Report on

Project Phase III

EXP-5: Differentiated Service with Traffic Shaping and Policing

Submitted to

Dr. A. Agarwal
Real-time Multimedia Communication over Internet (ELEC6181)

Department of Electrical and Computer Engineering
Gina Cody School of Engineering and Computer Science

By
Pranav Kumar Jha
Student ID: 40081750

Abstract

In this report, different queueing methods have been used on a DiffServ network to enhance the performance for time sensitive applications, also traffic shaping and policing have been applied to enhance the Quality of Service (QoS).

16th April, 2022

Contents

Objectives

The objective of this report is

1. To design a Differentiated Service (DiffServ) Network.
2. To apply traffic shaping and policing on three layer of the DiffServ networ which are Customer network, Wide Area Network (WAN) cloud and Internet Service Providers (ISP) network.
3. To use differentiated service to distinguish traffic types
4. To apply two proper types of queuing methods.
5. To analyze the performance based on this scenarios:
    - Network with best effort
    - Network with policing and shaping
    - Network with queuing and traffic shaping and policing
6. To consider shaping always pushes 20% more traffic than allowed, and policing allows 10% only.

List of Figures

1 Introduction

In this experiment, a Differentiated Service (DiffServ) network is designed using Omnet++ 5.7 with Inet 4.2 framework. The network performance is measured with two different queueing methods namely, best effort queueing and weighted round robin queueing with and without traffic shaping and policing which have been applied to the appropriate routers to enhance the performance of the DiffServ network. These will be discussed in detail in the following sections.

1.1 Network Topology

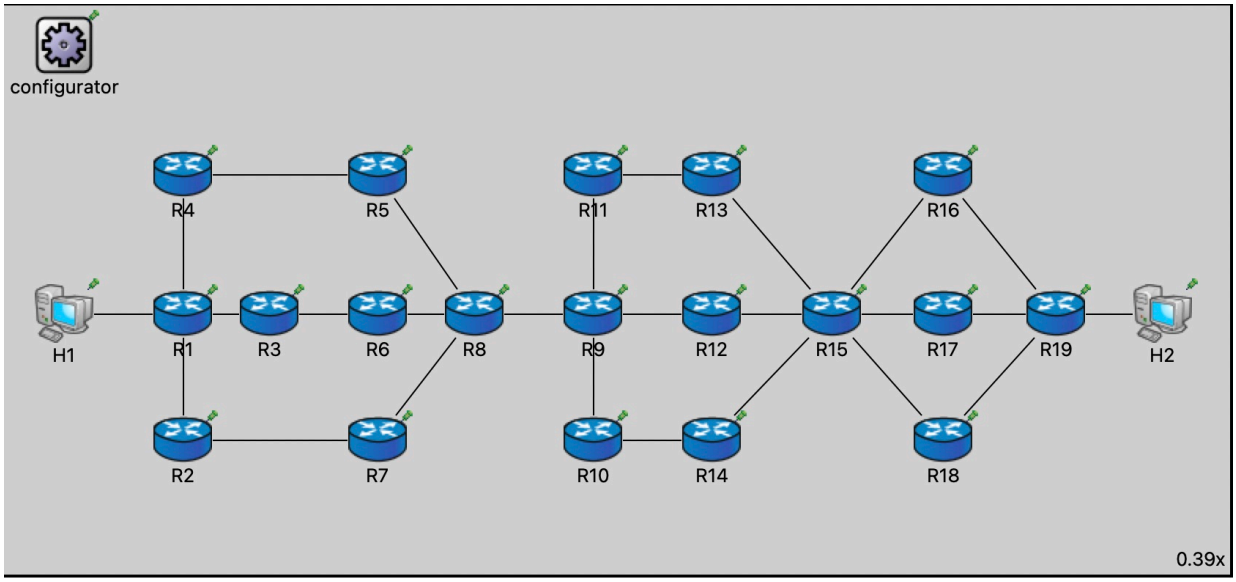


Figure 1.1: Network topology

The network is shown below in the Fig. 1.1 which consists of 19 IPv4 router [R1, R2,...,R19] in total which support wireless, Ethernet, PPP and external interfaces and two standard IPv4 hosts [H1, H2] with Stream Control Transmission Protocol (SCTP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP) layers and their respective applications. Ipv4NetworkConfigurator module is used to assign IPv4 addresses and to set up static routing for the DiffServ network. The network has multiple paths from source H1 to destination H2 such as, [R1-R3-R6-R8-R9-R12-R15-R16-R19], [R1-R3-R6-R8-R9-R12-R15-R17-R19], [R1-R3-R6-R8-R9-R12-R15-R18-R19], etc.

The standard hosts are connected to the routers via Ethernet 100M connections to their respective ppp interfaces and the connections between routers are via ppp interfaces using two different channels namely, edgeline and coreline, which provides better security, reliability and fast transmission of packets. These channels extends the ThruputMeteringChannel from Inet 4.2 and their characteristics are given as

- delay = 2ms;
- datarate = edgeDatarate (32kbps)/coreDatare (16kbps);
- thruputDisplayFormat = ”b B U”;

2 General Parameters and Applications

The ‘result-recording-modes’ are enabled in order to capture the statistical results from the experiments. Further, Open Shortest Path First (OSPF) parameters are also set to true and the ‘OSPFConfig.xml’ file is presented in the Appendix Section 7. The default queueing methods are also provided to the network as shown below,

```
# default queues
**.queue.typename = "EtherQosQueue"
**.queue.dataQueue.typename = "DropTailQueue"
**.queue.packetCapacity = 100
**.queue.dataQueue.packetCapacity = 100
```

Figure 2.1: Default queues

To provide different types of services to the customers, three basic applications have been configured on both the hosts H1 and H2 which are

1. UdpBasicBurst - This is used to stream voice data into the network.
2. UdpBasicApp - This is used to stream video data into the network.
3. TcpBasicClientApp - This is used to stream basic web services.

Further, to configure the network with different link configurations, the edgeline and the coreline have been used between the routers for better security, reliable and to serve packets as

fast as possible. Also, the traffic classifier and markers TC1 are applied at R1 and R19 to identify different types of traffic and classify them and put them in relevant queue, and to associate those traffic types with specific markings as shown in the Fig. 2.2 [1]. The 'filters.xml' file is given in the Appendix Section 7.

```
#Traffic Classifier and Marker
**.R1.eth[?].ingressTC.typename = "TC1"
**.R19.eth[?].ingressTC.typename = "TC1"

**.ingressTC.numClasses = 3
**.ingressTC.classifier.filters = xmldoc("filters.xml", "//experiment[@id='default']")
**.ingressTC.marker.dscps = "AF11 AF21 AF31 AF41 BE"
```

Figure 2.2: Traffic classification and markings

2.1 Configuring Shaping and Policing

Router R8 from the customer network is connected to the ISP Router R9 in the WAN network where shaping have been applied to the egress of the customer router R8 and policing is applied on the ingress traffic R8 of the ISP network. Further, shaping allows 20% more traffic than allowed and policing allows only 10%.

Shaping on Router R8:

```
#Shaping on Router 8
**.R8.ppp[2].egressTC.typename = "TrafficConditioner"
#**.R2.ppp[2].ppp.queue.interfaceTableModule = "^.^.^.interfaceTable"
**.R8.ppp[2].egressTC.efMeter.cir = "70%"
**.R8.ppp[2].egressTC.efMeter.cbs = 50KiB
**.R8.ppp[2].egressTC.defaultMeter.cir = "20%"
**.R8.ppp[2].egressTC.defaultMeter.cbs = 2KiB
**.R8.ppp[2].egressTC.defaultMeter.ebs = 4KiB
```

Figure 2.3: Shaping on R8

Policing on Router R9:

```
#Policing on Router 9
**.R9.ppp[0].ingressTC.typename = "TrafficConditioner"
#**.R4.ppp[0].ppp.queue.interfaceTableModule = "^.^.^.interfaceTable"
**.R9.ppp[0].ingressTC.efMeter.cir = "50%"
**.R9.ppp[0].ingressTC.efMeter.cbs = 40KiB
**.R9.ppp[0].ingressTC.defaultMeter.cir = "10%"
**.R9.ppp[0].ingressTC.defaultMeter.cbs = 2KiB
**.R9.ppp[0].ingressTC.defaultMeter.ebs = 4KiB
```

Figure 2.4: Policing on R9

2.2 Configuring Experiments

The experiments are configured based on these three scenarios:

- **Exp 1: Configuring Best Effort (DropTail) queuing without policing and shaping**

  In this experiment, Best Effort queueing is configured without Traffic Conditioning and the configuration is shown in Fig. 2.5.

```
[Config Exp1DropTailWithoutPolicingAndShaping]

extends = Apps, Exp
**.ppp[*].ppp.queue.typename = "DropTailQueue"
**.eth[*].mac.queue.typename = "EtherQosQueue"
**.eth[*].mac.queue.dataQueue.typename = "DropTailQueue"
**.queue.packetCapacity = 100
```

Figure 2.5: Best effort queueing without policing and shaping

- **Exp 2: Configuring Best Effort (DropTail) queuing with policing and shaping** In this experiment, Best Effort queueing is configured with Traffic Conditioning and the configuration is shown in Fig. 2.6.

```
[Config Exp2DropTailWithPolicingAndShaping]

extends = Apps, Exp, PolicingAndShaping
**.ppp[*].ppp.queue.typename = "DropTailQueue"
**.eth[*].mac.queue.typename = "EtherQosQueue"
**.eth[*].mac.queue.dataQueue.typename = "DropTailQueue"
**.queue.packetCapacity = 100
```

Figure 2.6: Best effort queueing with policing and shaping

- **Exp 3: Configuring WRR DiffServ (DSQueue1) queuing with policing and shaping** In this experiment, WRR queueing is configured with Traffic Conditioning and the configuration is shown in Fig. 2.7.

```
[Config Exp3RoundRobinWithPolicingAndShaping]

extends = Apps, Exp, PolicingAndShaping
**.R*.ppp[*].ppp.queue.typename = "DSQueue1"
**.R*.ppp[*].ppp.queue.packetCapacity = 100 #-1
**.R*.ppp[*].ppp.queue.*.packetCapacity = 100
**.R*.ppp[*].ppp.queue.wrr.weights = "10 40 5 1 1"
```

Figure 2.7: WRR DiffServ DSQueue1 with traffic policing and shaping

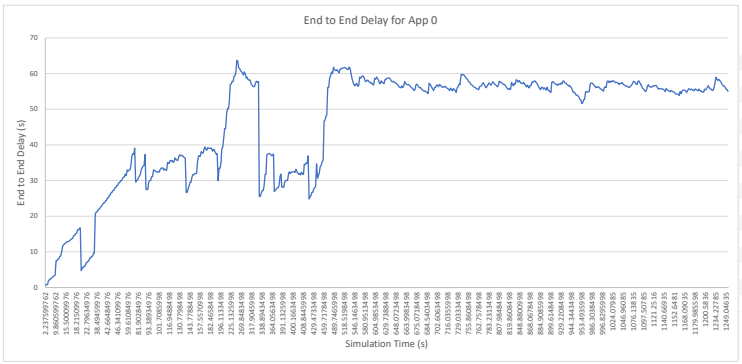
3 Exp 1: Results

The hosts H1 and H2 are configured for three different apps (audio [app0], video [app1] and data [app2]) and the following metrics will be measured for these application:

1. End to End Delay
2. Queueing Delay
3. Packets Dropped
4. Packets Sent and Received

3.1 End to End Delay

3.1.1 App0 [Audio]



(a)

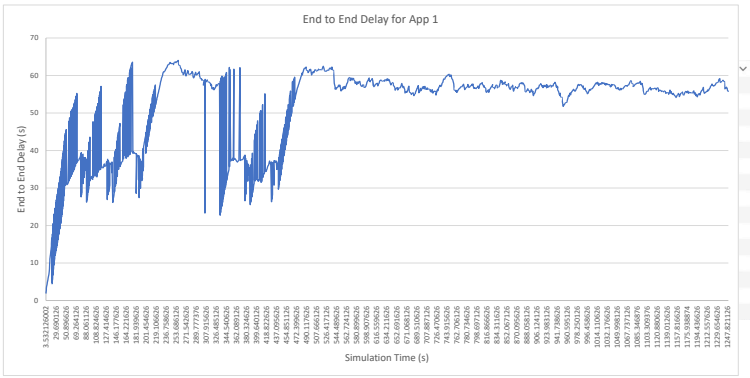
endToEndDelay:vector (vector)	46.21141687219259 (729)
Module name	DiffServ_SHPPOL.H2.app[0]
Type	double
Count	729
Mean	46.21141687219259
StdDev	15.494476986263422
Min	0.68878626
Max	63.663709265852
Start event number	1382
End event number	1035597
Start time	2.237599762304
End time	1249.736599762304
Interpolationmode	none
Source	dataAge(packetReceived)
Title	delay, vector
Unit	s

(b)

Figure 3.1: End to end delay for app0 at H2 for all 172 bytes packet vectors

In Fig. 3.1, the end to end delay for app0 at host H2 is shown for all the packets that has been sent from H1. The end to end delay is 46.21s for the audio transmission over the network which is configured for best effort (DropTail) queueing and without any traffic shaping and policing. This can further be minimized by applying proper QoS techniques such as traffic conditioning, queueing etc. We will see in the following sections that how the use of QoS techniques enhances the performance of the network.

3.1.2 App1 [Video]



(a)

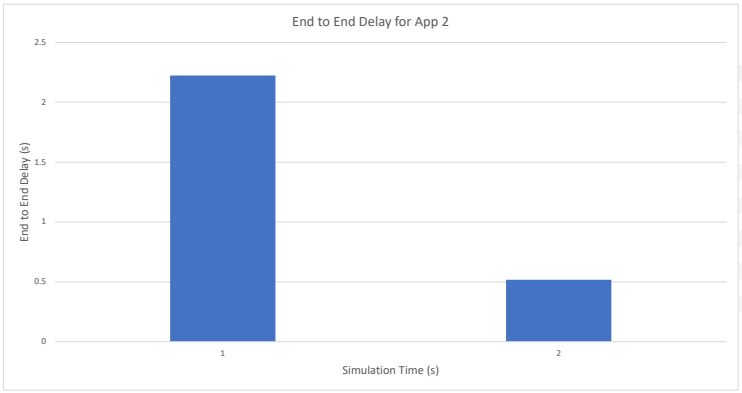
endToEndDelay:vector (vector)	52.00058127422139 (4291)
Module name	DiffServ_SHPPOL.H2.app[1]
Type	double
Count	4291
Mean	52.00058127422139
StdDev	10.875836598039458
Min	1.939281385915
Max	64.169281385915
Start event number	2559
End event number	1035569
Start time	3.532126002304
End time	1249.633126002304
Interpolationmode	none
Source	dataAge(packetReceived)
Title	end-to-end delay, vector
Unit	s

(b)

Figure 3.2: End to end delay for app1 at H2 for all 500 bytes packet vectors

The end to end delay for video transmission for app1 is 52s without any QoS services as shown in Fig. 3.2.

3.1.3 App2 [Data]



(a)

endToEndDelay:vector (vector)	0.34301514 (1)
Module name	DiffServ_SHPPOL.H2.app[2]
Type	double
Count	1
Mean	0.34301514
StdDev	Infinity
Min	0.34301514
Max	0.34301514
Start event number	881
End event number	881
Start time	1.70656162
End time	1.70656162
Interpolationmode	none
Source	dataAge(packetReceived)
Title	end-to-end delay, vector
Unit	s

(b)

Figure 3.3: End to end delay for app2 at H2 for 50 bytes packet vectors

The end to end delay for data transmission for app2 is 0.34s without any QoS services. We can see in Fig. 3.3 that only 1 packet vector of 50 bytes in length has been transmitted over the network for tcp traffic data.

3.2 Queueing Delay

3.2.1 Queueing time on H1

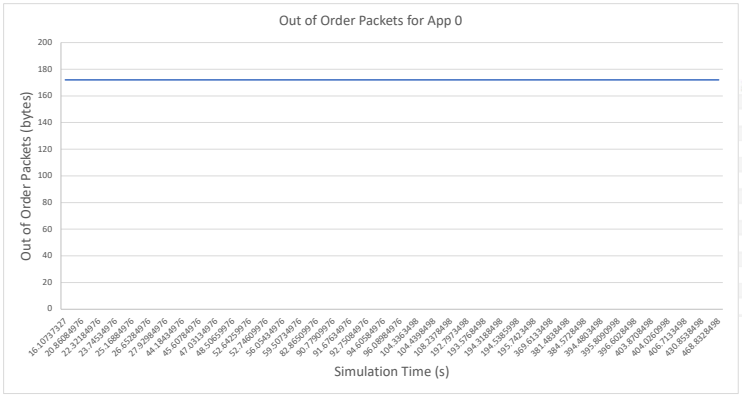
queueingTime:vector (vector)	3.899240203305657E-9 (51548
Module name	DiffServ_SHPPOL.H1.eth[0].mac
Type	double
Count	51548
Mean	3.899240203305657E-9
StdDev	1.4381167240298688E-7
Min	0.0
Max	1.024E-5
Start event number	506
End event number	1025546
Start time	1
End time	1210.249860682304
Interpolationmode	none
Source	queueingTime(packetPopped)
Title	queueing times, vector
Unit	s

Figure 3.4: Queueing time on H1 for eth0

As we can see in Fig. 3.4, the queueing time at host H2 ethernet interface, which uses Eth100M cable connection for connecting ppp interface of the router R1, is 3.89s.

3.3 Packets Dropped

3.3.1 App0 [Audio]



(a)

outOfOrderPk:vector(packetBytes) (vector)	172.0 (79)
Module name	DiffServ_SHPPOL.H2.app[0]
Type	double
Count	79
Mean	172.0
StdDev	0.0
Min	172.0
Max	172.0
Start event number	23059
End event number	423570
Start time	16.107373272441
End time	468.832849762304
Interpolationmode	none
Source	outOfOrderPk
Title	packets received out of order, ve

(b)

Figure 3.5: Out of order packets

The size of the packets that have been dropped are 172 bytes and the total count are 79 for app0 at H2. These packets were in queue but after a certain timeout or queueing delay they have been dropped. There could be other reasons as well such as network congestion, software bugs, and a number of other factors during data transmission that can cause packets loss.

Further, no data has been recorded for packet loss for app1[video] and app2[data] for Experiment 1.

3.4 Packets Sent and Received

3.4.1 App0 [Audio]

packetSent:vector(packetBytes) (vector)	172.0 (21510)
Module name	DiffServ_SHPPOL.H1.app[0]
Type	double
Count	21510
Mean	172.0
StdDev	0.0
Min	172.0
Max	172.0
Start event number	712
End event number	1021485
Start time	1.548813502304
End time	1198.798903571134
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app0

packetReceived:vector(packetBytes) (vector)	172.0 (729)
Module name	DiffServ_SHPPOL.H2.app[0]
Type	double
Count	729
Mean	172.0
StdDev	0.0
Min	172.0
Max	172.0
Start event number	1382
End event number	1035597
Start time	2.237599762304
End time	1249.736599762304
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

(b) Packets received on H2 for app0

Figure 3.6: Packet vectors sent from H1 and received on H2

The total packets sent by H1 over the network are 21510, 172 bytes in size for app0, and the received total packets are 729 at H2.

3.4.2 App1 [Video]

packetSent:vector(packetBytes) (vector)	500.0 (29961)
Module name	DiffServ_SHPPOL.H1.app[1]
Type	double
Count	29961
Mean	500.0
StdDev	0.0
Min	500.0
Max	500.0
Start event number	753
End event number	1022113
Start time	1.592844616389
End time	1199.992844616389
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app1

packetReceived:vector(packetBytes) (vector)	500.0 (4291)
Module name	DiffServ_SHPPOL.H2.app[1]
Type	double
Count	4291
Mean	500.0
StdDev	0.0
Min	500.0
Max	500.0
Start event number	2559
End event number	1035569
Start time	3.532126002304
End time	1249.633126002304
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

(b) Packets received on H2 for app1

Figure 3.7: Packet vectors sent from H1 and received on H2

The total packets that are sent by H1 over the network for app1 are 29961, 500 bytes in size and the total packets received on H2 are 4291.

3.4.3 App2 [Data]

packetSent:vector(packetBytes) (vector)	50.0 (2)
Module name	DiffServ_SHPPOL.H1.app[2]
Type	double
Count	2
Mean	50.0
StdDev	0.0
Min	50.0
Max	50.0
Start event number	632
End event number	523965
Start time	1.36354648
End time	589.249849062304
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app2

packetReceived:vector(packetBytes) (vector)	50.0 (1)
Module name	DiffServ_SHPPOL.H2.app[2]
Type	double
Count	1
Mean	50.0
StdDev	NaN
Min	50.0
Max	50.0
Start event number	881
End event number	881
Start time	1.70656162
End time	1.70656162
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

(b) Packets received on H2 for app2

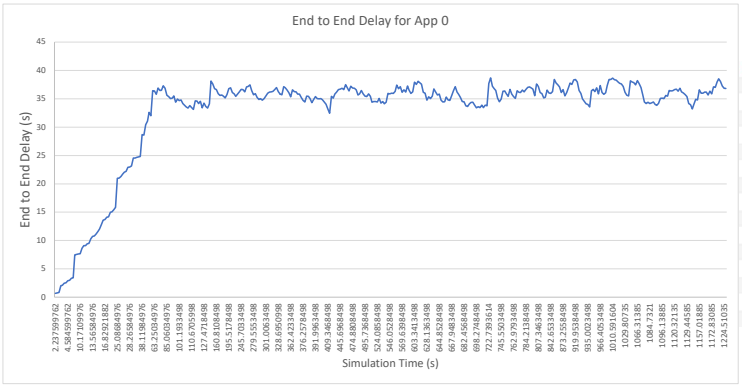
Figure 3.8: Packet vectors sent from H1 and received on H2

The number of packets sent by H1 for app2 are 2 only, 50 bytes in size, and the number received packets are also 1. The packets sent for tcp data is significantly low as compared to app0 or app1 data.

4 Exp 2: Results

4.1 End to End Delay

4.1.1 App0 [Audio]



(a)

endToEndDelay:vector (vector)	32.774202561814214 (380)
Module name	DiffServ_SHPPOL.H2.app[0]
Type	double
Count	380
Mean	32.774202561814214
StdDev	8.36460250282058
Min	0.68878626
Max	38.708862643764
Start event number	1396
End event number	899625
Start time	2.237599762304
End time	1228.501349762304
Interpolationmode	none
Source	dataAge(packetReceived)
Title	delay, vector
Unit	s

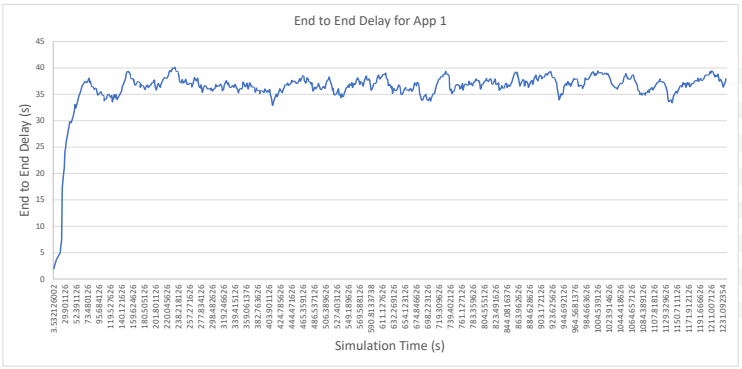
(b)

Figure 4.1: End to end delay for app0 at H2 for all 172 bytes packet vectors

In Fig. 4.1, the end to end delay for app0 at host H2 is shown for all the packets that has been sent from H1. The end to end delay is 32.77s for the audio transmission over the network which is configured for best effort (DropTail) queueing with traffic shaping and policing. This is better

than the end to end delay in Fig. 3.1. We will see in the discussion section 6 about the comparison of these results in detail.

4.1.2 App1 [Video]



(a)

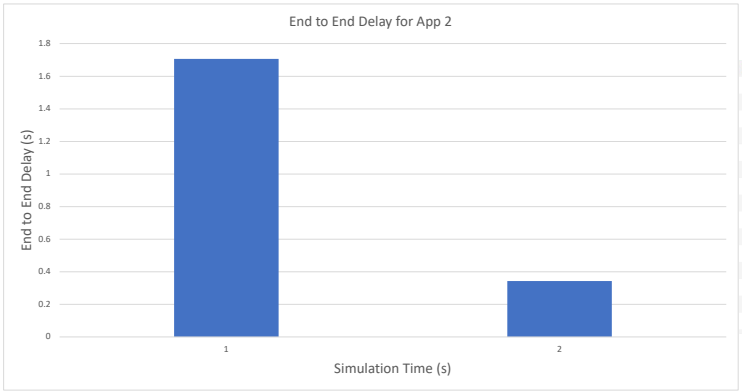
endToEndDelay:vector (vector)	36.22349666136683 (1010)
Module name	DiffServ_SHPPOL.H2.app[1]
Type	double
Count	1010
Mean	36.22349666136683
StdDev	4.1680971660144275
Min	1.939281385915
Max	40.150281385915
Start event number	2588
End event number	901199
Start time	3.532126002304
End time	1237.539626002304
Interpolationmode	none
Source	dataAge(packetReceived)
Title	end-to-end delay, vector
Unit	s

(b)

Figure 4.2: End to end delay for app1 at H2 for all 500 bytes packet vectors

The end to end delay, as shown in Fig. 4.2, for video transmission for app1 is 36.22s with traffic shaping and policing on Router R8 and R9, respectively.

4.1.3 App2 [Data]



(a)

endToEndDelay:vector (vector)	0.34301514 (1)
Module name	DiffServ_SHPPOL.H2.app[2]
Type	double
Count	1
Mean	0.34301514
StdDev	Infinity
Min	0.34301514
Max	0.34301514
Start event number	889
End event number	889
Start time	1.70656162
End time	1.70656162
Interpolationmode	none
Source	dataAge(packetReceived)
Title	end-to-end delay, vector
Unit	s

(b)

Figure 4.3: End to end delay for app2 at H2 for 50 bytes packet vectors

The end to end delay for data transmission, as shown in Fig. 4.3 for app2 is 0.34s with traffic shaping and policing and there is no change on these tcp traffic data by applying QoS services.

4.2 Queueing Delay

4.2.1 Queueing time on H1

queueingTime:vector (vector)	2.9906346377598514E-9 (5156
Module name	DiffServ_SHPPOL.H1.eth[0].mac
Type	double
Count	51568
Mean	2.9906346377598514E-9
StdDev	1.0810375805965758E-7
Min	0.0
Max	8.213488E-6
Start event number	508
End event number	893278
Start time	1
End time	1199.992844616389
Interpolationmode	none
Source	queueingTime(packetPopped)
Title	queueing times, vector
Unit	s

Figure 4.4: Queueing time on H1 for eth0

As we can see in Fig. 4.4, the queueing time at host H2 ethernet interface, which uses Eth100M cable connection for connecting ppp interface of the router R1, is 2.99s, which is better than the queueing time without any traffic conditioning as shown in Fig. 3.4, and we will see these comparisons in detail in the discussion section 6.

4.3 Packets Dropped

No packet loss has been recorded for Exp 2, best effort with shaping and policing in any of these apps app0, app1 and app2.

4.4 Packets Sent and Received

4.4.1 App0 [Audio]

packetSent:vector(packetBytes) (vector)	172.0 (21510)
Module name	DiffServ_SHPPOL.H1.app[0]
Type	double
Count	21510
Mean	172.0
StdDev	0.0
Min	172.0
Max	172.0
Start event number	720
End event number	892713
Start time	1.548813502304
End time	1198.798903571134
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app0

packetReceived:vector(packetBytes) (vector)	172.0 (380)
Module name	DiffServ_SHPPOL.H2.app[0]
Type	double
Count	380
Mean	172.0
StdDev	0.0
Min	172.0
Max	172.0
Start event number	1396
End event number	899625
Start time	2.237599762304
End time	1228.501349762304
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

(b) Packets received on H2 for app0

Figure 4.5: Packet vectors sent from H1 and received on H2

The total packets sent by H1 over the network are 21510, 172 bytes in size for app0, and the received total packets are 380, only at H2.

4.4.2 App1 [Video]

packetSent:vector(packetBytes) (vector)	500.0 (29961)
Module name	DiffServ_SHPPOL.H1.app[1]
Type	double
Count	29961
Mean	500.0
StdDev	0.0
Min	500.0
Max	500.0
Start event number	761
End event number	893274
Start time	1.592844616389
End time	1199.992844616389
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app1

packetReceived:vector(packetBytes) (vector)	500.0 (1010)
Module name	DiffServ_SHPPOL.H2.app[1]
Type	double
Count	1010
Mean	500.0
StdDev	0.0
Min	500.0
Max	500.0
Start event number	2588
End event number	901199
Start time	3.532126002304
End time	1237.539626002304
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

(b) Packets received on H2 for app1

Figure 4.6: Packet vectors sent from H1 and received on H2

The total packets that are sent by H1 over the network for app1 are 29961, 500 bytes in size and the total packets received on H2 are 1010.

4.4.3 App2 [Data]

packetSent:vector(packetBytes) (vector)	50.0 (1)
Module name	DiffServ_SHPPOL.H1.app[2]
Type	double
Count	1
Mean	50.0
StdDev	NaN
Min	50.0
Max	50.0
Start event number	636
End event number	636
Start time	1.36354648
End time	1.36354648
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app2

packetReceived:vector(packetBytes) (vector)	50.0 (1)
Module name	DiffServ_SHPPOL.H2.app[2]
Type	double
Count	1
Mean	50.0
StdDev	NaN
Min	50.0
Max	50.0
Start event number	889
End event number	889
Start time	1.70656162
End time	1.70656162
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

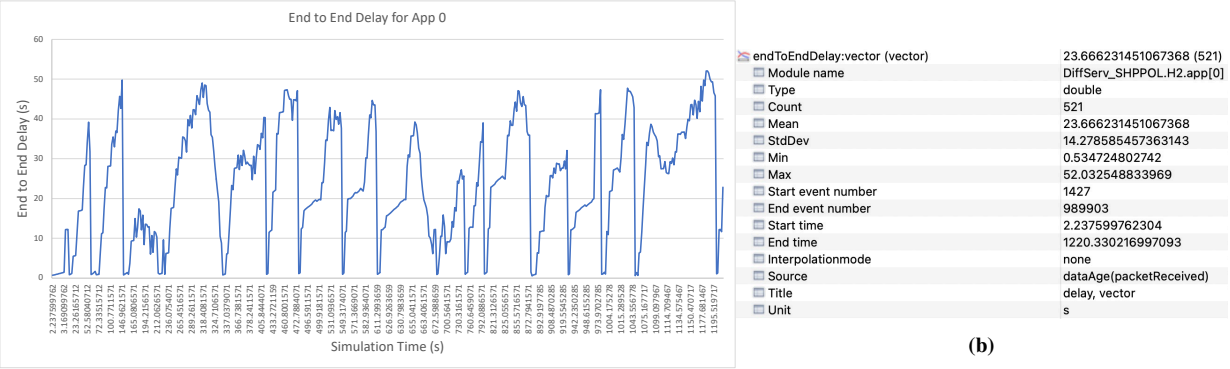
(b) Packets received on H2 for app2

Figure 4.7: Packet vectors sent from H1 and received on H2

The number of packets sent by H1 for app2 are 1 only, 50 bytes in size, and the number of received packets are also 1. Once again, the packets sent for tcp data is significantly low as compared to app0 or app1 data.

5 Exp 3: Results

5.0.1 App0 [Audio]

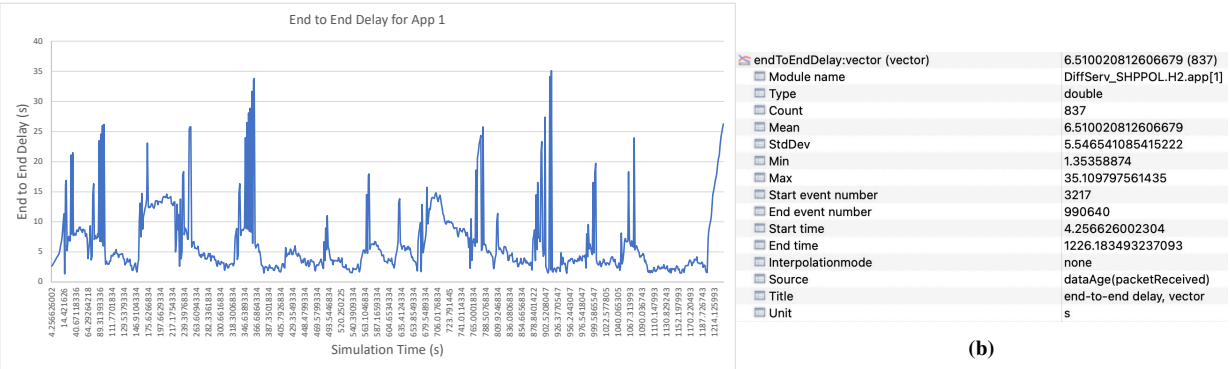


(a) **(b)**

Figure 5.1: End to end delay for app0 at H2 for all 172 bytes packet vectors

In Fig. 5.1, the end to end delay for app0 at host H2 is shown for all the packets that has been sent from H1. The end to end delay is 23.66s for the audio transmission over the network which is configured for WRR DiffServ (DSQueue1) queueing along with traffic shaping and policing. This is better than the end to end delay in Fig. 3.1 and Fig. 4.1.

5.0.2 App1 [Video]

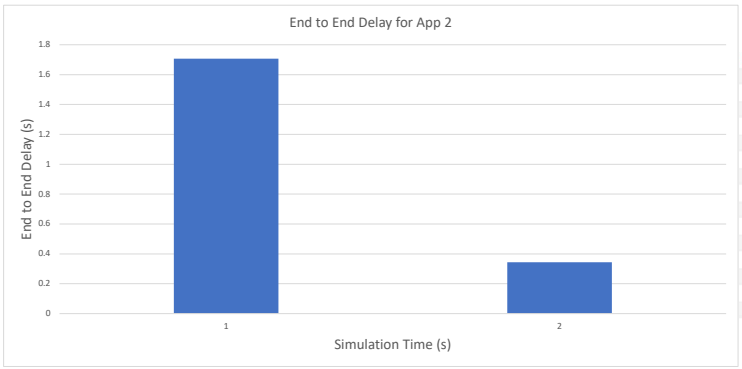


(a) **(b)**

Figure 5.2: End to end delay for app1 at H2 for all 500 bytes packet vectors

The end to end delay, as shown in Fig. 5.2, for video transmission for app1 is 6.51s with traffic shaping and policing on the customer network and WAN.

5.0.3 App2 [Data]



(a)

endToEndDelay:vector (vector)	0.34301514 (1)
Module name	DiffServ_SHPPOL.H2.app[2]
Type	double
Count	1
Mean	0.34301514
StdDev	Infinity
Min	0.34301514
Max	0.34301514
Start event number	889
End event number	889
Start time	1.70656162
End time	1.70656162
Interpolationmode	none
Source	dataAge(packetReceived)
Title	end-to-end delay, vector
Unit	s

(b)

Figure 5.3: End to end delay for app2 at H2 for 50 bytes packet vectors

The end to end delay for data transmission, as shown in Fig. 5.3 for app2 is 0.34s with traffic shaping and policing and there is no change on these tcp traffic data by applying QoS services.

5.1 Queueing Delay

5.1.1 Queueing time on H1

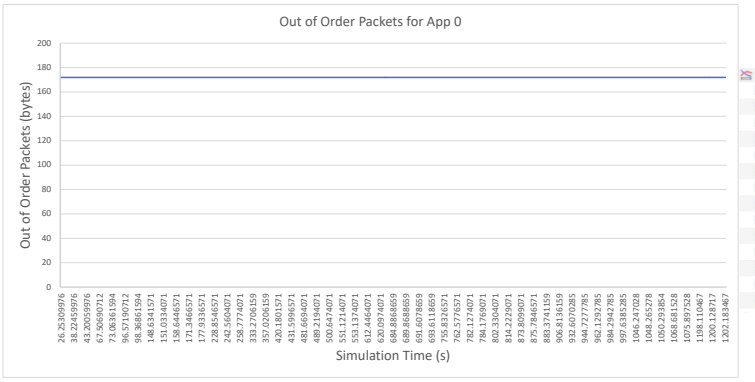
queueingTime:vector (vector)	3.7028172059708446E-9 (51517
Module name	DiffServ_SHPPOL.H1.eth[0].mac
Type	double
Count	51517
Mean	3.7028172059708446E-9
StdDev	1.366027355251876E-7
Min	0.0
Max	1.024E-5
Start event number	508
End event number	989378
Start time	1
End time	1219.863918036389
Interpolationmode	none
Source	queueingTime(packetPopped)
Title	queueing times, vector
Unit	s

Figure 5.4: Queueing time on H1 for eth0

As we can see in Fig. 5.4, the queueing time at host H2 ethernet interface, which uses Eth100M cable connection for connecting ppp interface of the router R1, is 3.70s.

5.2 Packets Dropped

5.2.1 App0 [Audio]



(a)

outOfOrderPk:vector(packetBytes) (vector)	172.0 (261)
Module name	DiffServ_SHPPOL.H2.app[0]
Type	double
Count	261
Mean	172.0
StdDev	0.0
Min	172.0
Max	172.0
Start event number	32272
End event number	986009
Start time	26.253099762304
End time	1202.183466997093
Interpolationmode	none
Source	outOfOrderPk
Title	packets received out of order, ve

(b)

Figure 5.5: Out of order packets

The size of the packets that have been dropped are 172 bytes and the total count are 261 for app0 at H2. No packet loss has been detected for app1 and app2 data transmission over the network.

5.3 Packets Sent and Received

5.3.1 App0 [Audio]

packetSent:vector(packetBytes) (vector)	172.0 (21510)
Module name	DiffServ_SHPPOL.H1.app[0]
Type	double
Count	21510
Mean	172.0
StdDev	0.0
Min	172.0
Max	172.0
Start event number	720
End event number	983418
Start time	1.548813502304
End time	1198.798903571134
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app0

packetReceived:vector(packetBytes) (vector)	172.0 (521)
Module name	DiffServ_SHPPOL.H2.app[0]
Type	double
Count	521
Mean	172.0
StdDev	0.0
Min	172.0
Max	172.0
Start event number	1427
End event number	989903
Start time	2.237599762304
End time	1220.330216997093
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

(b) Packets received on H2 for app0

Figure 5.6: Packet vectors sent from H1 and received on H2

The total packets sent by H1 over the network are 21510, 172 bytes in size for app0, and the received total packets are 521 at H2.

5.3.2 App1 [Video]

packetSent:vector(packetBytes) (vector)	500.0 (29961)
Module name	DiffServ_SHPPOL.H1.app[1]
Type	double
Count	29961
Mean	500.0
StdDev	0.0
Min	500.0
Max	500.0
Start event number	761
End event number	984310
Start time	1.592844616389
End time	1199.992844616389
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app1

packetReceived:vector(packetBytes) (vector)	500.0 (837)
Module name	DiffServ_SHPPOL.H2.app[1]
Type	double
Count	837
Mean	500.0
StdDev	0.0
Min	500.0
Max	500.0
Start event number	3217
End event number	990640
Start time	4.256626002304
End time	1226.183493237093
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

(b) Packets received on H2 for app1

Figure 5.7: Packet vectors sent from H1 and received on H2

The total packets that are sent by H1 over the network for app1 are 29961, 500 bytes in size and the total packets received on H2 are 837.

5.3.3 App2 [Data]

packetSent:vector(packetBytes) (vector)	50.0 (2)
Module name	DiffServ_SHPPOL.H1.app[2]
Type	double
Count	2
Mean	50.0
StdDev	0.0
Min	50.0
Max	50.0
Start event number	636
End event number	109693
Start time	1.36354648
End time	118.863906416389
Interpolationmode	none
Source	packetSent
Title	packets sent, vector(packetByte

(a) Packets sent from H1 for app2

packetReceived:vector(packetBytes) (vector)	50.0 (1)
Module name	DiffServ_SHPPOL.H2.app[2]
Type	double
Count	1
Mean	50.0
StdDev	NaN
Min	50.0
Max	50.0
Start event number	889
End event number	889
Start time	1.70656162
End time	1.70656162
Interpolationmode	none
Source	packetReceived
Title	packets received, vector(packetI

(b) Packets received on H2 for app2

Figure 5.8: Packet vectors sent from H1 and received on H2

The number of packets sent by H1 for app2 are 2 only, 50 bytes in size, and the number of received packets are 1. The packets sent for tcp data is significantly low as compared to app0 or app1 data.

6 Discussion

6.1 Comparison of results

6.1.1 End to end delay

App	Exp 1	Exp 2	Exp 3
app0 [audio]	46.21s	32.77s	23.66s
app1 [video]	52.00s	36.22s	6.51s
app2 [data]	0.34s	0.34s	0.34s

Table 6.1: End to end delay comparison between Exp 1, Exp 2 and Exp 3

As shown in the Table 6.3 above, the end to end delay calculation is summarized. For app0[audio], the end to end delay in Exp 1 (46.21s) is improving after applying shaping and policing in Exp 2 (32.77s) and changing the queueing to WRR from best effort in Exp 3 (23.66s). For app1[video], the end to end delay performance in Exp 1 has improved from 52s to 36.22s in Exp 2 after applying traffic conditioning, however, it has improved significantly in Exp 3 which has a value of 6.51s. For app2[data], there is no improvement in any of the experiments.

6.1.2 Queueing delay

In Table 6.2, the queueing time for H1 Eth0 interface has been given for all three experiments. It can be observed that with traffic shaping and policing, the queueing time has improved from 3.89s (Exp 1) to 2.99s (Exp2), however, after changing the queueing to WRR in Exp 3, the queueing time is deteriorated and attains the value of 3.70s.

H1	Exp 1	Exp 2	Exp 3
Eth0	3.89s	2.99s	3.70s

Table 6.2: Queueing delay comparison

6.1.3 Packets dropped

The packets have been dropped for audio data (app0) in Exp 1 and Exp 3, where no traffic conditioning has been applied. Whereas, there is no loss of packets that have been recorded for other apps in all these experiments for app1 and app2.

6.1.4 Packets sent and received

For number of packets that are sent over the network for app0[audio] are 21510 in Exp 1-Exp 3, and the received packets are 729, 380 and 521 for Exp 1, Exp 2 and Exp 3, respectively. It's evident that applying traffic shaping and policing with best effort queueing, supressed the reception of packets (Exp 2 and Exp 3), but with WRR in Exp 3, it's been improved.

App	**Exp 1**	**Exp 1**	**Exp 2**	**Exp 2**	**Exp 3**	**Exp 3**
	Sent	Received	Sent	Received	Sent	Received
0	21510	729	21510	380	21150	521
1	29961	4291	29961	1010	29961	837
2	2	1	1	1	2	1

Table 6.3: Comparison between packets sent and received for Exp 1, Exp 2 and Exp 3

7 Conclusion

In this report, the network has been designed with best effort (DropTail) queueing and traffic shaping and policing have been applied at the egress and ingress of the core routers. Further, the queueing has been changed to WRR (DiffServ) and the performnace is measured against each of the experiments configured. For the designed network, the best effort queueing with traffic shaping and policing (Exp 2), in terms of end to end delay, has outperforms the best effort without traffic shaping and policing (Exp 1) and WRR with traffic shaping and policing (Exp 3) for apps(0[audio] and 1[video]. But for app2, the performance is not good in any of these experiments. Also, there has no packet loss been recorded in the Exp 2 against any of the apps (audio, video and data). Further, for queueing delay, best effort with traffic shaping and policing has again performed better that others. But, for packets received, best effort without traffic shaping and policing has proved to be better.

References

[1] C. Press. “Cisco ip telephony flash cards: Weighted random early detection (wred),” *Cisco Press,*. [Online]. Available: https://www.ciscopress.com/articles/article.asp?p=352991&seqNum=6

Appendix

OSPFConfig.xml

```
<?xml version="1.0"?>
<OSPFASConfig>
<Area id="0.0.0.0">
<AddressRange address="H1" mask="H1" />
<AddressRange address="H2" mask="H2" />
<AddressRange address="R1>R2" mask="R1>R2" />
<AddressRange address="R2>R1" mask="R2>R1" />
<AddressRange address="R1>R3" mask="R1>R3" />
<AddressRange address="R3>R1" mask="R3>R1" />
<AddressRange address="R1>R4" mask="R1>R4" />
<AddressRange address="R4>R1" mask="R4>R1" />
<AddressRange address="R2>R7" mask="R2>R7" />
<AddressRange address="R7>R2" mask="R7>R2" />
<AddressRange address="R3>R6" mask="R3>R6" />
<AddressRange address="R6>R3" mask="R6>R3" />
<AddressRange address="R5>R4" mask="R5>R4" />
<AddressRange address="R4>R5" mask="R4>R5" />
<AddressRange address="R5>R8" mask="R5>R8" />
<AddressRange address="R8>R5" mask="R8>R5" />
<AddressRange address="R6>R8" mask="R6>R8" />
<AddressRange address="R8>R6" mask="R8>R6" />
<AddressRange address="R9>R8" mask="R9>R8" />
<AddressRange address="R8>R9" mask="R8>R9" />
<AddressRange address="R8>R7" mask="R8>R7" />
<AddressRange address="R7>R8" mask="R7>R8" />
<AddressRange address="R9>R10" mask="R9>R10" />
<AddressRange address="R10>R9" mask="R10>R9" />
<AddressRange address="R9>R11" mask="R9>R11" />
<AddressRange address="R11>R9" mask="R11>R9" />
<AddressRange address="R9>R12" mask="R9>R12" />
<AddressRange address="R12>R9" mask="R12>R9" />
```

```
32 <AddressRange address="R10>R14" mask="R10>R14" />
33 <AddressRange address="R14>R10" mask="R14>R10" />
34 <AddressRange address="R11>R13" mask="R11>R13" />
35 <AddressRange address="R13>R11" mask="R13>R11" />
36 <AddressRange address="R15>R12" mask="R15>R12" />
37 <AddressRange address="R12>R15" mask="R12>R15" />
38 <AddressRange address="R15>R13" mask="R15>R13" />
39 <AddressRange address="R13>R15" mask="R13>R15" />
40 <AddressRange address="R14>R15" mask="R14>R15" />
41 <AddressRange address="R15>R14" mask="R15>R14" />
42 <AddressRange address="R15>R16" mask="R15>R16" />
43 <AddressRange address="R16>R15" mask="R16>R15" />
44 <AddressRange address="R15>R17" mask="R15>R17" />
45 <AddressRange address="R17>R15" mask="R17>R15" />
46 <AddressRange address="R15>R18" mask="R15>R18" />
47 <AddressRange address="R18>R15" mask="R18>R15" />
48 <AddressRange address="R16>R19" mask="R16>R19" />
49 <AddressRange address="R19>R16" mask="R19>R16" />
50 <AddressRange address="R17>R19" mask="R17>R19" />
51 <AddressRange address="R19>R17" mask="R19>R17" />
52 <AddressRange address="R18>R19" mask="R18>R19" />
53 <AddressRange address="R19>R18" mask="R19>R18" />
54 </Area>
55 <Router name="**" RFC1583Compatible="true">
56 <BroadcastInterface ifName='eth[*]' areaID='0.0.0.0' interfaceOutputCost=
       '0' />
57 <PointToPointInterface ifName='ppp[*]' areaID='0.0.0.0'
       interfaceOutputCost='0' />
58 </Router>
59 </OSPFASConfig>
```

filters.xml

```
<filters>
<experiment id="default">
<filter srcAddress="H1" destPort="1000" gate="0"/>
<filter srcAddress="H1" destPort="2000" gate="1"/>
<filter srcAddress="H1" destPort="3000" gate="2"/>
<filter srcAddress="H2" gate="0"/>
</experiment>
</filters>
```

ned file

```
package project_DiffServ_SHPPOL;

import inet.examples.inet.netperfmeter.coreChannel;
import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurat
import inet.common.misc.ThruputMeteringChannel;
import inet.node.ethernet.Eth100M;
import inet.node.inet.Router;
import inet.node.inet.StandardHost;
@license(LGPL);

network DiffServ_SHPPOL
{
    parameters:
        double edgeDatarate @unit(bps);
        double coreDatarate @unit(bps);
        @display("bgb=2139.5813,991.825");
    //Channels
    types:
        channel edgeline extends ThruputMeteringChannel
        {
            delay = 2ms;
            datarate = edgeDatarate;
            thruputDisplayFormat = "b B U";
        }
        channel coreline extends ThruputMeteringChannel
        {
            delay = 2ms;
            datarate = coreDatarate;
            thruputDisplayFormat = "b B U";
        }

    submodules:
        R1: Router {
            @display("p=311.86252,536.8125");
            gates:
                ethg[1];
        }
        R2: Router {
            @display("p=311.86252,756.65");
        }
        R3: Router {
            @display("p=462.68124,536.8125");
        }
        R4: Router {
            @display("p=311.86252,293.96875");
        }
        R5: Router {
            @display("p=651.84375,293.96875");
        }
        R6: Router {
            @display("p=651.84375,536.8125");
        }
        R7: Router {
            @display("p=651.84375,756.65");
        }

        R11: Router {
            @display("p=1027.6125,293.96875");
        }
        R12: Router {
            @display("p=1234.6688,536.8125");
        }
        R13: Router {
            @display("p=1234.6688,293.96875");
        }
        R9: Router {
            @display("p=1027.6125,536.8125");
        }
        R10: Router {
            @display("p=1027.6125,756.65");
        }
        R14: Router {
            @display("p=1234.6688,756.65");
        }
        R8: Router {
            @display("p=820.5563,536.8125");
        }
        R15: Router {
            @display("p=1444.2812,536.8125");
        }
        R16: Router {
            @display("p=1638.5563,293.96875");
        }
        R17: Router {
            @display("p=1638.5563,536.8125");
        }
        R18: Router {
            @display("p=1638.5563,756.65");
        }
        R19: Router {
            @display("p=1835.3876,536.8125");
            gates:
                ethg[1];
        }
        H1: StandardHost {
            @display("p=104.80625,536.8125");
            gates:
                ethg[1];
        }
        H2: StandardHost {
            @display("p=2021.9938,536.8125");
            gates:
                ethg[1];
        }
        configurator: Ipv4NetworkConfigurator {
            @display("p=105.21875,71.09375");
        }

    connections:
        //Connections between hosts and routers
        H1.ethg++ <--> Eth100M <--> R1.ethg++;
        H2.ethg++ <--> Eth100M <--> R19.ethg++;

        //Connections between routers
        R1.pppg++ <--> edgeline <--> R4.pppg++;
        R1.pppg++ <--> edgeline <--> R2.pppg++;
        R13.pppg++ <--> edgeline <--> R15.pppg++;
        R12.pppg++ <--> edgeline <--> R15.pppg++;
        R7.pppg++ <--> edgeline <--> R8.pppg++;
        R3.pppg++ <--> edgeline <--> R6.pppg++;
        R13.pppg++ <--> edgeline <--> R11.pppg++;
        R2.pppg++ <--> coreline <--> R7.pppg++;
        R4.pppg++ <--> edgeline <--> R5.pppg++;
        R5.pppg++ <--> coreline <--> R8.pppg++;
        R8.pppg++ <--> coreline <--> R9.pppg++;
        R9.pppg++ <--> coreline <--> R12.pppg++;
        R9.pppg++ <--> coreline <--> R11.pppg++;
        R9.pppg++ <--> coreline <--> R10.pppg++;
        R10.pppg++ <--> coreline <--> R14.pppg++;
        R14.pppg++ <--> coreline <--> R15.pppg++;
        R1.pppg++ <--> edgeline <--> R3.pppg++;
        R6.pppg++ <--> edgeline <--> R8.pppg++;
        R15.pppg++ <--> coreline <--> R16.pppg++;
        R15.pppg++ <--> coreline <--> R17.pppg++;
        R19.pppg++ <--> coreline <--> R16.pppg++;
        R19.pppg++ <--> coreline <--> R17.pppg++;
        R15.pppg++ <--> edgeline <--> R18.pppg++;
        R18.pppg++ <--> edgeline <--> R19.pppg++;
}
```

omnetpp.ini file

```
[General]
network = DiffServ_SHPPOL

sim-time-limit = 1250s
**.result-recording-modes = all
**.scalar-recording = true

# default queues
**.queue.typename = "EtherQosQueue"
**.queue.dataQueue.typename = "DropTailQueue"
**.queue.packetCapacity = 100
**.queue.dataQueue.packetCapacity = 100

# Enable OSPF on all routers
**.R*.hasOspf = true
**.R*.ospf.ospfConfig = xmldoc("OSPFConfig.xml")

[Config Apps]
**.H?.numApps = 3
# first app: voice streaming
**.H1.app[0].typename = "UdpBasicBurst"
**.H1.app[0].destAddresses = "H2"
**.H1.app[0].chooseDestAddrMode = "once"
**.H1.app[0].destPort = 1000
**.H1.app[0].startTime = uniform(1s,2s)
**.H1.app[0].stopTime = 1200s
**.H1.app[0].messageLength = 172B # 160B voice + 12B Rtp header
**.H1.app[0].burstDuration = exponential(0.352s)
**.H1.app[0].sleepDuration = exponential(0.650s)
**.H1.app[0].sendInterval = 20ms

**.H2.app[0].typename = "UdpBasicBurst"
**.H2.app[0].localPort = 1000
**.H2.app[0].delayLimit = 0ms
**.H2.app[0].destAddresses = ""
**.H2.app[0].chooseDestAddrMode = "once"
**.H2.app[0].destPort = 0
**.H2.app[0].messageLength = 0B
**.H2.app[0].burstDuration = 0s
**.H2.app[0].sleepDuration = 0s
**.H2.app[0].sendInterval = 0ms

#second app: video
**.H1.app[1].typename = "UdpBasicApp"
**.H1.app[1].destPort = 2000
**.H1.app[1].startTime = uniform(1s,2s)
**.H1.app[1].stopTime = 1200s
**.H1.app[1].sendInterval = 40ms
**.H1.app[1].messageLength = 500B
**.H1.app[1].destAddresses = "H2"

**.H2.app[1].typename = "UdpSink" #"UdpEchoApp"
**.H2.app[1].localPort = 2000

#third app: tcp traffic
**.H1.app[2].typename = "TcpBasicClientApp"
**.H1.app[2].connectAddress = "H2"
**.H1.app[2].connectPort = 3000
**.H1.app[2].numRequestsPerSession = 1 # HTTP 1.0
**.H1.app[2].requestLength = 50B
**.H1.app[2].replyLength = 100B
**.H1.app[2].thinkTime = 40ms
**.H1.app[2].idleInterval = 10s
**.H1.app[2].stopTime = 1200s

**.H2.app[2].typename = "TcpGenericServerApp"
**.H2.app[2].localPort = 3000

[Config Exp]
**.edgeDatarate = 32kbps
**.coreDatarate = 16kbps
```

```
#Traffic Classifier and Marker
**.R1.eth[?].ingressTC.typename = "TC1"
**.R19.eth[?].ingressTC.typename = "TC1"

**.ingressTC.numClasses = 3
**.ingressTC.classifier.filters = xmldoc("filters.xml", "//experiment[@id='default']")
**.ingressTC.marker.dscps = "AF11 AF21 AF31 AF41 BE"

# statistics
**.H?.app[*].sentPk.result-recording-modes = count
**.H?.app[*].rcvdPk.result-recording-modes = count
**.H?.app[*].dropPk.result-recording-modes = vector
**.H?.app[*].endToEndDelay.result-recording-modes = vector # for computing median

**.R?.ppp[*].**Queue.rcvdPk.result-recording-modes = count
**.R?.ppp[*].**Queue.dropPk.result-recording-modes = count
**.R?.ppp[*].**Queue.queueLength.result-recording-modes = timeavg
**.R?.ppp[*].**Queue.queueingTime.result-recording-modes = vector # for computing median
**.R?.ppp[*].**Queue.*.scalar-recording = true

**.app[*].sentPk*.scalar-recording = true
**.app[*].rcvdPk*.scalar-recording = true
**.app[*].endToEndDelay*.scalar-recording = true

**.afQueue.*.scalar-recording = true

[Config PolicingAndShaping]

#Shaping on Router 8
**.R8.ppp[2].egressTC.typename = "TrafficConditioner"
#**.R2.ppp[2].ppp.queue.interfaceTableModule = "^.^.^.interfaceTable"
**.R8.ppp[2].egressTC.efMeter.cir = "70%"
**.R8.ppp[2].egressTC.efMeter.cbs = 50KiB
**.R8.ppp[2].egressTC.defaultMeter.cir = "20%"
**.R8.ppp[2].egressTC.defaultMeter.cbs = 2KiB
**.R8.ppp[2].egressTC.defaultMeter.ebs = 4KiB

#Policing on Router 9
**.R9.ppp[0].ingressTC.typename = "TrafficConditioner"
#**.R4.ppp[0].ppp.queue.interfaceTableModule = "^.^.^.interfaceTable"
**.R9.ppp[0].ingressTC.efMeter.cir = "50%"
**.R9.ppp[0].ingressTC.efMeter.cbs = 40KiB
**.R9.ppp[0].ingressTC.defaultMeter.cir = "10%"
**.R9.ppp[0].ingressTC.defaultMeter.cbs = 2KiB
**.R9.ppp[0].ingressTC.defaultMeter.ebs = 4KiB

[Config Exp1DropTailWithoutPolicingAndShaping] #Best Effort without Policing and Shaping

extends = Apps, Exp
**.ppp[*].ppp.queue.typename = "DropTailQueue"
**.eth[*].mac.queue.typename = "EtherQosQueue"
**.eth[*].mac.queue.dataQueue.typename = "DropTailQueue"
**.queue.packetCapacity = 100

[Config Exp2DropTailWithPolicingAndShaping] #Best Effort with Policing and Shaping

extends = Apps, Exp, PolicingAndShaping
**.ppp[*].ppp.queue.typename = "DropTailQueue"
**.eth[*].mac.queue.typename = "EtherQosQueue"
**.eth[*].mac.queue.dataQueue.typename = "DropTailQueue"
**.queue.packetCapacity = 100

[Config Exp3RoundRobinWithPolicingAndShaping] # Weighted Round Robin with Policing and Shaping

extends = Apps, Exp, PolicingAndShaping
**.R*.ppp[*].ppp.queue.typename = "DSQueue1" #Diffserv Queue
**.R*.ppp[*].ppp.queue.packetCapacity = 100 #-1
**.R*.ppp[*].ppp.queue.*.packetCapacity = 100
**.R*.ppp[*].ppp.queue.wrr.weights = "10 40 5 1 1"
```