and mark EIGRP packets because each of the EIGRP packets is link-local traffic and the EIGRP packets which you marked on ingress at R1 were not sent to R2. The new packets for the link between R1 and R2 must now be classified by an access list or NBAR. However, any NTP packets traversing the link will already be marked with IP precedence 7. You should to treat EIGRP and NTP packets in the same traffic class for consistency.

Would you use the **match-all** or **match-any** keyword when creating the “prec7” class map? Explain.

You would use the **match-any** keyword so that you can match EIGRP traffic based on NBAR and NTP traffic based on IP Precedence 7.

Create the class map as follows.

```
R2(config)# class-map prec0
R2(config-cmap)# match precedence 0
R2(config-cmap)# class-map prec3
R2(config-cmap)# match precedence 3
R2(config-cmap)# class-map prec5
R2(config-cmap)# match precedence 5
R2(config-cmap)# class-map match-any prec7
R2(config-cmap)# match precedence 7
R2(config-cmap)# match protocol eigrp
```

Next, create the QoS policy to shape and queue the traffic. The syntax for entering the policy map and per-class configuration will be the same as above. However, rather than changing packet properties, we will set up low-latency queuing (LLQ) for the interface. LLQ is a variant of class-based weighted fair queuing (CBWFQ). Configuring CBWFQ involves assigning each traffic class dedicated bandwidth, either through exact bandwidth amounts or relative percentage amounts. LLQ is the configured the same way, except that one or more traffic classes are designated as priority traffic and assigned to an expedite queue. All traffic that enters the expedite queue up to the bandwidth limit will be sent as soon as possible, preempting traffic from non-priority classes.

While you configure either CBWFQ or LLQ, you can allocate a certain bandwidth for a traffic class, using the **bandwidth** *rate* command, where *rate* is a bandwidth amount in kilobits per second. Alternatively, use the **bandwidth percentage** *percent* command to allocate a percentage of bandwidth, where 100 percent of the bandwidth is set by the informational bandwidth parameter that you configured in Step 1.

For LLQ solely, issue the **priority** *rate* command or the **priority percentage** *percent* command in policy map configuration mode. These commands have the same arguments, which have the same effect as the **bandwidth** commands, except that they designate that queue as the priority queue.

Create a policy named “llqpolicy” on R2. The policy should allocate 10 percent of traffic to the “prec7” traffic class, 15 percent to the “prec5” traffic class, 30 percent to the “prec3” traffic class, and 20 percent to the “prec0” traffic class. Expedite traffic that falls into the “prec7” traffic class. Also, select weighted fair-queuing as the queuing method in the default traffic class with the **fair-queue** command.


```
R2(config)# policy-map llqpolicy
R2(config-pmap)# class prec7
R2(config-pmap-c)# priority percent 10
R2(config-pmap-c)# class prec5
R2(config-pmap-c)# bandwidth percent 15
R2(config-pmap-c)# class prec3
R2(config-pmap-c)# bandwidth percent 30
R2(config-pmap-c)# class prec0
R2(config-pmap-c)# bandwidth percent 20
R2(config-pmap-c)# class class-default
R2(config-pmap-c)# fair-queue
```

Verify your QoS policy configuration using the **show policy-map** command. Notice that the priority queue is a variant on the regular queues.

```
R2# show policy-map
  Policy Map llqpolicy
    Class prec7
      Strict Priority
      Bandwidth 10 (%)
    Class prec5
      Bandwidth 15 (%) Max Threshold 64 (packets)
    Class prec3
      Bandwidth 30 (%) Max Threshold 64 (packets)
    Class prec0
      Bandwidth 20 (%) Max Threshold 64 (packets)
    Class class-default
      Flow based Fair Queueing
      Bandwidth 0 (kbps) Max Threshold 64 (packets)
```

What traffic types would usually belong in a priority queue in a production environment?

Routing protocol traffic belongs in a priority queue so that adjacencies do not get lost. Any delay-sensitive traffic, such as Voice over IP (VoIP) or interactive video traffic, also belongs in a priority queue.

Use the same **service-policy** command from earlier to apply this policy map to the Serial 0/0/1 interface on R2 in an outbound direction.

```
R2(config)# interface serial 0/0/1
R2(config-if)# service-policy output llqpolicy
```

Verify using the interface-specific version of **show policy-map**.

```
R2# show policy-map interface serial0/0/1
 Serial0/0/1

  Service-policy output: llqpolicy

    Class-map: prec7 (match-any)
      3995 packets, 3387767 bytes
      5 minute offered rate 81000 bps, drop rate 80000 bps
      Match:  precedence 7
        3941 packets, 3384319 bytes
```



```
    5 minute rate 81000 bps
  Match: protocol eigrp
    54 packets, 3448 bytes
    5 minute rate 0 bps
  Queueing
    Strict Priority
    Output Queue: Conversation 40
    Bandwidth 10 (%)
    Bandwidth 12 (kbps) Burst 300 (Bytes)
    (pkts matched/bytes matched) 3947/3384695
    (total drops/bytes drops) 3524/3314514

Class-map: prec5 (match-all)
  8378 packets, 7165609 bytes
  5 minute offered rate 165000 bps, drop rate 146000 bps
  Match:  precedence 5
  Queueing
    Output Queue: Conversation 41
    Bandwidth 15 (%)
    Bandwidth 19 (kbps)Max Threshold 64 (packets)
    (pkts matched/bytes matched) 8378/7165609
    (depth/total drops/no-buffer drops) 64/7459/0

Class-map: prec3 (match-all)
  10295 packets, 8813462 bytes
  5 minute offered rate 197000 bps, drop rate 163000 bps
  Match:  precedence 3
  Queueing
    Output Queue: Conversation 42
    Bandwidth 30 (%)
    Bandwidth 38 (kbps)Max Threshold 64 (packets)
    (pkts matched/bytes matched) 10293/8810571
    (depth/total drops/no-buffer drops) 64/8500/0

Class-map: prec0 (match-all)
  3239 packets, 2830395 bytes
  5 minute offered rate 73000 bps, drop rate 52000 bps
  Match:  precedence 0
  Queueing
    Output Queue: Conversation 43
    Bandwidth 20 (%)
    Bandwidth 25 (kbps)Max Threshold 64 (packets)
    (pkts matched/bytes matched) 3239/2830395
    (depth/total drops/no-buffer drops) 60/1988/0

Class-map: class-default (match-any)
  26 packets, 1524 bytes
  5 minute offered rate 0 bps, drop rate 0 bps
  Match: any
  Queueing
    Flow Based Fair Queueing
    Maximum Number of Hashed Queues 32
    (total queued/total drops/no-buffer drops) 0/0/0
```

Challenge: Verifying IP Precedence

The topic of IP accounting is outside the scope of this curriculum. However, it is a useful tool for the verification of a marking policy. Issue the **ip accounting precedence** *direction* command in interface configuration mode to enable IP accounting on an interface. Apply this command on R3 for the Serial 0/0/1

interface that shows incoming markings from R2. View the accounting records for IP precedence by issuing the **show interfaces precedence** command.

```
R3(config)# interface serial0/0/1
R3(config-if)# ip accounting precedence input

R3# show interface precedence
Serial0/0/1
  Input
    Precedence 0:  10 packets, 5121 bytes
    Precedence 1:  230 packets, 85385 bytes
    Precedence 3:  193 packets, 127000 bytes
    Precedence 5:  88 packets, 62727 bytes
    Precedence 6:  5 packets, 320 bytes
    Precedence 7:  148 packets, 16984 bytes
```

Can you think of another simple way to count packets with each IP Precedence marking? You do not need to actually implement it. HINT: Think access lists.

You can create an extended access list with a permit statement for each IP Precedence, and then use **show access-lists** to look at the counters for each line. This is shown in the following output.

```
R3(config)# access-list 100 permit ip any any precedence 0
R3(config)# access-list 100 permit ip any any precedence 1
R3(config)# access-list 100 permit ip any any precedence 2
R3(config)# access-list 100 permit ip any any precedence 3
R3(config)# access-list 100 permit ip any any precedence 4
R3(config)# access-list 100 permit ip any any precedence 5
R3(config)# access-list 100 permit ip any any precedence 6
R3(config)# access-list 100 permit ip any any precedence 7
R3(config)# interface serial 0/0/1
R3(config-if)# ip access-group 100 in

R3# show access-list
Extended IP access list 100
    10 permit ip any any precedence routine
    20 permit ip any any precedence priority (88 matches)
    30 permit ip any any precedence immediate
    40 permit ip any any precedence flash (99 matches)
    50 permit ip any any precedence flash-override
    60 permit ip any any precedence critical (48 matches)
    70 permit ip any any precedence internet (9 matches)
    80 permit ip any any precedence network (74 matches)
```

Final Configurations

```
R1# show run
hostname R1
!
class-map match-any critical
 match protocol eigrp
 match protocol ntp
class-map match-any interactive
 match protocol telnet
 match protocol ssh
```



```
 match protocol xwindows
class-map match-any web
 match protocol http
 match protocol pop3
 match protocol smtp
!
policy-map markingpolicy
 class critical
  set precedence 7
 class interactive
  set precedence 5
 class web
  set precedence 3
 class class-default
  set precedence 0
!
interface FastEthernet0/0
 ip address 172.16.10.1 255.255.255.0
 ip nbar protocol-discovery
 no shutdown
!
interface Serial0/0/0
 ip address 172.16.12.1 255.255.255.0
 clock rate 800000
 service-policy output markingpolicy
 no shutdown
!
router eigrp 1
 network 172.16.0.0
 no auto-summary
end

R2# show run
hostname R2
!
class-map match-all prec5
 match  precedence 5
class-map match-any prec7
 match  precedence 7
 match protocol eigrp
class-map match-all prec0
 match  precedence 0
class-map match-all prec3
 match  precedence 3
!
policy-map llqpolicy
 class prec7
  priority percent 10
 class prec5
  bandwidth percent 15
 class prec3
  bandwidth percent 30
 class prec0
  bandwidth percent 20
 class class-default
  fair-queue
!
interface Serial0/0/0
 ip address 172.16.12.2 255.255.255.0
 no shutdown
!
interface Serial0/0/1
 bandwidth 128
```



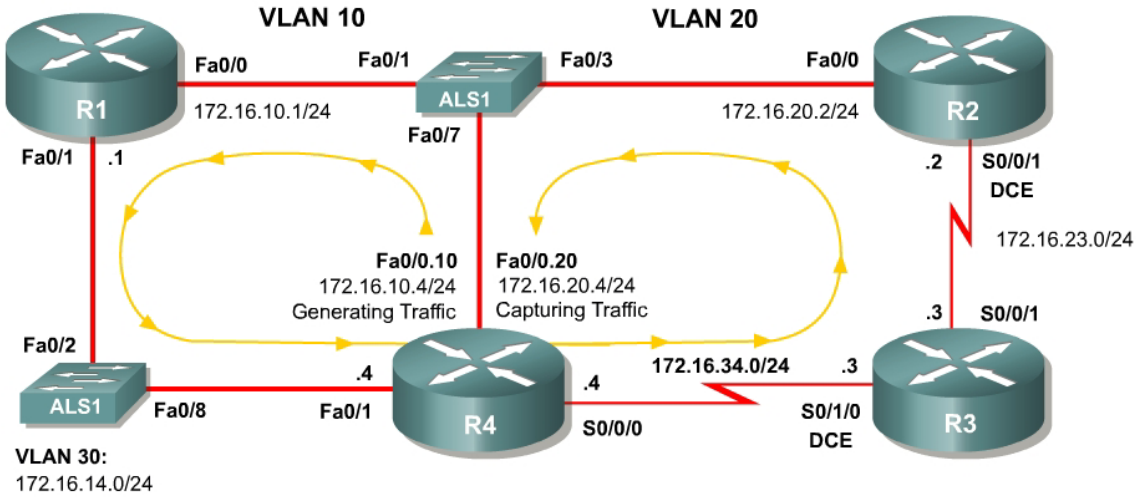
```
 ip address 172.16.23.2 255.255.255.0
 clock rate 128000
 service-policy output llqpolicy
 no shutdown
!
router eigrp 1
 network 172.16.0.0
 no auto-summary
!
end

R3# show run
hostname R3
!
interface FastEthernet0/1
 ip address 172.16.20.3 255.255.255.0
 no shutdown
!
interface Serial0/0/1
 ip address 172.16.23.3 255.255.255.0
 no shutdown
!
router eigrp 1
 network 172.16.0.0
 no auto-summary
end
```


Lab 4.6 Class-based Marking, Shaping, and Policing

Learning Objectives

- Mark packets with DSCP values
- Implement class-based TCP Header Compression
- Configure class-based traffic shaping and policing
- Create and apply nested service policies

Topology Diagram



Scenario

In this lab, you will implement classification using network-based application recognition (NBAR) using the Modular QoS CLI (MQC) to configure quality of service on R1 and R2. You will configure class-based marking, shaping, and policing mechanisms.

You should complete Lab 4.5 before beginning this lab because this lab will build on the concepts of NBAR and marking that you configured in that scenario.

Preparation

This lab relies on the Advanced Pagent Configuration, which you should have created in Lab 3.2: Preparing for QoS.

Prior to beginning this lab, configure R4 and the switch according to the Advanced Pagent Configuration. You may easily accomplish this on R4 by

loading the *advanced-ios.cfg* file from flash memory into the NVRAM, and reloading.

```
TrafGen# copy flash:advanced-ios.cfg startup-config
Destination filename [startup-config]?
[OK]
2875 bytes copied in 1.456 secs (1975 bytes/sec)
TrafGen# reload
Proceed with reload? [confirm]
```

On the switch, load the *advanced.cfg* file into NVRAM and reload the device.

```
ALS1# copy flash:advanced.cfg startup-config
Destination filename [startup-config]?
[OK]
2875 bytes copied in 1.456 secs (1975 bytes/sec)
ALS1# reload
Proceed with reload? [confirm]
```

Next, instruct TGN to load the *advanced-tgn.cfg* file. At the end of Step 1, you will begin generating TGN traffic.

```
TrafGen# tgn load-config advanced-tgn.cfg
```

Step 1: Configure the Physical Interfaces

Configure all of the physical interfaces shown in the diagram. Set the clock rate on both serial links to 800000 bits per second and use the **no shutdown** command on all necessary interfaces. Set the informational bandwidth parameter appropriately on the serial interfaces.

```
R1(config)# interface fastethernet 0/0
R1(config-if)# ip address 172.16.10.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# interface fastethernet 0/1
R1(config-if)# ip address 172.16.14.1 255.255.255.0
R1(config-if)# no shutdown

R2(config)# interface serial 0/0/1
R2(config-if)# bandwidth 800
R2(config-if)# ip address 172.16.23.2 255.255.255.0
R2(config-if)# clockrate 800000
R2(config-if)# no shutdown
R2(config-if)# interface fastethernet 0/0
R2(config-if)# ip address 172.16.20.2 255.255.255.0
R2(config-if)# no shutdown

R3(config)# interface serial 0/0/1
R3(config-if)# bandwidth 800
R3(config-if)# ip address 172.16.23.3 255.255.255.0
R3(config-if)# no shutdown
R3(config-if)# interface serial 0/1/0
R3(config-if)# bandwidth 800
R3(config-if)# ip address 172.16.34.3 255.255.255.0
R3(config-if)# clockrate 800000
R3(config-if)# no shutdown

R4(config)# interface fastethernet 0/1
R4(config-if)# ip address 172.16.14.4 255.255.255.0
```



```
R4(config-if)# no shutdown
R4(config-if)# interface serial 0/0/0
R3(config-if)# bandwidth 800
R4(config-if)# ip address 172.16.34.4 255.255.255.0
R4(config-if)# no shutdown
```

Now that R4 can reach R1 172.16.10.1 address via ARP, begin generating TGN traffic.

```
TrafGen# tgn start
```

Step 2: Configure Routing

Establish adjacencies for routing with Open Shortest Path First (OSPF). Include all connected subnets within the 172.16.0.0/16 major network for all four routers.

```
R1(config)# router ospf 1
R1(config-router)# network 172.16.0.0 0.0.255.255 area 0

R2(config)# router ospf 1
R2(config-router)# network 172.16.0.0 0.0.255.255 area 0

R3(config)# router ospf 1
R3(config-router)# network 172.16.0.0 0.0.255.255 area 0

R4(config)# router ospf 1
R4(config-router)# network 172.16.0.0 0.0.255.255 area 0
```

Step 3: Mark Packets with DSCP

Various Internet Engineering Task Force Request for Comments (IETF RFCs) have outlined a set of quality of service (QoS) per-hop behaviors (PHBs). These RFCs define a marking scheme as well as a set of actions or preferences to be followed at each hop as that data packet traverses the routed path. These RFCs build on the redefinition of the markable byte in the IP header from type of service (ToS) to differentiated services (DiffServ). These standardized PHBs define marking scheme to set six bits in the DiffServ Code Point (DSCP) field.

According to the PHB RFCs, a DSCP marking is slightly different than IP Precedence, in that it includes the queuing treatment and drop probability. Since the DiffServ byte overlaps the legacy ToS byte in an IP packet, DSCP values are backwards-compatible in networks or QoS tools that rely solely on IP Precedence. You can mark IP packets with two different types of DSCP markings: Expedited Forwarding (EF) for priority traffic (such as voice packets), and Assured Forwarding (AF). Simply marking traffic correctly does not configure the QoS tools to implement the various PHBs. However, markings with standardized meanings can drastically improve the understanding of QoS in a network.

There are no classes of EF traffic, but the RFCs define multiple classes within the AF marking. The names for the AF classes follow the pattern AF*xy*, where *x*

and *y* are each small integral numbers. The *x* value represents the traffic class, while the *y* value represents the drop probability within that traffic class. There are four defined traffic classes numbered 1 through 4 and three drop priorities numbered 1 through 3. The larger the drop priority, the more likely the packet is to be dropped. For instance, you can configure weighted random early detection (WRED) to drop packets based on DSCP values.

For this scenario, R1 will classify via NBAR and mark packets with the EF and AF DSCP markings. All QoS actions will be performed within the MQC, so you will need to create traffic classes on each router. For more information on NBAR or MQC, consult the Lab 4.5: Class-based Queuing and NBAR.

To set a DSCP value, use the policy-map class configuration sub-prompt command **set dscp** *value*. Notice the available values shown in the output below.

```
R1(config-pmap-c)# set dscp ?
  <0-63>     Differentiated services codepoint value
  af11       Match packets with AF11 dscp (001010)
  af12       Match packets with AF12 dscp (001100)
  af13       Match packets with AF13 dscp (001110)
  af21       Match packets with AF21 dscp (010010)
  af22       Match packets with AF22 dscp (010100)
  af23       Match packets with AF23 dscp (010110)
  af31       Match packets with AF31 dscp (011010)
  af32       Match packets with AF32 dscp (011100)
  af33       Match packets with AF33 dscp (011110)
  af41       Match packets with AF41 dscp (100010)
  af42       Match packets with AF42 dscp (100100)
  af43       Match packets with AF43 dscp (100110)
  cos        Set packet DSCP from L2 COS
  cs1        Match packets with CS1(precedence 1) dscp (001000)
  cs2        Match packets with CS2(precedence 2) dscp (010000)
  cs3        Match packets with CS3(precedence 3) dscp (011000)
  cs4        Match packets with CS4(precedence 4) dscp (100000)
  cs5        Match packets with CS5(precedence 5) dscp (101000)
  cs6        Match packets with CS6(precedence 6) dscp (110000)
  cs7        Match packets with CS7(precedence 7) dscp (111000)
  default    Match packets with default dscp (000000)
  ef         Match packets with EF dscp (101110)
  qos-group  Set packet dscp from QoS Group.
```

Classify traffic on R1 as follows:

Create three traffic classes:

Critical: OSPF or Network Time Protocol (NTP) traffic. These protocols are used for network control. Mark with DSCP value EF.

Interactive: Telnet, SSH, and X-Windows traffic. These protocols are used for remote administration. Mark with DSCP value AF41.

Web: HTTP, POP3, and SMTP traffic. These protocols are used for web and e-mail access. Mark with DSCP value AF32.


```
R1(config)# class-map match-any critical
R1(config-cmap)# match protocol ospf
R1(config-cmap)# match protocol ntp
R1(config-cmap)# class-map match-any interactive
R1(config-cmap)# match protocol telnet
R1(config-cmap)# match protocol ssh
R1(config-cmap)# match protocol xwindows
R1(config-cmap)# class-map match-any web
R1(config-cmap)# match protocol http
R1(config-cmap)# match protocol pop3
R1(config-cmap)# match protocol smtp
```

Mark all other traffic with the default DSCP of 0.

Create the QoS policy map named “markingpolicy” and apply it outbound towards R4 on the Fast Ethernet 0/1 interface.

```
R1(config)# policy-map markingpolicy
R1(config-pmap)# class critical
R1(config-pmap-c)# set dscp ef
R1(config-pmap-c)# class interactive
R1(config-pmap-c)# set dscp af41
R1(config-pmap-c)# class web
R1(config-pmap-c)# set dscp af32
R1(config-pmap-c)# class class-default
R1(config-pmap-c)# set dscp default
R1(config-pmap-c)# interface fastethernet0/1
R1(config-if)# service-policy output markingpolicy
```

Verify the QoS configuration with the **show policy-map** command. Also, verify that the marking strategy is actively marking traffic with the **show policy-map interface** *interface* command.

```
R1# show policy-map
  Policy Map markingpolicy
    Class critical
      set dscp ef
    Class interactive
      set dscp af41
    Class web
      set dscp af32
    Class class-default
      set dscp default

R1# show policy-map interface fastethernet0/1
 FastEthernet0/1

  Service-policy output: markingpolicy

    Class-map: critical (match-any)
      242695 packets, 186052247 bytes
      5 minute offered rate 2475000 bps, drop rate 0 bps
      Match: protocol ospf
        108 packets, 7992 bytes
        5 minute rate 0 bps
      Match: protocol ntp
        242587 packets, 186044255 bytes
        5 minute rate 2475000 bps
      QoS Set
        dscp ef
```



```
          Packets marked 242695
<OUTPUT OMITTED>
```

Why would a network administrator decide to use IP Precedence over DSCP, or vice-versa?

IP Precedence is simpler and more straightforward with only 3 bits. DSCP is more powerful because of the granularity it allows. Which one is better depends on the traffic profile (diversity of traffic) and which one the administrator feels is more appropriate for their network.

Step 4: Configuring Class-Based Shaping

Traffic shaping is a QoS tool that allows you to define an average or peak rate at which traffic will be sent at an egress interface. Excess traffic is queued for sending later.

Observe the following rules when shaping or policing traffic:

1. At OSI Layer 1, data can only be sent at the clock rate (access rate) of the medium.

2. At OSI Layer 2, frames can be sent to approximate variable rates up to the Layer 1 clock rate by interchanging sending frames and restricting the sending of frames. In other words, traffic must be sent in bursts of data at exactly the access rate within each time interval to shape or police traffic at a specific rate.

Shaping and policing allow you to either allow the Cisco IOS to determine the amount of traffic to send within each time interval or to specify the number of bytes in the **shape** or **police** commands.

Shaping may be configured on a per-interface basis with Generic Traffic Shaping (GTS), or in a per-class basis through the MQC. Additionally, for Frame Relay networks which operate based on the concept of virtual circuits (VCs), Frame Relay Traffic Shaping (FRTS) can even be configured on a per-VC basis. In this scenario, you will use the MQC to configure Class-Based Traffic Shaping (CBTS) and simulate the function of GTS using CBTS in the Step 5.

In this step, shape all traffic traveling from R4 to R3 across the serial link to a peak rate. Create a policy map and classify traffic only into the default class; then shape peak egress rate of the default class on R4. This method of using one traffic class within the policy map to shape traffic can effectively simulate the function of GTS when you apply the policy map to an interface. Configure the peak traffic rate for a class, using the **shape peak** *rate* command. Use a

peak traffic rate of 400 kbps. You can also configure the burst values more granularly, but this is beyond the scope of this lab.

```
R4(config)# policy-map shapingpolicy
R4(config-pmap)# class class-default
R4(config-pmap-c)# shape peak 400000
R4(config-pmap-c)# interface serial0/0/0
R4(config-if)# service-policy output shapingpolicy
```

Verify the configuration using the **show** commands for policy-maps.

```
R4# show policy-map
  Policy Map shapingpolicy
    Class class-default
      Traffic Shaping
         Peak Rate Traffic Shaping
         CIR 400000 (bps) Max. Buffers Limit 1000 (Packets)

R4# show policy-map interface serial0/0/0
 Serial0/0/0

  Service-policy output: shapingpolicy

    Class-map: class-default (match-any)
      546427 packets, 418135512 bytes
      5 minute offered rate 7644000 bps, drop rate 7092000 bps
      Match: any
      Traffic Shaping
           Target/Average   Byte   Sustain   Excess    Interval  Increment
             Rate           Limit  bits/int  bits/int  (ms)      (bytes)
           800000/400000    2500   10000     10000     25        2500

        Adapt  Queue     Packets   Bytes     Packets   Bytes     Shaping
        Active Depth                        Delayed   Delayed   Active
        -      96        46540     24706516  46536     24703845  yes
```

The generated traffic is dense enough to completely saturate the serial link and/or the shaping profile, so you cannot see the function of the burst values; however, you can see that shaping is active and that packets have been delayed in transmission on account of that shaping.

What happens to the DSCP markings on IP packets traversing the serial link from R4 to R3 if no other traffic classes are referenced within the policy map?

They retain their markings as long as the traffic classes that are selected by the policy map do not set the marking values to something else.

Step 5: Configure Nested Service Policies

When you begin to create more complex QoS policies, you may find the need to apply a named policy-map inside of a class in another policy-map. You noted before that only the default class was used in the shaping policy in Step 4.

One possible scenario in which this would be necessary is if you want to apply granularity in marking, queuing, or shaping packets in distinct traffic classes but want to apply an aggregate shaper or policer to all of the traffic exiting the interface. Apply the differentiated actions in a single policy map. Then, set the shaping action in the default class in another policy map and apply the first policy map as an MQC action within the second policy map.

Use the policy map you configured in Step 4 as the outer policy map which will be applied directly to the interface. Create a new policy map to be used inside the outer policy map. Shape the individual classes using the inner policy map and shape the aggregate over all of the traffic classes in the outer policy map.

Create another policy (with appropriate classes) as shown below that shapes EF traffic to 40kbps, AF41 traffic should get 80kpbs, and AF32 traffic should get shaped to 120kbps. Apply this new policy inside the class configuration of the policy created in Step 4 using the **service-policy** *name* command.

```
R4(config)# class-map ef
R4(config-cmap)# match dscp ef
R4(config-cmap)# class-map af41
R4(config-cmap)# match dscp af41
R4(config-cmap)# class-map af32
R4(config-cmap)# match dscp af32
R4(config-cmap)# policy-map innerpolicy
R4(config-pmap)# class ef
R4(config-pmap-c)# shape peak 40000
R4(config-pmap-c)# class af41
R4(config-pmap-c)# shape peak 80000
R4(config-pmap-c)# class af32
R4(config-pmap-c)# shape peak 120000
R4(config-pmap-c)# policy-map shapingpolicy
R4(config-pmap)# class class-default
R4(config-pmap-c)# service-policy innerpolicy
```

Verify with the **show policy-map** command and the **show policy-map interface serial 0/0/0** command.

```
R4# show policy-map
  Policy Map shapingpolicy
    Class class-default
      Traffic Shaping
         Peak Rate Traffic Shaping
         CIR 400000 (bps) Max. Buffers Limit 1000 (Packets)
      service-policy innerpolicy

  Policy Map innerpolicy
    Class ef
      Traffic Shaping
         Peak Rate Traffic Shaping
         CIR 40000 (bps) Max. Buffers Limit 1000 (Packets)
    Class af41
      Traffic Shaping
         Peak Rate Traffic Shaping
         CIR 80000 (bps) Max. Buffers Limit 1000 (Packets)
    Class af32
      Traffic Shaping
         Peak Rate Traffic Shaping
```



```
         CIR 120000 (bps) Max. Buffers Limit 1000 (Packets)

R4# show policy-map interface serial0/0/0
 Serial0/0/0

  Service-policy output: shapingpolicy

    Class-map: class-default (match-any)
      492271 packets, 376494434 bytes
      5 minute offered rate 6900000 bps, drop rate 509000 bps
      Match: any
      Traffic Shaping
           Target/Average   Byte   Sustain   Excess    Interval  Increment
             Rate           Limit  bits/int  bits/int  (ms)      (bytes)
           800000/400000    2500   10000     10000     25        2500

        Adapt  Queue     Packets   Bytes     Packets   Bytes     Shaping
        Active Depth                         Delayed   Delayed   Active
        -      42        24271     17196294  23348     16930349  yes

      Service-policy : innerpolicy

        Class-map: ef (match-all)
          62585 packets, 47610351 bytes
          5 minute offered rate 905000 bps, drop rate 0 bps
          Match:  dscp ef (46)
          Traffic Shaping
               Target/Average   Byte   Sustain   Excess    Interval  Increment
                 Rate           Limit  bits/int  bits/int  (ms)      (bytes)
                80000/40000     2000   8000      8000      200       2000

            Adapt  Queue     Packets   Bytes     Packets   Bytes     Shaping
            Active Depth                         Delayed   Delayed   Active
            -      64        2140      1647406   2135      1644763   yes
<OUTPUT OMITTED>
```

Step 6: Configure Traffic Policing

The difference between shaping traffic and policing traffic is that shapers attempt to smooth out a traffic profile whereas policers merely force the traffic to conform to a certain rate without buffering the excess. Policers drop excess packets and do not carry traffic from one interval to the next.

Create a new policy map to police traffic passing from R3 to R2. Police the total rate of egress traffic exiting R3's Serial 0/0/1 interface to 400 kbps.

Police the default class to the specified rate by issuing the **police rate** *rate type* command. You may also set up more granular parameters for the policer to use by issuing the **?** character.

```
R3(config)# policy-map policingpolicy
R3(config-pmap)# class class-default
R3(config-pmap-c)# police rate 400000 bps
R3(config-pmap-c-police)# interface serial0/0/1
R3(config-if)# service-policy output policingpolicy
```


Verify with the usual commands. Notice that some of the details of policing, such as the burst size, have been set up automatically since we did not specify them.

```
R3# show policy-map
  Policy Map policingpolicy
    Class class-default
     police rate 400000 bps burst 12500 bytes
       conform-action transmit
       exceed-action drop

R3# show policy-map interface serial0/0/1
 Serial0/0/1

  Service-policy output: policingpolicy

    Class-map: class-default (match-any)
      9702 packets, 6764207 bytes
      5 minute offered rate 158000 bps, drop rate 44811000 bps
      Match: any
      police:
          rate 400000 bps, burst 12500 bytes
        conformed 5912 packets, 3113901 bytes; actions:
          transmit
        exceeded 3768 packets, 3648918 bytes; actions:
          drop
        conformed 79000 bps, exceed 89000 bps
```

Step 7: Configure Class-Based TCP Header Compression

In Lab 4.3: Configuring TCP Header Compression, you configured TCP header compression on an entire interface. In the MQC, you can configure TCP and RTP header compression as a QoS action for specific traffic classes.

Issue the **compression header ip** *type* command, where *type* is either the **tcp** or **rtp** keyword. Configure TCP header compression on R4 for only AF32 traffic heading towards R3 using the existing policy-maps. For more information on header compression, consult the Lab 4.3.

```
R4(config)# policy-map innerpolicy
R4(config-pmap)# class af32
R4(config-pmap-c)# compression header ip tcp
```

If this was actual TCP traffic and not spoofed traffic, you would see packets being compressed. Because the TCP headers are not all being created naturally, some elements of the TCP header are incompressible. Notice that in the output of the **show policy-map** command no headers have been compressed. The traffic that is being generated is not legitimate TCP traffic so it will not be compressed.

```
R4# show policy-map interface
  Policy Map shapingpolicy
    Class class-default
      Traffic Shaping
         Peak Rate Traffic Shaping
         CIR 400000 (bps) Max. Buffers Limit 1000 (Packets)
```



```
    service-policy innerpolicy

  Policy Map innerpolicy
    Class ef
      Traffic Shaping
         Average Rate Traffic Shaping
         CIR 40000 (bps) Max. Buffers Limit 1000 (Packets)
    Class af41
      Traffic Shaping
         Average Rate Traffic Shaping
         CIR 80000 (bps) Max. Buffers Limit 1000 (Packets)
    Class af32
      Traffic Shaping
         Average Rate Traffic Shaping
         CIR 120000 (bps) Max. Buffers Limit 1000 (Packets)
      compress:
          header ip tcp
```

How could you create compressible TCP packets given the current topology?

Telnet from R4 to R3 after preparing R3 for telnet access.

Implement your solution and verify that packets are being compressed.

```
R3(config)# line vty 0 4
R3(config-line)# password cisco
R3(config-line)# login

R4# telnet 172.16.34.3
Trying 172.16.34.3 ... Open


User Access Verification

Password: cisco
R3> exit
[Ctrl+Shift+6, x]

R4# show policy-map interface serial 0/0/1
```

Final Configurations

```
R1# show run
!
hostname R1
!
class-map match-any critical
 match protocol ospf
 match protocol ntp
class-map match-any interactive
 match protocol telnet
 match protocol ssh
 match protocol xwindows
class-map match-any web
 match protocol http
 match protocol pop3
 match protocol smtp
```



```
!
policy-map markingpolicy
 class critical
  set dscp ef
 class interactive
  set dscp af41
 class web
  set dscp af32
 class class-default
  set dscp default
!
interface FastEthernet0/0
 ip address 172.16.10.1 255.255.255.0
 no shutdown
!
interface FastEthernet0/1
 ip address 172.16.14.1 255.255.255.0
 service-policy output markingpolicy
 no shutdown
!
router ospf 1
 network 172.16.0.0 0.0.255.255 area 0
!
end

R2# show run
!
hostname R2
!
interface FastEthernet0/0
 ip address 172.16.20.2 255.255.255.0
 no shutdown
!
interface Serial0/0/1
 ip address 172.16.23.2 255.255.255.0
 clock rate 800000
 no shutdown
!
router ospf 1
 network 172.16.0.0 0.0.255.255 area 0
!
end

R3# show run
!
hostname R3
!
policy-map policingpolicy
 class class-default
  police rate 400000 bps
!
interface Serial0/0/1
 ip address 172.16.23.3 255.255.255.0
 service-policy output policingpolicy
 no shutdown
!
interface Serial0/1/0
 ip address 172.16.34.3 255.255.255.0
 clockrate 800000
 no shutdown
!
router ospf 1
 network 172.16.0.0 0.0.255.255 area 0
```



```
!
line vty 0 4
 password cisco
 login
!
end
```

Pagent-related commands are removed from R4's output. Only commands related to this lab are shown.

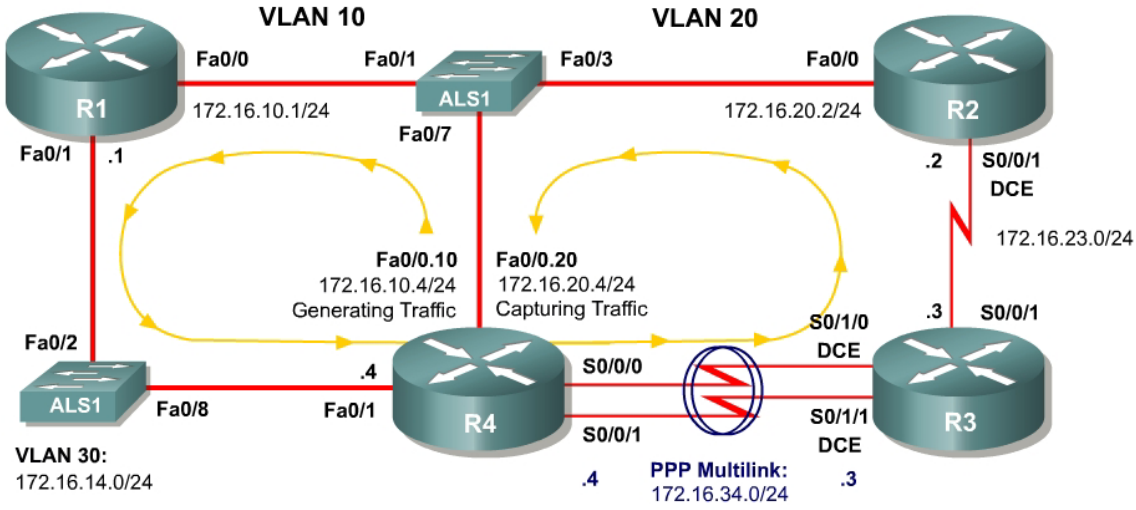
```
R4# show run
!
hostname R4
!
class-map match-all af41
 match  dscp af41
class-map match-all ef
 match  dscp ef
class-map match-all af32
 match  dscp af32
!
policy-map innerpolicy
 class ef
  shape average 40000
 class af41
  shape average 80000
 class af32
  shape average 120000
   compress header ip tcp
policy-map shapingpolicy
 class class-default
  shape peak 400000
  service-policy innerpolicy
!
interface FastEthernet0/1
 ip address 172.16.14.4 255.255.255.0
 no shutdown
!
interface Serial0/0/0
 ip address 172.16.34.4 255.255.255.0
 service-policy output shapingpolicy
 no shutdown
!
router ospf 1
 network 172.16.0.0 0.0.255.255 area 0
!
end
```


Lab 4.8 Shaping and Policing

Learning Objectives

- Use shaping to avoid the effects of policing

Topology Diagram



Scenario

In this lab, you will explore how traffic shaping interacts with traffic policing.

This lab will use the NQR tool from the Pagent toolset to observe delay and jitter statistics as you implement your solutions. You will investigate how different shaping and policing affect packet delay. If you have extra time to complete this lab, do not hesitate to extend this scenario to more configurations than simply those given here.

Typically, commands and command output will only be shown if they have not been implemented in preceding Module 4 labs, so it is highly recommended that you complete Labs 4.1 through 4.7 to ensure knowledge of the queuing, shaping, and policing strategies and their configurations.

Preparation

This lab relies on the Advanced Pagent Configuration which you should have created in Lab 3.1: Preparing for QoS.

Prior to beginning this lab, configure R4 and the switch according to the Advanced Pagent Configuration. You may easily accomplish this on R4 by

loading the *advanced-ios.cfg* file from flash memory into the NVRAM, and reloading.

```
R4# copy flash:advanced-ios.cfg startup-config
Destination filename [startup-config]?
[OK]
2875 bytes copied in 1.456 secs (1975 bytes/sec)
R4# reload
Proceed with reload? [confirm]
```

On the switch, load the *advanced.cfg* file into NVRAM and reload the device.

```
ALS1# copy flash:advanced.cfg startup-config
Destination filename [startup-config]?
[OK]
2875 bytes copied in 1.456 secs (1975 bytes/sec)
ALS1# reload
Proceed with reload? [confirm]
```

Unlike Labs 4.6 and 4.7, this lab will use the NQR tool in the Pagent toolset rather than the TGN traffic generator. Do not load the TGN traffic generator configuration.

Step 1: Configure Physical Interfaces and Routing

1. Configure all IP addresses shown in the diagram and use a clockrate of 800 kbps on all serial links. On the serial interfaces, set the informational bandwidth appropriately.

2. Bind the serial links between R3 and R4 in a PPP multilink. Do not configure Link Fragmentation and Interleaving (LFI) on the multilink interface.

3. Configure OSPF to route for all networks shown in the diagram.

4. Make sure that the outbound queuing method for R3's serial interface facing R2 is WFQ.

```
R1(config)# interface fastethernet 0/0
R1(config-if)# ip address 172.16.10.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# interface fastethernet 0/1
R1(config-if)# ip address 172.16.14.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# router ospf 1
R1(config-router)# network 172.16.0.0 0.0.255.255 area 0

R2(config)# interface fastethernet 0/0
R2(config-if)# ip address 172.16.20.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# interface serial 0/0/1
R2(config-if)# ip address 172.16.23.2 255.255.255.0
R2(config-if)# clockrate 800000
R2(config-if)# no shutdown
R2(config-if)# router ospf 1
R2(config-router)# network 172.16.0.0 0.0.255.255 area 0
```



```
R3(config)# interface serial 0/0/1
R3(config-if)# ip address 172.16.23.3 255.255.255.0
R3(config-if)# fair-queue
R3(config-if)# no shutdown
R3(config-if)# interface serial 0/1/0
R3(config-if)# clockrate 800000
R3(config-if)# bandwidth 800
R3(config-if)# encapsulation ppp
R3(config-if)# ppp multilink
R3(config-if)# ppp multilink group 1
R3(config-if)# no shutdown
R3(config-if)# interface serial0/1/1
R3(config-if)# clockrate 800000
R3(config-if)# bandwidth 800
R3(config-if)# encapsulation ppp
R3(config-if)# ppp multilink
R3(config-if)# ppp multilink group 1
R3(config-if)# no shutdown
R3(config-if)# interface multilink 1
R3(config-if)# ip address 172.16.34.3 255.255.255.0
R3(config-if)# router ospf 1
R3(config-router)# network 172.16.0.0 0.0.255.255 area 0

R4(config)# interface fastethernet 0/1
R4(config-if)# ip address 172.16.14.4 255.255.255.0
R4(config-if)# no shutdown
R4(config-if)# interface serial 0/0/0
R4(config-if)# bandwidth 800
R4(config-if)# encapsulation ppp
R4(config-if)# ppp multilink
R4(config-if)# ppp multilink group 1
R4(config-if)# no shutdown
R4(config-if)# interface serial 0/0/1
R4(config-if)# bandwidth 800
R4(config-if)# encapsulation ppp
R4(config-if)# ppp multilink
R4(config-if)# ppp multilink group 1
R4(config-if)# no shutdown
R4(config-if)# interface multilink 1
R4(config-if)# ip address 172.16.34.4 255.255.255.0
R4(config-if)# router ospf 1
R4(config-router)# network 172.16.0.0 0.0.255.255 area 0
```

Step 2: Configure NQR on R4

The NQR tool in the Pagent toolset can assist network administrators in discovering delay and jitter statistics for traffic traversing their network. Enter NQR configuration mode by issuing the **nqr** command from the privilege EXEC prompt.

Copy and paste the configuration shown below into NQR on R4. This configuration will simulate two traffic streams: a constant high-bandwidth stream and a bursty, lower-bandwidth stream concurrent with it. Please see appendix A for the NETLAB compatible version.

```
fastethernet0/0
add tcp
```



```
send 2000
rate 150
length random 200 to 1000
datalink ios-dependent fastethernet0/0.10
l2-arp-for 172.16.10.1
l3-src 172.16.10.4
l3-dest 172.16.20.4
l4-dest 21
fastethernet0/0.20 ios-dependent capture
add clone-of 1
l4-dest 23
send 500
rate 100
burst on
burst duration on 1000
burst duration off 3000
```

The NQR configuration here sends a controlled amount of packets—2000 for the larger stream, 500 for the smaller stream—and will stop when all packets are sent.

To begin NQR testing, issue either the **start send** command in NQR configuration mode or the **nqr start send** command from privileged EXEC mode. Time will pass, and then the router will inform you when all packets have been sent. There is no need to stop the streams since they will stop on their own.

Finally, issue the **show pkt-seq-drop-stats**, **show delay**, and **show jitter** NQR commands to display drop/resequencing, delay, and jitter statistics, respectively. Example output is shown below, although this type of output will not be shown again later in the lab. Record all statistics by copying and pasting them into a text editor such as Notepad. Record a baseline reading for your current topology.

```
R4(NQR:OFF,Fa0/0:2/2)# start send
R4(NQR:SEND,Fa0/0:2/2)#

   Send process complete.

R4(NQR:WAIT,Fa0/0:2/2)#
R4(NQR:OFF,Fa0/0:2/2)# show pkt-seq-drop-stats

Summary of packet sequence/drop stats of traffic streams
  ts#   template interface      sent     recvd   dropped   out-of-seq  max-seq
  1     TCP      Fa0/0.10*      2000      1919        81           37      568
  2     TCP      Fa0/0.10*       500       500         0            0      500

R4(NQR:OFF,Fa0/0:2/2)# show delay-stats

Summary of delay-stats of traffic streams
 ts#    template interface    min-delay    max-delay    avg-delay  stdev-delay
  1     TCP      Fa0/0.10*     0.004364     0.580043     0.238835     0.143506
  2     TCP      Fa0/0.10*     0.004390     0.273886     0.098115     0.077852

R4(NQR:OFF,Fa0/0:2/2)# show jitter-stats

Summary of jitter-stats of traffic streams
 ts#    template interface   min-jitter   max-jitter   avg-jitter stdev-jitter
```



```
1       TCP       Fa0/0.10*      0.000033      0.367644      0.116765      0.083715
2       TCP       Fa0/0.10*      0.000370      0.156045      0.066655      0.040675
```

Notice that packets are even dropped when no policing or shaping is configured because congestion occurred with only default queuing tools in place.

Step 3: Configure Traffic Policing

On R3, police egress traffic toward R2 to a rate of 700 kbps. Configure this either on a per-interface basis or using a policy-map to police the default class.

Then, run the NQR test again and record and compare statistics with the baseline statistics you captured in Step 2.

This is the way using the MQC, to use the interface-level way you would use the **rate-limit out** command :

```
R3(config)# policy-map mypolicy
R3(config-pmap)# class class-default
R3(config-pmap-c)# police 700000
R3(config-pmap-c-police)# interface serial0/0/1
R3(config-if)# service-policy output mypolicy
```

Run NQR again, record all statistics, and then compare NQR statistics.

How did these packet drop statistics compare to the earlier ones?

More packets were dropped, since the rate was lowered from the original rate (the clockrate set for that link), so some packets had to be dropped.

Identify where packet drops occurred in the topology using the **show interfaces** command.

Step 4: Configure Traffic Shaping

Configure R4 to shape traffic exiting the multilink interface. Shape the traffic down to the same rate that you are using to police traffic on R3. Use either the class-based method by shaping the default class or using the Generic Traffic Shaping on the multilink interface..

This is the way using the MQC, to use the interface-level way you would use the interface-level command **traffic-shape**:

```
R4(config)# policy-map mypolicy
R4(config-pmap)# class class-default
R4(config-pmap-c)# shape peak 700000
R4(config-pmap-c)# interface multilink 1
R4(config-if)# service-policy output mypolicy
```


Run NQR again, record all statistics, and then compare NQR statistics.

How would shaping engender fewer packet drops even if the policing rate was not changed?

When using shaping, traffic that goes over the shaping rate will be buffered, up to a point. Shaping tries to get the traffic to fit a certain profile. After the buffer is filled up, excess traffic will be dropped.

To what real-life scenario is this situation similar?

If a service provider polices to a specified CIR value, it's better to buffer the packets at the exit to your network to minimize the number of packet drops.

Final Configurations

```
R1# show run
!
hostname R1
!
interface FastEthernet0/0
 ip address 172.16.10.1 255.255.255.0
 no shutdown
!
interface FastEthernet0/1
 ip address 172.16.14.1 255.255.255.0
 no shutdown
!
router ospf 1
 network 172.16.0.0 0.0.255.255 area 0
!
end

R2# show run
!
hostname R2
!
interface FastEthernet0/0
 ip address 172.16.20.2 255.255.255.0
 no shutdown
!
interface Serial0/0/1
 ip address 172.16.23.2 255.255.255.0
 clock rate 800000
 no shutdown
!
router ospf 1
 network 172.16.0.0 0.0.255.255 area 0
!
end

R3# show run
!
```



```
hostname R3
!
policy-map mypolicy
 class class-default
   police 700000
!
interface Multilink1
 ip address 172.16.34.3 255.255.255.0
 ppp multilink
 ppp multilink group 1
!
interface Serial0/0/1
 ip address 172.16.23.3 255.255.255.0
 service-policy output mypolicy
 no shutdown
!
interface Serial0/1/0
 bandwidth 800
 no ip address
 encapsulation ppp
 clock rate 800000
 ppp multilink
 ppp multilink group 1
 no shutdown
!
interface Serial0/1/1
 bandwidth 800
 no ip address
 encapsulation ppp
 clock rate 800000
 ppp multilink
 ppp multilink group 1
 no shutdown
!
router ospf 1
 network 172.16.0.0 0.0.255.255 area 0
!
end

R4# show run
!
hostname R4
!
policy-map mypolicy
 class class-default
  shape peak 700000
!
interface Multilink1
 ip address 172.16.34.4 255.255.255.0
 ppp multilink
 ppp multilink group 1
 service-policy output mypolicy
!
interface FastEthernet0/1
 ip address 172.16.14.4 255.255.255.0
 no shutdown
!
interface Serial0/0/0
 bandwidth 800
 ip address 172.16.34.4 255.255.255.0
 encapsulation ppp
 ppp multilink
 ppp multilink group 1
```



```
 no shutdown
!
interface Serial0/0/1
 bandwidth 800
 no ip address
 encapsulation ppp
 ppp multilink
 ppp multilink group 1
 no shutdown
!
router ospf 1
 network 172.16.0.0 0.0.255.255 area 0
!
end
```

Appendix A: NetLab-compatible NQR Configuration

NQR Configuration on R4

```
fastethernet0/0
add tcp
send 2000
rate 150
length random 200 to 1000
l2-dest $R1 Fa0/0's MAC$
l3-src 172.16.10.4
l3-dest 172.16.20.4
l4-dest 21
fastethernet0/0 capture
add clone-of 1
l4-dest 23
send 500
rate 100
burst on
burst duration on 1000
burst duration off 3000
```

