

# Differentiated Services

## Simulations using Omnet++

CSC 467: Switching, Network Traffic and Quality of Service

Jakob Roberts - v00484900

# Table of Contents

<b>1 Introduction</b>	<b>3</b>
1.1 Differentiated Services - Definition	3
1.2 DiffServ Simulation Environment	5
<b>2 Experimental Models</b>	<b>6</b>
2.1 Test 1	7
2.2 Test 2	11
2.3 Test 3	12
<b>3 Conclusion</b>	<b>15</b>
<b>4 References</b>	<b>16</b>

# 1 Introduction

With further connected devices like cell phones, tablets, and products in the Internet of Things, managing all the new traffic has become increasingly critical in order to provide the quality of service users would expect. As such, some network models will be simulated using OMNeT++ to evaluate their performance and capabilities of rendering acceptable quality of service. With these simulations, an accurate prediction can be made as to which would be the more beneficial service to implement.

## 1.1 Differentiated Services - Definition

Simply put, the computer networking architecture *Differentiated Services* (DiffServ) is designed to provide quality of service on modern networks by addressing problems associated with traffic classification, simplicity, and scalability. It has the capabilities to differentiate between critical and non critical traffic and provide a lower latency to the important services first.

By replacing the Type of Service in the ipv4 header with the Differentiated Services Code Point (DSCP), the traffic can be properly marked/classified. When passed through a router that is capable of

handling a DiffServ packet, each marked packet will be passed through based on its classified per-hop behaviour (PHB) that defines where the packet is to be forwarded and its traffic type association. All routers on

a network will be aware of the classification types of the packets being sent. A simple example of a type of DiffServ structure is shown in Figure 1.



Figure 1: Simple DiffServ Representation

When the packet's contents are evaluated by an edge router of the network, DiffServ's 6-bit DSCP in the 8-bit DiffServ field are used to determine the packet's classification. Networks can have up to 64 ( $2^6$ , 6-bit DSCP) different traffic classifications based on the DSCP's contents. In typical network structures, a set of the following 5 PHB classifications are commonly used:

- |                                      |  |
|--------------------------------------|--|
| <b>Default PHB</b>                   | Usually assigned for the best-effort traffic. This is for all the traffic that does not meet any pre-defined criteria.   |
| <b>Expedited Forwarding (EF) PHB</b> | Reserved for traffic requiring a low loss, latency, and jitter. Used for voice, video, and anything real-time.   |
| <b>Voice Admit PHB</b>               | Has similar characteristics to EF PHB but is strictly reserved for traffic involving voice communications.   |
| <b>Assured Forwarding (AF) PHB</b>   | Split into 12 different categories: 4 classes and 3 different drop priorities. This class provides assurance that the packet is delivered given that a defined rate is adhered to. If the defined rate is violated, congestion can occur and the assurance is nullified.   |
| <b>Class Selector PHBs</b>           | This classification is intended strictly to allow for backwards compatibility for network devices using the Precedence field to determine packet preference. Even if a DiffServ router receives a packet from a non-DiffServ router, it can still manage to understand the given information using the class selector. |

As Voice Admit is a subset of EF, for the sake of testing, it will be omitted as a significant category. Having the set of classes to differentiate packet importance allows traffic to be properly distributed and less disturbance is noticed among users.

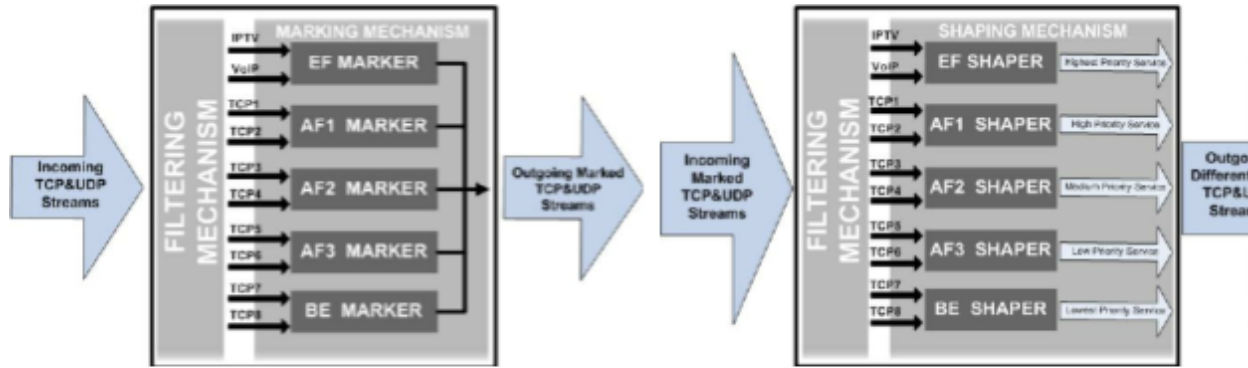


Figure 2: DiffServ Edge Router

Figure 3: DiffServ Core Router

In the DiffServ network there are two types of routers, edge routers that take care of packet marking, and core routers that take care of packet distribution within the subnet. After the edge routers on the network have done the hard work of classifying the packets to their type, the core router functionality can be limited to applying the PHB treatment to packets based on their given marking. As shown in Figure 2 [4], the edge router takes in the packets and filters them to match the appropriate marker. In Figure 3 [4], the core router takes the already marked packets and aligns them with their appropriate priority.

## 1.2 DiffServ Simulation Environment

These following simulations were performed using OMNeT++ V5.0. OMNeT++ is an extensible, modular, and component-based C++ simulation library and framework used for building network simulators. [2] It provides a component architecture for models where modules are created then used in simulations to test potential networking environments.

## 2 Experimental Models

A main network topology was used and is detailed in Figure 4. The topology consists of 8 hosts and 6 routers: Hosts 1, 2, 3, and 4 are information sources, Hosts 5, 6, 7, and 8 are destinations, Routers 0, 1, 4, and 5 are the edge routers, and Routers 2 and 3 are the core routers.

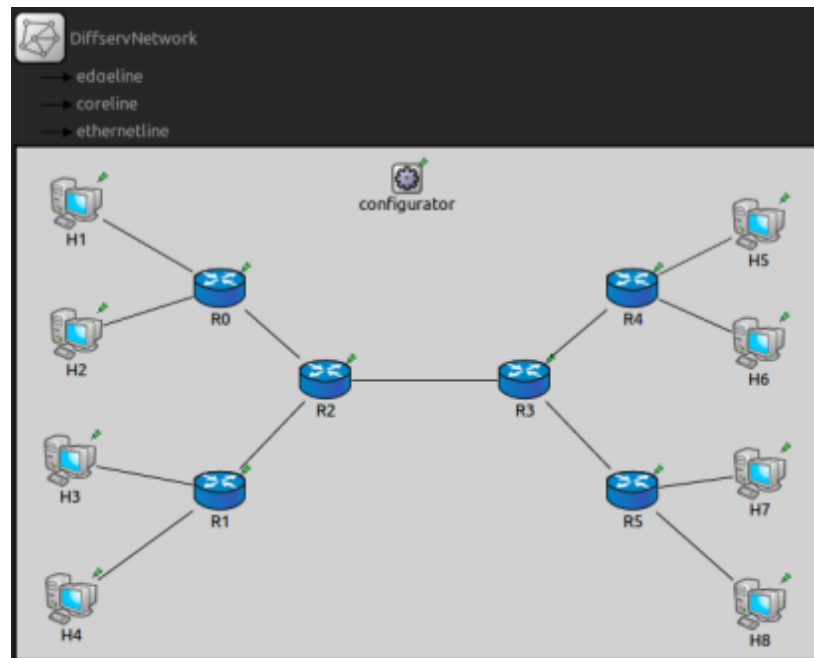


Figure 4: DiffServ Testing Network Topology

The topology is used for three different experiments with the DiffServ network design. The first experiment uses drop tail queues for all input packets and will avoid congestion by removing any overflow. The second experiment reduces classifications to just AF and BE, and uses a FIFO queue with a red drop after classification in order to avoid erroneous congestion. The third experiment will implement TCP and RED enabled routers.

## 2.1 Test 1

The queuing system for this test is in OMNeT++ as DSQueue1 and is implemented with the network topology listed above. In this scenario, after the initial marking/classification has been done between EF, AF, and BE groups, the AF and BE groups are then put into a weighted round robin and are then fed with the EF into another scheduler. This final scheduler takes the EF entries first then the rest afterwards. All of the queues post marking have a drop-tail queue implemented as their means for congestion control.

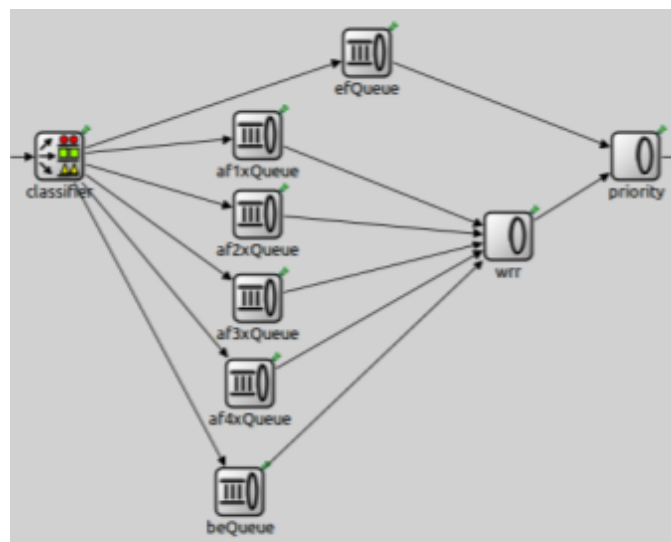


Figure 5: DSQueue1

The layout for the queueing and priority of the system can be seen in Figure 5 above. The simulation uses both voice and video data packets that are being sent between hosts. Figure 6 lists the configuration details for the entire test set.

```

1 [General]
2 network = DiffServNetwork
3 sim-time-limit = 1250s
4 #cmdenv-express-mode = false
5
6 **result-recording-modes =
7 **scalar-recording = false
8 debug-statistics-recording = true
9
10 # default queues
11 **queueType = "DropTailQueue"
12 **queue.frameCapacity = 100
13 **queue.dataQueue.frameCapacity = 100
14
15 [Config Apps]
16 **H[1..8].numUdpApps = 2 # 0 = voice, 1 = video
17
18 # voice streaming
19 **H[1..4].udpApp[0].typename = "UDPBasicBurst"
20 **H1.udpApp[0].destAddresses = "H5"
21 **H2.udpApp[0].destAddresses = "H6"
22 **H3.udpApp[0].destAddresses = "H7"
23 **H4.udpApp[0].destAddresses = "H8"
24 **H[1..4].udpApp[0].chooseDestAddrMode = "once"
25 **H[1..4].udpApp[0].destPort = 2000
26 **H[1..4].udpApp[0].startTime = uniform(1s,2s)
27 **H[1..4].udpApp[0].stopTime = 1200s
28 **H[1..4].udpApp[0].messageLength = 172B # 160B voice + 12B RTP header
29 **H[1..4].udpApp[0].burstDuration = exponential(0.352s)
30 **H[1..4].udpApp[0].sleepDuration = exponential(0.650s)
31 **H[1..4].udpApp[0].sendInterval = 20ms
32
33 **H[5..8].udpApp[0].typename = "UDPBasicAPP"
34 **H[5..8].udpApp[0].localPort = 2000
35 **H[5..8].udpApp[0].delayLimit = 0ms
36
37 **H[5..8].udpApp[0].destAddresses = ""
38 **H[5..8].udpApp[0].chooseDestAddrMode = "once"
39 **H[5..8].udpApp[0].destPort = 0
40 **H[5..8].udpApp[0].messageLength = 0B
41 **H[5..8].udpApp[0].burstDuration = 0s
42 **H[5..8].udpApp[0].sleepDuration = 0s
43 **H[5..8].udpApp[0].sendInterval = 0ms
44
45 # video streaming
46 **H[1..4].udpApp[1].typename = "UDPBasicAPP"
47 **H[1..4].udpApp[1].destPort = 1000
48 **H[1..4].udpApp[1].startTime = uniform(1s,2s)
49 **H[1..4].udpApp[1].stopTime = 1200s
50 **H[1..4].udpApp[1].sendInterval = 40ms
51 **H[1..4].udpApp[1].messageLength = 500B
52 **H1.udpApp[1].destAddresses = "H5"
53 **H2.udpApp[1].destAddresses = "H6"
54 **H3.udpApp[1].destAddresses = "H7"
55 **H4.udpApp[1].destAddresses = "H8"
56
57 **H[5..8].udpApp[1].typename = "UDPSink"
58 **H[5..8].udpApp[1].localPort = 1000
59
60 #
61 # Experiment 1
62 #
63
64 [Config Exp1Setup]
65 **edgeDataRate = 500kbps
66 **coreDataRate = 500kbps
67
68 **R[eth*].ingressTCType = "TC1"
69 **ingressTC.numClasses = 4
70 **ingressTC.classifier.filters = xmldoc("filters.xml", "//experiment[@id='default']")
71 **ingressTC.marker.dsccps = "AF11 AF21 AF31 AF41 BE"
72
73 **R[ppp*].queueType = "DSQueue1"
74 **R[ppp*].queue.frameCapacity = 100
75
76 # statistics
77 **H[1..4].udpApp[*].sentPk.result-recording-modes = count
78 **H[5..8].udpApp[*].rcvdPk.result-recording-modes = count
79 **H[5..8].udpApp[*].endToEndDelay.result-recording-modes = vector # for computing median
80 **R2.ppp[2].**Queue.rcvdPk.result-recording-modes = count
81 **R2.ppp[2].**Queue.dropPk.result-recording-modes = count
82 **R2.ppp[2].**Queue.queueLength.result-recording-modes = timeavg
83 **R2.ppp[2].**Queue.queueingTime.result-recording-modes = vector # for computing median
84 **udpApp[*].sentPk.scalar-recording = true
85 **udpApp[*].rcvdPk.scalar-recording = true
86 **udpApp[*].endToEndDelay.scalar-recording = true
87 **R2.ppp[2].**Queue.scalar-recording = true
88 **afQueue.*.scalar-recording = true
89
90 [Config Exp1]
91 extends = Apps, Exp1Setup
92 **R[ppp*].queue.wrr.weights = "10 7 5 2 0"

```

Figure 6: Application and Experiment 1 configurations

The essentials to the configuration state that the voice packets being sent are in UDP burst mode (UDPBasicBurst) and the video packets are using UDPBasicAPP.

Some notable information that need to be explained from the configuration criteria will be explained in the following table:

<b>udpApp[x]</b>	Data type being sent, 0 for voice and 1 for video in this scenario.
<b>sim_time</b>	The allotted time for the simulation to be given to run.
<b>queue.frameCapacity</b>	The size of the drop-tail queues in the scenario.
<b>H{x...y}.udpApp[z].typename</b>	This defines the data transmission type (eg. UDPBasicBurst) for systems x to y using the data type z.
<b>H{x...y}.udpApp[z].startTime</b>	This defines the time in which the first burst of data from a source for systems x to y using the data type z.
<b>H{x...y}.udpApp[z].stopTime</b>	This defines the time where packets are stopping their sending from sources x to y using the data type z.

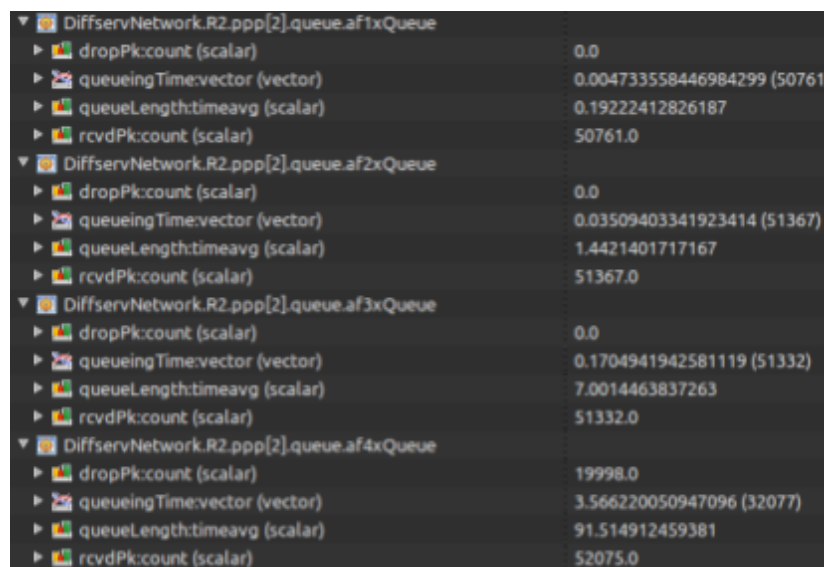


$H\{x...y\}.udpApp[z].messageLength$	This defines the data transmission size from systems x to y using the data type z.
$H\{x...y\}.udpApp[z].burstDuration$	This defines the burst duration from systems x to y using the data type z.
$H\{x...y\}.udpApp[z].sleepDuration$	This defines the time between bursts of information from systems x to y using the data type z.
$H\{x...y\}.udpApp[z].sendInterval$	This defines the sets the time between message bursts size from systems x to y using the data type z.

Weights:

AF1	10
AF2	7
AF3	5
AF4	2
BE	0

The 4 main source hosts sent voice and video data across the network to the 4 main destination host. None of the sent data used EF or BE and as such only data was really collected for the AF queues. Due to the start time being randomized to between 1 and 2 seconds for each host, there will be slight variations on each run of the test, but the results will all be very similar and will prove the same results under analysis. Figure 7 shows the results obtained from the first test.



DiffServNetwork.R2.ppp[2].queue.af1xQueue	
dropPk:count (scalar)	0.0
queueingTime:vector (vector)	0.004733558446984299 (50761)
queueLength:timeavg (scalar)	0.19222412826187
rcvdPk:count (scalar)	50761.0
DiffServNetwork.R2.ppp[2].queue.af2xQueue	
dropPk:count (scalar)	0.0
queueingTime:vector (vector)	0.03509403341923414 (51367)
queueLength:timeavg (scalar)	1.4421401717167
rcvdPk:count (scalar)	51367.0
DiffServNetwork.R2.ppp[2].queue.af3xQueue	
dropPk:count (scalar)	0.0
queueingTime:vector (vector)	0.1704941942581119 (51332)
queueLength:timeavg (scalar)	7.0014463837263
rcvdPk:count (scalar)	51332.0
DiffServNetwork.R2.ppp[2].queue.af4xQueue	
dropPk:count (scalar)	19998.0
queueingTime:vector (vector)	3.566220050947096 (32077)
queueLength:timeavg (scalar)	91.514912459381
rcvdPk:count (scalar)	52075.0

Figure 7: Test 1 results

From the test results, we can plainly see that the AF4 queue received all the drops compared to the other three AF queues.

We can calculate the approximate average throughput to each of the destination hosts by combining the received packet numbers for both voice and video connections and dividing it by the total simulation time to give a throughput value in KBPS.

<b>H5</b>	20800+29961 /1250=	40.6088 KBPS
<b>H6</b>	21412+29954 /1250=	41.0928 KBPS
<b>H7</b>	21379+29954 /1250=	41.0664 KBPS
<b>H8</b>	10560+21517 /1250=	27.2616 KBPS

Using the weights for each of the AF categories we can calculate their individual data type transfer weights:

$\lambda[x]$ (data type) = $1000/(\text{send interval}) * (\text{burst duration}) * (\text{packet length}) * (\# \text{ of bits})$
$\lambda[0] = 1000 / 20 * .352 * 172 * 8 = 24217 \text{ bits/sec for voice}$
$\lambda[1] = 1000 / 40 * 572 * 8 = 100,000 \text{ bits/sec for video}$

From the values we received we could also calculate  $\rho$  for each of the AF queues:

Knowing: $\lambda=124217$ , $\rho = \lambda / \mu$ , $\mu = \text{<priority> / <total\_priorities> * <data\_rate>}$		
AF1	$\mu = 10 / 24 * 500 \text{ kbps} = 208\text{kbps}$	$\rho = \lambda / \mu = 124217 / 208000 = 0.6$
AF2	$\mu = 7 / 24 * 500 \text{ kbps} = 146\text{kbps}$	$\rho = \lambda / \mu = 124217 / 146000 = 0.85$
AF3	$\mu = 5 / 24 * 500 \text{ kbps} = 104\text{kbps}$	$\rho = \lambda / \mu = 124217 / 104000 = 1.2$
AF4	$\mu = 2 / 24 * 500 \text{ kbps} = 42\text{kbps}$	$\rho = \lambda / \mu = 124217 / 42000 = 3$

## 2.2 Test 2

The second test involved using another queueing scheme in OMNeT++ called DSQueue2. This scenario again uses the above network topology, but the queue runs off of a single AF FIFO queue that drops any red classified packets. The queueing system can be seen in the following Figure 8. It is much simpler than the queue used in the first set of tests.

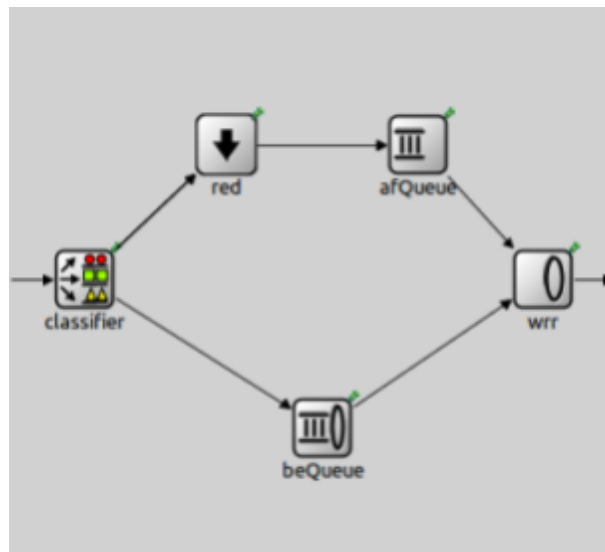


Figure 8: DSQueue2 layout

Many of the same settings were used from the first test but some are changed and are shown in Figure 8. The main things that were changed are the Edge data rate, the voice data type name, the sendInterval for computer 4, the typename for computer 8, and applying a few more filters.

```

[Config Exp2]
**.edgeDatarate = 10Mbps
**.coreDatarate = 500kbps

**.H4.numUdpApps = 1
**.H4.udpApp[0].typename = "UDPBasicApp"
**.H4.udpApp[0].destPort = 1000
**.H4.udpApp[0].startTime = uniform(1s,2s)
**.H4.udpApp[0].stopTime = 1200s
**.H4.udpApp[0].sendInterval = ${iaTime=400ms,200ms,133ms,100ms,80ms,67ms,57ms,50ms,44ms,40ms} # rates: 10kbps,20kbps,...,100kbps
**.H4.udpApp[0].messageLength = 500B-20B-8B
**.H4.udpApp[0].destAddresses = "H8"

**.H8.numUdpApps = 1
**.H8.udpApp[0].typename = "UDPSink"
**.H8.udpApp[0].localPort = 1000

**.R7.eth[*].ingressTCType = "TC1"
**.ingressTC.numClasses = 4
**.ingressTC.classifier.filters = xmldoc("filters.xml", "//experiment[@id='default']")

**.R7.ppp[*].queueType = "DSQueue2"
**.R7.**.beQueue.frameCapacity = 100
**.R7.**.red.minths = "60 30 10"
**.R7.**.red.maxths = "100 70 40"
**.R7.**.red.maxps = "0.40 0.70 1.00"
**.R7.**.wrr.weights = "1 1"

# statistics
**.H{1..3}.udpApp[*].sentPk.result-recording-modes = count
**.H{5..7}.udpApp[*].rcvdPk.result-recording-modes = count
**.R2.ppp[2].queue.afQueue.queueLength.result-recording-modes = timeavg,vector
**.udpApp[*].sentPk.scalar-recording = true
**.udpApp[*].rcvdPk.scalar-recording = true
**.afQueue.*.scalar-recording = true

[Config Exp21]
extends = Exp2, Apps
**.ingressTC.marker.dscps = "AF11 AF12 AF13 AF11 BE"

```

Figure 9: Test 2 configurations

As in the DSQueue2 representation there is only 1 AF so there were no predefined weights to be handed out. With these increased specifically in the edge data rate, it seemed that the queue did get overloaded at some point as it received a total count of 316360 packets, but all of the packets seemed to transfer without a problem because the sending and receiving destination packet counts are equal across the network.

## 2.3 Test 3

For the final test, a new queue was created (DSQueue3) in order to accommodate TCP applications and RED-drop enabled routers. Yet again we are using the same original network topology, but each of the AF entries and the EF entry have been switched out from drop-tail to FIFO queues and have had a RED-drop attached to it. The newly created DSQueue3 can be seen in the following Figure 10.

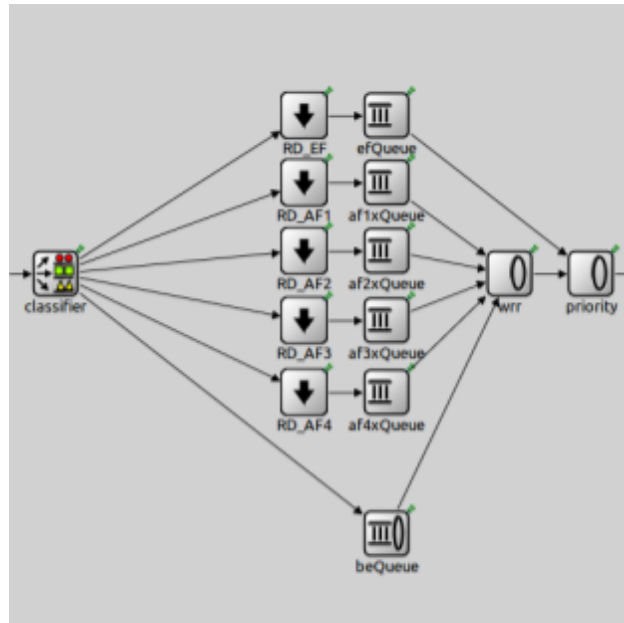


Figure 10: DSQueue3 for TCP

The TCP application that was created in OMNeT++ was essentially TCP file transfer services that had the same sources as before H1-H4 transferring to their respective destinations H5-H8. The simulated file transfer was 2MB in size with rates of 100KBPS on the edges and 10KBPS on the core edges. The simulation was run for 1250s and further information on the configuration parameters can be seen in Figure 11. The experiment was run a single time as there should be little expected variance between results.

The results were conclusive with the predictions associated with the weights assigned to each of the AF PHBs. The core network had 1/10th the data transfer rate as compared to the edge links and this meant that the core became very congested after transfers began between nodes and caused a significant amount of packet loss.

During the packet transfer process from the respective systems, there was significant packet loss that indicated that the RED system was doing its job by dropping packets that were not able to be transferred without further increasing congestion.

```

1 [General]
2 network = DiffservNetwork
3 sim-time-limit = 1250s
4 #cmdenv-express-mode = false
5 #**.result-recording-modes =
6 #**.scalar-recording = false
7 #debug-statistics-recording = true
8
9 # default queues
10 #**.queueType = "DropTailQueue"
11 **.queue.frameCapacity = 1000
12 #**.queue.dataQueue.frameCapacity = 100
13
14 [Config Apps3]
15 **.H{1..8}.numTcpApps = 1
16 **.H{1..4}.tcpApp[*].typename = "TCPSessionApp"
17 **.H{1..4}.tcpApp[0].active = true
18 **.H{1..4}.tcpApp[0].localAddress = ""
19 **.H{1..4}.tcpApp[0].localPort = -1
20 **.H1.tcpApp[0].connectAddress = "H5"
21 **.H2.tcpApp[0].connectAddress = "H6"
22 **.H3.tcpApp[0].connectAddress = "H7"
23 **.H4.tcpApp[0].connectAddress = "H8"
24 **.H{1..4}.tcpApp[0].connectPort = 1000
25 **.H{1..4}.tcpApp[0].tOpen = exponential(0.1s)
26 **.H{1..4}.tcpApp[0].tSend = 0.4s
27 **.H{1..4}.tcpApp[0].sendBytes = 2000000B
28 **.H{1..4}.tcpApp[0].sendScript = ""
29 **.H{1..4}.tcpApp[0].tClose = 25s
30 **.H{5..8}.tcpApp[*].typename = "TCPSinkApp"
31 # Experiment 3
32 [Config Exp1Setup]
33 **.edgeDataRate = 100kbps
34 **.coreDataRate = 10kbps
35
36 **.R?.eth[*].ingressTCType = "TC1"
37 **.ingressTC.numClasses = 4
38 **.ingressTC.classifier.filters = xmldoc("filters.xml", "//experiment[@id='default']")
39 **.ingressTC.marker.dscps = "AF11 AF21 AF31 AF41 BE"
40
41 **.R?.ppp[*].queueType = "DSQueue3"
42 **.R?.ppp[*].queue.frameCapacity = 1000
43
44 [Config Exp3]
45 extends = Apps3, Exp1Setup
46 **.R?.ppp[*].queue.wrr.weights = "10 7 5 2 0"
47 **.R?.**.RD**.minths = "5 10 20 30 50"
48 **.R?.**.RD**.maxths = "15 25 30 50 100"
49 **.R?.**.RD**.maxps = "1 1 1 1 1"

```

Figure 11: TCP test configuration

### 3 Conclusion

OMNeT++ is a very powerful simulation tool that can be used to visualize and test a seemingly infinite network scenarios. From using this application and doing testing, the simulations performed seemed to produce accurate results, although there was some randomization, it is likely attributed to the varied start time of transfers in the UDP tests. In an ideal circumstance to simulate a more realistic real life scenario, the tests would be performed with a larger base of computers on the network, there would be longer and more strenuous tests involved, but the thresholds associated with packet loss would not be as strict. I feel that I only scratched the surface of the capabilities of OMNeT++ and I still have a lot to learn about the powerful tool, however these tests were able to provide a solid introduction to the way the software works.

## 4 References

- [1] Differentiated Services Definition and information [http://www.cisco.com/c/en/us/td/docs/ios/12\\_2/qos/configuration/guide/fqos\\_c/qcfdfsrv.html](http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfdfsrv.html)
- [2] OMNeT++ <https://omnetpp.org/>
- [3] Image slide 11 <https://www.slideshare.net/ThamerAlamery/intserv-diffserv>
- [4] Edge + Core Routers [https://www.researchgate.net/figure/235245894\\_fig1\\_Figure-2-Overall-architecture-of-the-QoS-aware-DVBIP-infrastructure](https://www.researchgate.net/figure/235245894_fig1_Figure-2-Overall-architecture-of-the-QoS-aware-DVBIP-infrastructure)





