


32603	Deliverable D4.4.2v1	
-------	----------------------	---

Project Number:	<b>IST-2001-32603</b>
Project Title:	<b>6NET</b>
CEC Deliverable Number:	<b>32603/GRNET/DS/4.4.2/A1</b>
Contractual Date of Delivery to the CEC:	28 February 2004
Actual Date of Delivery to the CEC:	6 April 2004
Title of Deliverable:	Report on IPv6 QoS tests
Work package contributing to Deliverable:	WP4
Type of Deliverable*:	R
Deliverable Security Class**:	PU
Editor:	Athanassios Liakopoulos
Reviewer:	Martin Dunmore
Contributors:	Ch. Bouras, A.Gkamas, D.Pripmas, K.Stamos, J.Orozco, D.Ros, V.Siris, G.Fotiadis, R.Korpela, R.Duncan, A.Sevasti, G. Velde, R.Roth.

\* Type: P - Prototype, R - Report, D - Demonstrator, O - Other

\*\* Security Class: PU- Public, PP – Restricted to other programme participants (including the Commission), RE – Restricted to a group defined by the consortium (including the Commission), CO – Confidential, only for members of the consortium (including the Commission)

### **Abstract:**

This document mainly presents the first phase QoS tests performed in 6NET. Most of the partners analysed the operation and interaction of QoS mechanisms, e.g. shaping or queuing, in IPv6 testbeds or production networks and validated the performance of QoS implementations for different platforms. Finally, the last section presents the second phase of QoS tests that are planned to be performed in 6NET and describes the different Class of Services that will be supported at the core.

**Keywords: QoS, IPv6, DiffServ, Policing, Shaping, IP Premium, LBE, EF, AF.**

---

## Executive Summary

This document presents the first phase QoS tests performed in 6NET. Most of the partners analysed the operation and interaction of QoS mechanisms, e.g. shaping or queuing, in IPv6 testbeds or production networks and validated the performance of QoS implementations for different platforms, such as commercial routers from different vendors.

The first set of tests investigates how it is possible to implement a QoS service that is suitable for real time applications. The QoS service under testing was based on DiffServ EF (expedited forwarding) PHB, which provides strict priority to properly marked packets. The second set of tests investigates the “intra-class fairness” for AF (assured forwarding) traffic, i.e. how does (free) link capacity is fairly shared among different flows in a (partially) congested network. The third set of tests investigates the combined operation and interaction of different QoS mechanisms, such as traffic policing, traffic shaping and class-based queuing. These mechanisms are usually applied on edge routers at the interface towards the network provider network. Therefore, inappropriate tuning of these mechanisms may severely affect services provided. The last set of tests investigated the performance achieved by data transfer applications that used Less than Best Effort (LBE) services in a production network.

Finally, chapter 6 includes the plans for the second phase of QoS tests that are going to be performed during the project. As scheduled, 6NET (access and) core network will adhere to the Differentiated Services framework and will provide at least three different class of services. A high priority service based on EF PHB will be provided to a small portion of network traffic, especially for traffic generated with real time applications. Also, BE and LBE services will be used with elastic applications that do not require delay or bandwidth guarantees. The most significant QoS mechanisms, such as traffic policing, will be tested in the 6NET WAN environment and “objective” tests will be performed with traffic generators. Finally, further tests will be performed with real time applications, e.g. videoconference, and end-users will be asked to “subjectively” score the QoS performance services in the 6NET network.

## Table of Contents

1	Introduction.....	6
2	Computer Technology Institute (CTI): Providing QoS for Real Time Applications .....	6
2.1	Introduction.....	6
2.2	CTI QoS Experience .....	7
2.3	Description of the Experimental Procedure .....	8
2.3.1	Introduction.....	8
2.3.2	Investigation of prioritization mechanism .....	9
2.3.3	Experimental testing for real time applications .....	13
2.3.4	Investigation of WRED mechanism .....	15
2.3.5	Investigation of policing profile.....	17
2.4	Next Steps .....	17
3	GET/ENST-Bretagne: Intra-class fairness in DiffServ/AF over IPv6.....	18
3.1	Introduction.....	18
3.1.1	Overview of Assured Forwarding.....	18
3.1.2	Intra-class fairness in AF .....	18
3.1.3	Objective .....	19
3.2	Intra-class fairness for TCP aggregates .....	19
3.2.1	Overview.....	19
3.2.2	Test platform.....	20
3.2.3	Specification.....	21
3.3	Results.....	23
3.3.1	Bandwidth distribution.....	23
3.3.2	Fairness index .....	26
3.4	Conclusions and future work .....	28
4	University of Crete (UoC): A Testbed Investigation of QoS Mechanisms for Supporting SLAs in IPv6.....	28
4.1	Introduction.....	28
4.2	Testbed analysis .....	29
4.3	Traffic Generators .....	29
4.3.1	Iperf 1.7.0 for Linux.....	29
4.3.2	A Simple Example .....	30
4.4	Quality of Service mechanism analysis .....	30

4.4.1	Token Bucket .....	30
4.4.2	Cisco Traffic Policing .....	31
4.4.3	Cisco Generic Traffic Shaping (GTS) .....	32
4.4.4	Linux Traffic Policing.....	32
4.4.5	Linux Traffic Shaping.....	33
4.5	Experimental Results .....	33
4.5.1	Policing tests .....	33
4.5.2	Interaction of Policing and Shaping.....	35
4.5.3	Interaction of Policing and Class Based Queuing.....	40
5	CSC /Funet: Testing LBE in IPv6 Network.....	41
5.1	Previous work in IPv6/QoS.....	41
5.2	Report on testing of the LBE concept in Funet production network .....	42
5.2.1	The goal of this test.....	42
5.2.2	Topology .....	42
5.2.3	Equipment .....	43
5.2.4	Test procedure.....	43
5.2.5	Part I: Sources .....	44
5.2.6	Part II: Sources.....	44
5.2.7	Part III: Sources .....	44
5.3	Conclusions.....	45
5.4	Future work.....	45
6	6NET Quality of Service Framework .....	45
6.1	Introduction.....	45
6.2	The DiffServ QoS Architecture .....	46
6.3	6NET QoS Model .....	47
6.4	Class Specifications .....	48
6.4.1	IP Premium - Expedited Forwarding (EF).....	48
6.4.2	Default/Best Effort (BE).....	48
6.4.3	Less than Best Effort (LBE).....	48
6.5	QoS Semantics – DSCP values.....	48
6.6	Traffic Metrics .....	49
6.7	Application Profiles .....	50
6.8	Monitoring and Measurement Requirements.....	50
7	References.....	52

32603	Deliverable D4.2.2v1	<i>6net</i>
-------	----------------------	-------------

8	Appendix A: 6NET Quality of Service Testing Plan.....	53
8.1	Introduction.....	53
8.2	Implementation of Framework.....	53
8.3	Traffic generators.....	53
8.4	Passive Monitoring .....	55
8.5	Testing QoS mechanisms.....	56
8.5.1	Marking.....	56
8.5.2	Traffic Policing .....	56
8.5.3	Shaping .....	57
8.6	Test with non-BE Traffic Classes .....	57
8.6.1	Premium/Expedited Forwarding.....	57
8.6.2	Less Than Best Effort (LBE) .....	58
8.7	Application Tests .....	58
8.8	Schedule.....	59
8.9	6NET partners involved to the tests.....	59
9	Appendix B: Router configuration in LBE tests.....	60

## 1 Introduction

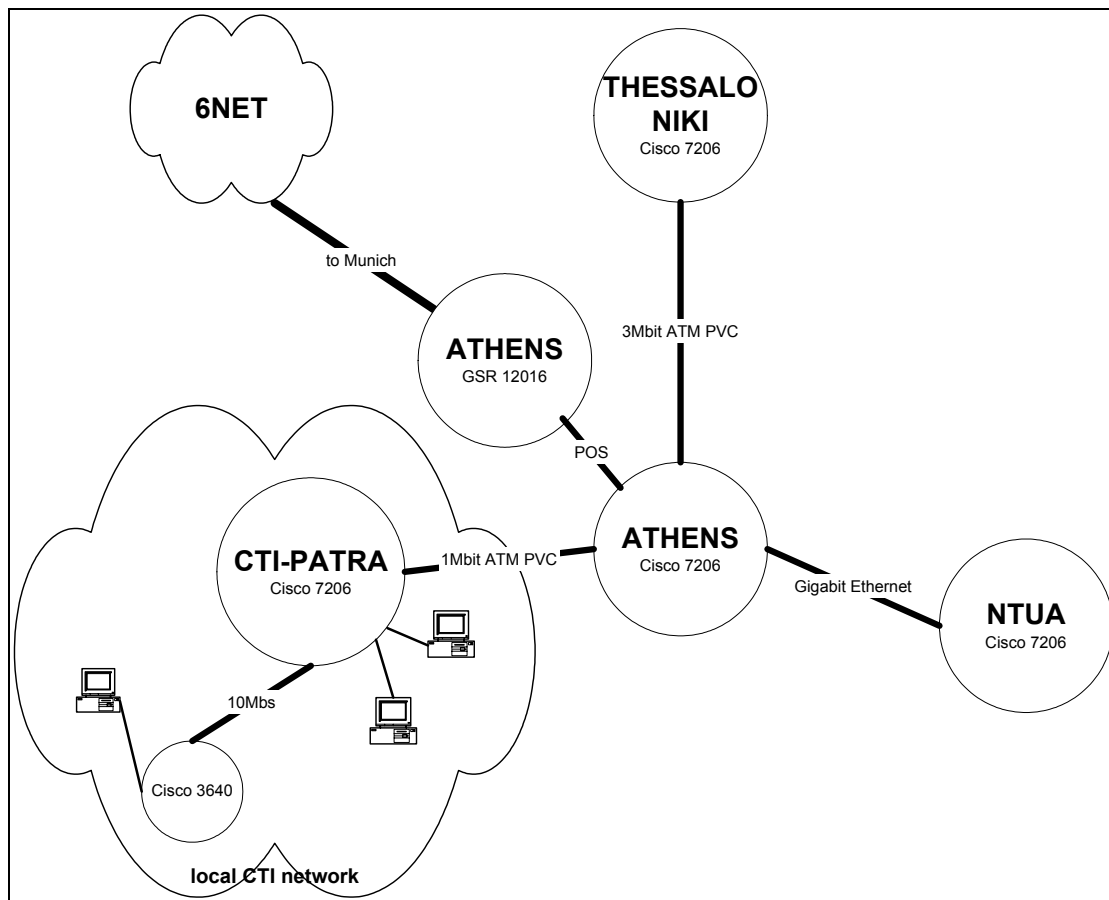
This document presents the results of the QoS tests that have been performed in the concept of the Work Package 4 in 6NET project. All these tests have been performed by individual partners on their small-scale internal networks (testbeds). Particularly, chapter 2 describes tests performed by Computer Technology Institute (CTI) that investigated how it is possible to implement a QoS service that is suitable for real time applications. Chapter 3 presents tests performed by GET/ENST Bretagne that investigated the “intra-class fairness” for AF (assured forwarding) traffic. Furthermore, chapter 4 refers to tests performed by University of Crete’s QoS that investigated the combined operation and interaction of different QoS mechanisms, such as traffic policing, traffic shaping and class-based queuing. Chapter 5 presents the experiments performed by CSC/Funet with Less than Best Effort (LBE) service in their production network.

Finally, chapter 6 describes the QoS Framework that will be implemented and tested in 6NET’s core network. Initially, QoS services that will be supported in the core and access network are presented and their characteristics are discussed. At the end, in Appendix A, the deliverable includes the testing plan of the aforementioned QoS services.

## 2 Computer Technology Institute (CTI): Providing QoS for Real Time Applications

### 2.1 Introduction

Today the number of the Internet users is large and increases every day. The Internet traffic consists of different application’s traffic, which all has the same treatment by the network. This treatment causes many problems to some applications, like real time applications (for example videoconference, etc), because they are sensitive on parameters as delay, packet loss or jitter. The solution is the QoS techniques, which can provide specific guarantees to applications. We have performed some tests with DiffServ architecture on our local testbed, and the results are presented in this document. Figure 1 presents our local testbed within the Greek part of the 6NET network. At a later stage, we are interested on extending those experiments on the 6NET’s backbone network in cooperation with other partners.



**Figure 1: Greek part of 6NET Network Topology**

## 2.2 CTI QoS Experience

In this section our previous work on QoS and Diffserv architecture will be described. Our experience also covers Expedited Forwarding Services and Assured Forwarding services. It consists of theoretical work and a large number of simulation tests on a network simulator. The first objective was to study and describe the gold service (EF based service) on an IPv4 network. The traffic on the simulation tests consisted of gold traffic from various applications and background traffic. The latter corresponded to the real traffic on the Internet, and was treated by the best effort service. The simulator that had been used was the Network Simulator 2 (NS-2) [6] with some modifications and changes for our needs [4]. Moreover, our previous work includes tests on an AF based service, called relative.

On the testing scenarios have been used and tested several methods and they are summarized on the following points:

- Classification, using the DSCP field of IPv4 header
- Policing, using the Token Bucket Policy
- Action, all out of profile packets are dropped

- Queuing, have been used several methods, Priority Queuing, Weighted Fair Queuing (WFQ) or Modified Deficit Round Robin (MDRR-CISCO mechanism available on series 12000)
- Queue management, using RED and WRED method

Generally, our previous work [4] was concentrated on EF and AF based services and on testing them on a simulation environment.

## 2.3 Description of the Experimental Procedure

### 2.3.1 Introduction

Our main goal is to implement and test a QoS service that will be applied on IPv6 networks and will service flows from real time applications, giving them the appropriate QoS level that they need. The service is based on DiffServ architecture (expedited forwarding) and provides strict priorities to packets that are produced from real time applications. This service has been applied on a real testbed IPv6 network that has been created internally on CTI and is presented in Figure 1. The software version used by the routers is the CISCO IOS 12.2(13)T.

The QoS service that we implemented intends to provide prioritization to traffic coming from real time applications. Its operation is to classify the packets that belong to this application and use “a priority queue” for these. The other traffic on the router will be treated as it normally would, with best-effort service. In addition, a token bucket policy will be applied on the traffic that is produced by the real time application, in order to ensure that this traffic will not exceed pre-agreed characteristics (SLA agreements) and also prevent the background traffic from extra delay or packet loss. So, our work is focused on investigation and implementation of the QoS service. Next, we want to investigate the operation of the policing mechanisms on IPv6 networks and finally, we want to evaluate the performance of the whole service.

The service has been implemented using the Class Based Weighted Fair Queuing mechanism. This mechanism actually extends the classic Weighted Fair Queuing mechanism and can provide strict packet priority. The service follows the classic guidelines of the DiffServ architecture, so the packets are classified at the ingress point of the IPv6 domain. In addition, policing of the incoming traffic also takes place according to specific policing rules. In particular, the experiments have been performed using the CISCO [3] routers models 7206 and 3610, with various PCs connected to them. For the experimental scenarios we are trying to be as realistic as possible. We insert background traffic in the network with cross connect method that is a mix of TCP and UDP traffic, which is generated by the Iperf traffic generator [7]. This traffic is simply treated as best effort. In addition, we insert foreground traffic that simulates an aggregate of real time traffic. This traffic is also a mix of UDP traffic generated by a traffic generator and RTP traffic generated by an OpenH323 application, which has been ported to operate on IPv6 networks by CTI in the framework of WP5 of the 6NET project. The OpenH323 library is an Open Source implementation of the ITU H.323 teleconferencing protocol.

On the network devices, we have applied a policing and marking mechanism in order to mark the background and foreground traffic. In particular, we distinguish the background and foreground traffic with different access lists and mark them accordingly using for the foreground traffic the DSCP ef (pre defined in CISCO IOS) and for the foreground the DSCP value default (0). Next, the output interfaces of the network devices have been configured in order to send the packets that have been marked with DSCP ef with strict priority (using the LLQ mechanism). In addition, we have



configured a token bucket policy mechanism to be applied on packets that belong to foreground traffic. This mechanism is intended for distinguishing the foreground packets in case that they are arriving in the input interface with a faster rate than the pre-agreed rate.

### 2.3.2 Investigation of prioritization mechanism

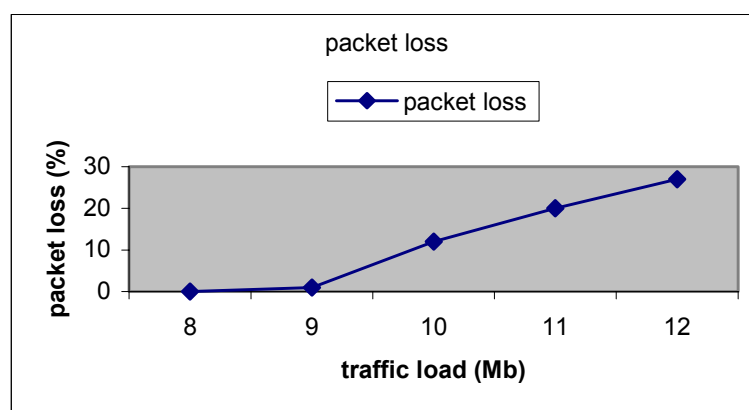
The first thing that we did was to investigate the operation of the classification and prioritization mechanisms. The experiments performed using the CISCO routers 7206 (CTI-Patra) and 3640 in CTI's internal network. The classification mechanism was implemented using ipv6 access lists and creating a policing class in the input interface of the router. According to the ipv6 access list that the packets belong to, the policy class assigns the DSCP values. Next, on the output interface of the router, we have configured a second policy class that gives strict priority to the packets that have been marked with DSCP value ef. In order to be sure that the configured mechanisms operate as expected, we have done a number of tests and their results are presented below.

First, we disabled the above mechanisms and sent background and foreground traffic to the network that had the following characteristics: Both foreground and background traffic were created with Iperf traffic generator and were UDP with average rate 12Mbps and 1.5 Mbps respectively. The backbone link is 10Mbit and in this case we expected to have many dropped packets. Actually, the results were 22% packet drops for both foreground and background traffic.

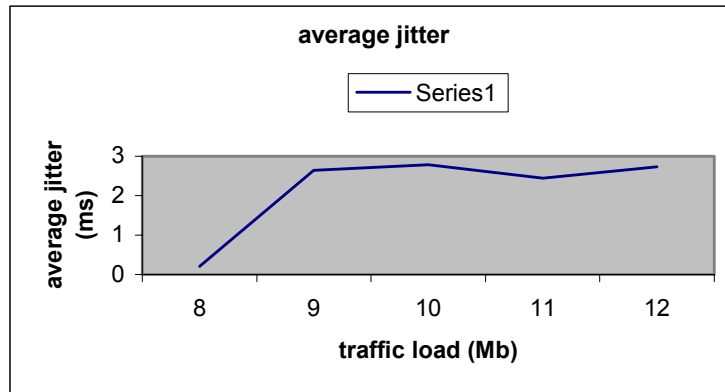
The next step was to investigate the network's behaviour when it only has best effort traffic (UDP or both UDP and TCP). At this stage, we performed several tests that are summarized on the following tables. First, we performed tests using only UDP traffic, generated by the Iperf traffic generator. The next table presents the experimental results.

Inserted traffic	Actual throughput	Packet loss	Average jitter
8M	7.98Mbps	0%	0.205ms
9M	8.89Mbps	0.99%	2.641ms
10M	8.82Mbps	12%	2.787ms
11M	8.74Mbps	20%	2.441ms
12M	8.66Mbps	27%	2.730ms

**Table 1: Best -effort experiments (only UDP)**



**Figure 2: The packet loss when network usage increases**



**Figure 3: The average jitter when network usage increases**

As we can see from the results, the actual throughput that can fill the backbone links is 8.80 Mbps. In addition, the packet loss and jitter when the network usage increases also increase exponentially as the graphics show.

Next, we repeated the same scenarios using both TCP and UDP traffic generated by the Iperf traffic generator. Similarly, the next table presents the results. At this point we should note that while for UDP the traffic generator tries to send the UDP traffic at the specified rate, for TCP it simply follows the TCP algorithm, thereby gradually increasing bandwidth consumption as long as there is no congestion, and rapidly dropping the transmission rate if the TCP sender notices packet losses. In addition, those experiments run for 3 minutes, so the results for the TCP are quite stable and accurate.

Inserted UDP traffic	Throughput UDP	Throughput TCP (average)
3Mbps	2.99Mbps	2.74Mbps
4Mbps	3.99Mbps	2.49Mbps

**Table 2 Best -effort experiments (UDP and TCP)**

According to the above results, we concluded that for almost 8Mbps traffic rate and above the network is fully congested. In addition, when we tried with more realistic traffic patterns (TCP and UDP traffic simultaneously) the traffic generator seems to be able to meet our traffic generation requirements. It produces the actual UDP traffic that earns “its piece of bandwidth” (according to its nature) and the remaining bandwidth is used by TCP.

The next step was to enable the classification and prioritization mechanism and try to test them for different traffic loads and scenarios. This stage was necessary in order to be convinced that the classification and prioritization mechanisms work efficiently. For this purpose, we performed a large number of experiments that are described in the following table.

Traffic load (bps)		Actual throughput (bps)		Packet loss		Av. Jitter	
B traffic	F traffic	B traffic	F traffic	B traffic	F traffic	B traffic	F traffic
10M (UDP)	500K (UDP)	8.49M	490K	15%	2%	2.441ms	3.515ms
9M (UDP)	500K (UDP)	8.48M	482K	5.5%	3.7%	2.699ms	6.421ms
12M (UDP)	250K (UDP)	8.49M	246K	29%	1.5%	2.678ms	4.449ms
9M (UDP)	250K (UDP)	8.58M	244K	7.6%	2.4%	2.864ms	4.333ms
8M (UDP)	250K (UDP)	7.98M	250K	0.0011%	0%	3.191ms	4.404ms
10M (UDP)	250K (UDP)	8.57M	239K	14%	4.5%	1.357ms	4.830ms
5M (UDP) + TCP	500K (UDP)	4.99M +2.49M	500K	UDP=0 TCPmany	0	2.627ms	6.420ms

**Table 3 Testing of classification and prioritization mechanism**

As Table 3 demonstrates, the prioritization mechanism seems to work efficiently. On each testing scenario the packet loss that the foreground experiences was significantly low, in contrast with the background traffic that suffers quite larger packet losses. Regarding the jitter metric, the table shows that the foreground traffic has larger average jitter. This point is actually due to the fact that the foreground traffic follows a longer path (it also crosses a part of CTI's production network). We have measured the average jitter for packets that have been generated from the same source as the foreground traffic, until the machines that insert the background traffic (this is the additional part of the path) and is almost 4.3ms. Taking into account this result, the jitter that the foreground traffic experiences is low. This is also a result that we expected as we have used the strict priority command on the policy map and this command uses the LLQ (Low Latency Queue) mechanism. This mechanism uses low latency queues for the classified packets that provide low delay and we therefore expect significantly lower jitter. Figure 4 and Figure 5 present the packet loss and jitter metrics for the foreground and background traffic for the above scenarios.

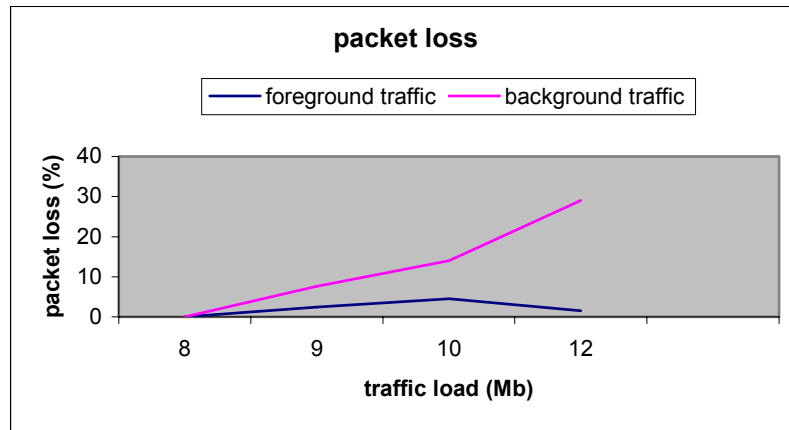


Figure 4: the packet loss for different traffic load (foreground=250K)

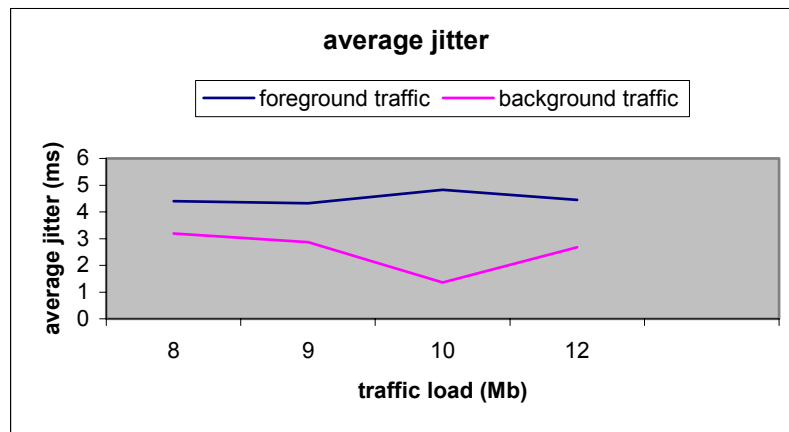


Figure 5: the average jitter for different traffic load (foreground=250K)

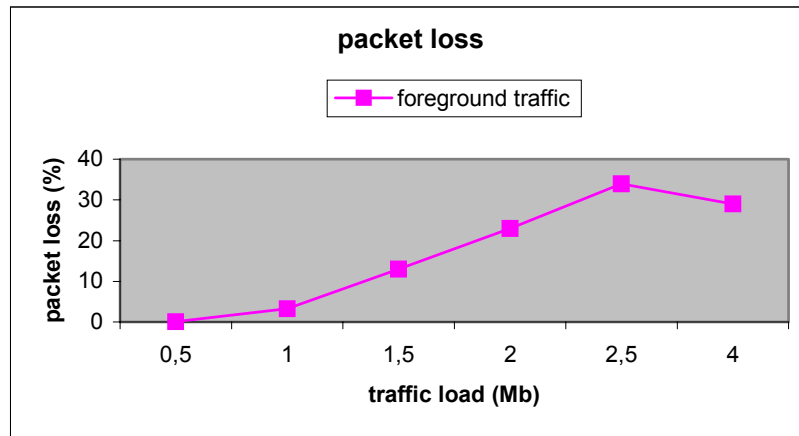
Afterwards, we performed some extra experiments where their goal was to investigate the behavior of the LLQ mechanisms in conditions where the percentage of bandwidth that foreground can use, is exceeded. In particular, as we had configured the policy map, the strict priority could use only 20% of the total bandwidth, in other words only 2Mbits. For this reason, we also performed the experiments.

Traffic load (bps)		Actual throughput (bps)		Packet loss		Average jitter	
Back-ground traffic	Fore-ground traffic	Back-ground traffic	Fore-ground traffic	Back-ground traffic	Fore-ground traffic	Back-ground traffic	Fore-ground traffic
5M	1.5M	4.99M	1.5M	0.0037	0.047	0.007ms	3.376ms
8M	0.5M	7.98M	500K	0.043%	0.047%	3.018ms	3.761ms
8M	1M	7.72M	990K	3.3%	1%	3.092ms	4.641ms

8M	1.5M	6.93M	1.43M	13%	4.6%	4.630ms	3.005ms
8M	2M	6.16M	1.8M	23%	10%	2.989ms	3.346ms
8M	2.5M	5.3M	2.16M	34%	13%	2.390ms	4.500ms
6M	4M	4.24M	3.16M	29%	21%	5.779ms	1.773ms
5M	2.5M	4.89M	2.5M	2.2%	0.094%	0.743ms	2.276ms

**Table 4 Experiments for testing the upper bound of strict priority**

Looking at the results from the above experiments, we conclude that when the rate exceeds the upper bound that the strict priority mechanism can guarantee, then its behavior seems to change. This can be shown better Figure 6.



**Figure 6: Strict priority' s behavior on upper bound**

The important difference that we can see on 4Mbps foreground traffic can be explained because in this experiment the background traffic load was lower than for all the other cases. According to the figure the packet loss increases proportionally from almost 1.5M and up.

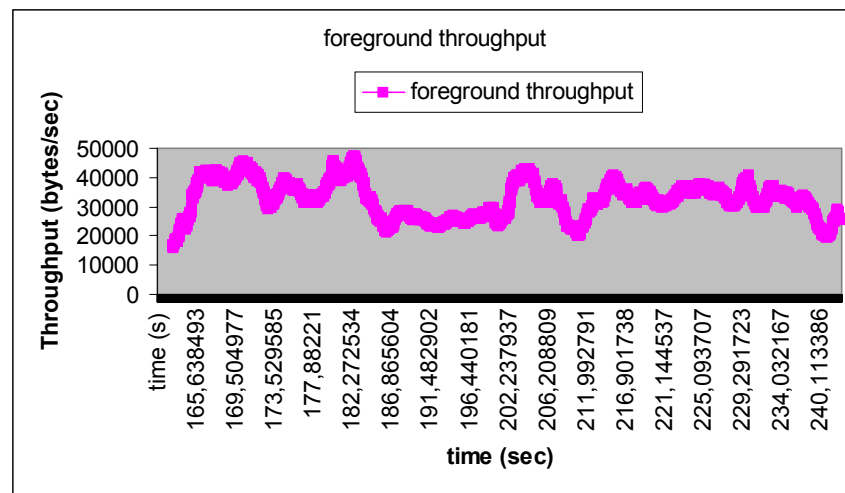
### 2.3.3 Experimental testing for real time applications

Finishing the above-described experimental stages, we have set up and evaluated the QoS mechanisms that can provide QoS guarantees. As we presented earlier, the used mechanism is the strict priority (which implements the Low Latency Queues). The reason why we used and investigated the behaviour of this mechanism is because it is extremely capable for real time applications that need low delay, packet loss and jitter.

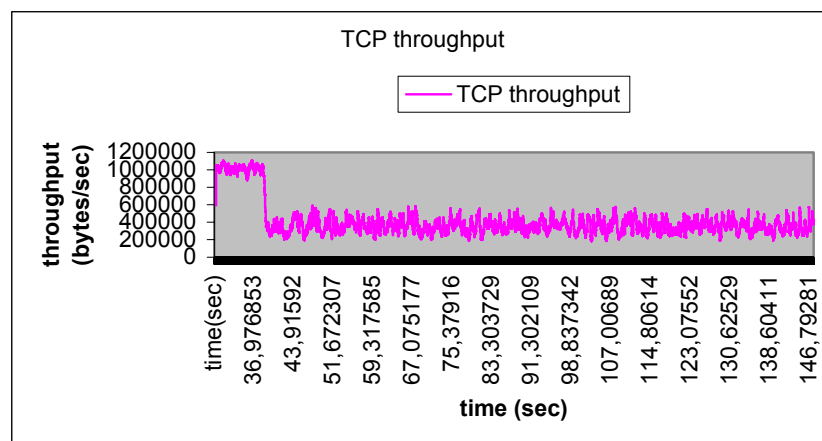
Following our goal, we now tried to simulate very realistic conditions of traffic load and tried to measure the performance of real time applications that experience the preferential treatment by those mechanisms. So, we used an application based on the OpenH323 library for implementing the following testing scenarios.

#### ➤ Testing scenario 1

Initially, we loaded the network with background traffic, generated by the Iperf traffic generator. The selected traffic is a mix of TCP and UDP traffic. At this point we should notice that we have started loading the network long enough before inserting the foreground traffic, in order for the TCP to obtain a stable state. Next we inserted foreground traffic that was generated by a videoconference (using the OpenPhone application based on the OpenH323 library). We performed this test for more than 5 minutes and recorded the packets that were exchanged. The background traffic was 5Mbps UDP traffic and also TCP traffic that tried to occupy as much bandwidth it could. Finally, the results showed that the UDP background traffic had only a few packets dropped. Similarly, the foreground traffic (OpenH323) had zero packet loss and excellent quality, which proves that the QoS mechanisms achieved their goal. In addition, the TCP background traffic was strangled by the strict priority mechanism. Figure 7 and Figure 8 present the throughput of the foreground traffic as well as the throughput of the TCP background traffic.



**Figure 7: Throughput of foreground traffic**

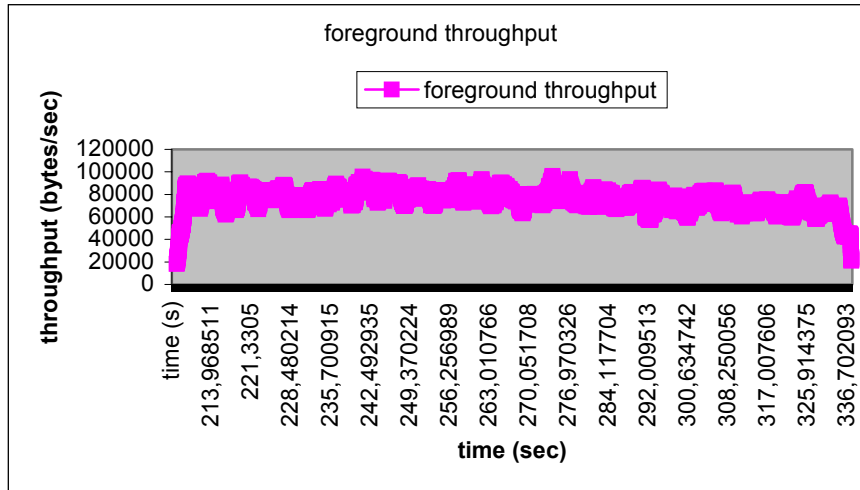


**Figure 8 Throughput of TCP background traffic**

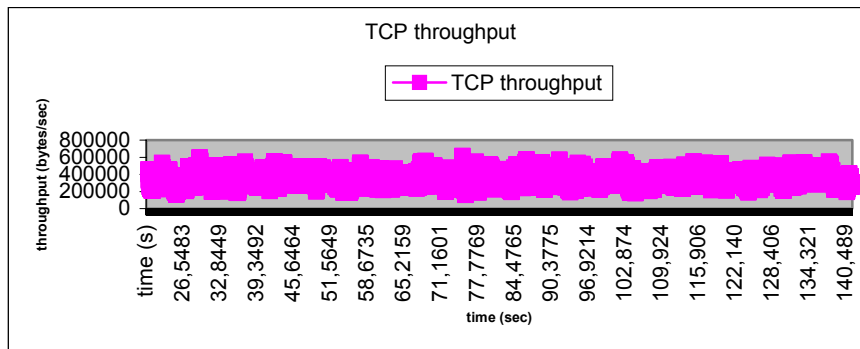
Looking at the figures, we can see that TCP initially sends many packets and after approximately 40 seconds it obtains its steady state. In addition, the foreground traffic has an average throughput of almost 300Kbps and very good video quality.

### ➤ Testing scenario 2

Similarly, we performed a second test with the same characteristics and traffic load. The only difference was that we also added at the foreground traffic extra UDP traffic (300Kbps), created by the Iperf traffic generator. The results were the same; the foreground traffic had almost zero packet loss (both UDP and RTP). In addition, the RTP traffic (OpenH323) had excellent video quality taking advantage of the operation of the strict priority mechanism (low latency queue). Figure 9 and Figure 10 present the throughput of the foreground traffic and TCP background traffic.



**Figure 9: Foreground throughput**



**Figure 10: TCP background throughput**

### 2.3.4 Investigation of WRED mechanism

Next, we tried to investigate the operation of the WRED mechanism. The WRED mechanism is a popular mechanism for congestion avoidance that has been tested extensively in IPv4. During our tests we tried this mechanism on our IPv6 domain. The goal of our test is first of all to test the correct operation of this mechanism on IPv6 and secondly to investigate its impact on the performance of the foreground traffic. At this stage we performed 2 separate tests that are described in the following sections.

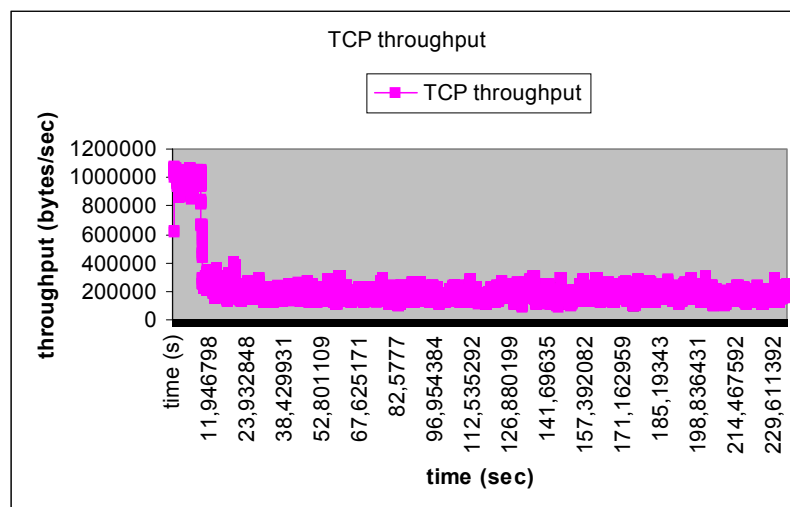
### ➤ Testing scenario 1

We set up the WRED mechanism in order to be applied on the background traffic using as thresholds 30 and 50 packets in the queue. In addition, the maximum queue size was 75 and the drop possibility was 10%. According to this configuration we repeated a similar test that we had also performed earlier (Figure 9, Figure 10), in order to compare the results. So we inserted background traffic that was a mix of TCP and UDP (5Mbps) and foreground traffic a mix of UDP (700Kbps) and RTP (OpenH323-based application). Finally, the foreground traffic still had only a few packet losses and very good quality of video. On the other hand, the background traffic had several drops that were caused by the WRED mechanism. The results were the following:

Traffic load (bps)	Actual throughput (bps)	Packet loss	Average jitter
UDP 5M (back)	4.87M	2%	4.800ms
TCP (back)	1.31M	-	-
UDP 700K (fore)	700K	0.034%	6.265ms
RTP	Average 300K	0.36%	-

**Table 5 Results testing the WRED mechanism**

According to the results, the foreground traffic does not seem to receive any impact from the operation of the WRED mechanism. The strict priority mechanism seems to work transparently. On the other hand, the background traffic has many packet losses, especially if we compare the result (2% losses of UDP) with the same experiment in the previous section without the WRED mechanism, where the result was less than 0.5%. So the WRED mechanism worked according to its specification and reduced the background traffic. The most significant observation arises when we see the TCP throughput of the background traffic (Figure 11) and compares it with the corresponding on the previous section. It is obvious that the throughput in this case is lower and that the WRED caused this reduction of TCP's rate.

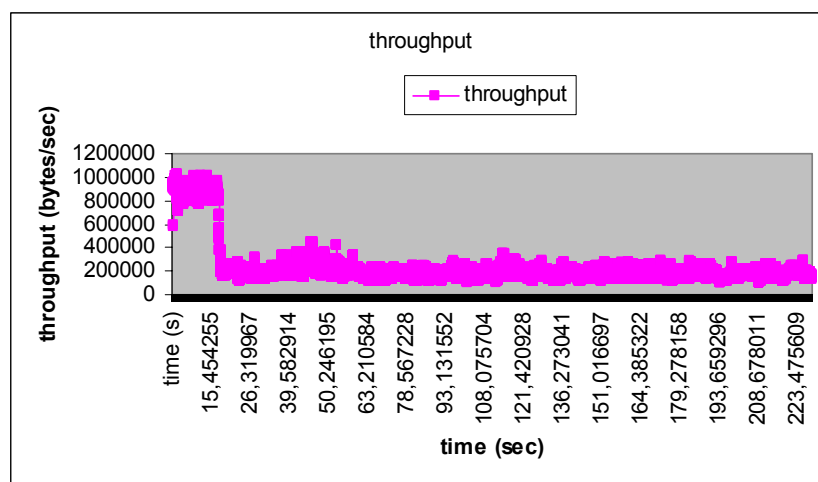


**Figure 11: TCP throughput when WRED has been applied**



## ➤ Testing scenario 2

After the first experiment, we repeated the same experiment while we had changed the thresholds of the WRED mechanism. This time we tried to approach the max queue size and configured the thresholds to be 55 and 75 packets respectively. The drop possibility was also 10%. Finally, we saw similar results regarding the foreground traffic, as the packet losses were almost zero and the video quality was very good. Now the background traffic had better behavior as only 0.92% of UDP traffic packets were lost. In addition, the TCP traffic had a bigger average throughput (1.36Mbps). So, at this experiment we allowed the queues to be more filled and achieved a better performance for the background traffic. In any case, the basic conclusion is that according to the results the CBWFQ mechanism seems to work transparently and the existence of the WRED mechanism does not have any significant impact on foreground's performance.



**Figure 12: TCP throughput with existence of WRED**

### 2.3.5 Investigation of policing profile

Finally, a very interesting and open issue for research is the way that the policy profile should be selected for aggregate of flows that contains real time traffic in order to follow pre-arranged SLAs. The policing mechanism is implemented using the token bucket algorithm. The role of this mechanism is to make sure that QoS guarantees are provided to traffic flows that obey the pre agreed rules (the mean rate that they send packets, their maximum bursts etc). Generally, the proper policing configuration is an open issue that always needs investigation. The policing mechanism gives many alternative solutions on how the network devices will treat the packets that obey and disobey the policing rules. In this case, it is the network administrator's responsibility to choose the action that the policing mechanism will apply, according to the policy that the domain wants to use. During our tests, we tried to use the policing mechanism but we faced some problems that hindered our experimentation with the policing mechanism. Therefore, our next goal is to investigate its operation on IPv6 and test it on wide- scale experiments.

## 2.4 Next Steps

At this point, we have finished our initial tests on our local tested and we presented our results in this document. Our next goals are to investigate the operation of the policing mechanism. In

addition, we would like to extend our experiments on the “Greek” part of the 6NET’s network or to participate in trials across the whole 6NET’s network.

At a later stage, we are interested in also testing the bandwidth mechanism of the MQC that only tries to allocate bandwidth to specific traffic classes. Otherwise, we are also interested in testing additional QoS features that maybe will be introduced on later versions of CISCO IOS for IPv6 QoS, as currently only a few mechanisms are supported on the IOS that we use.

### 3 GET/ENST-Bretagne: Intra-class fairness in DiffServ/AF over IPv6

#### 3.1 Introduction

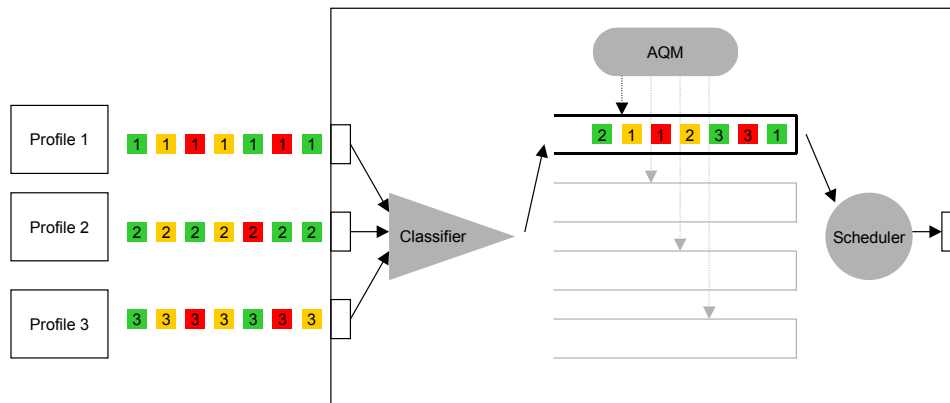
##### 3.1.1 Overview of Assured Forwarding

Assured Forwarding (AF) is one of the standardized Per Hop Behaviors of the DiffServ architecture. AF defines the differentiated forwarding of traffic aggregates in four classes with two or three packet priorities (commonly called colors) per class. The goal of AF is to assure each aggregate a target throughput. This is done by means of marking in edge routers and a combination of scheduling and differentiated dropping in core routers. A profile defines the committed rate for each aggregate. Before entering the AF backbone, the traffic of a particular aggregate is metered by the edge router. In two-color AF, packets that conform to the profile are marked green (or in-profile), and packets in excess of the committed rate are marked red (out-of-profile). In three-color AF, conforming packets are marked green and packets in excess are marked with two colors, yellow and red, in order to provide an additional level of control. If there is congestion in the AF backbone, red packets will be discarded with more likelihood than yellow packets, and these, will be more likely discarded than green packets. This is done by an Active Queue Management (AQM) mechanism for multiple priorities (generally, an extension of RED, like RIO or WRED).

##### 3.1.2 Intra-class fairness in AF

Service providers have considered offering an enhanced service based on the AF PHB. The idea is to offer customers some statistical assurance of throughput at the IP level. Since DiffServ is designed for aggregates, each one the classes of an AF core router can be shared by packets from several aggregates marked with different profiles. This means that packets marked for different target rates coexist in the same queue (Figure 13). If the service provider wants to offer an assured throughput service based on AF, he needs to know whether he can effectively control the distribution of bandwidth within a single AF class (or queue) by means of marking. Thus, we can say that this is a question of “intra-class fairness”, where the key router factors are the marker (at the edges) and the AQM mechanism (at the core). A different problem is that of “inter-class fairness”, the controlled distribution of bandwidth between the four classes or queues, which depends mostly on the scheduler. The inter-class fairness issue has not been studied in our tests for two reasons. First, we believe that the salient point of AF is the presence of colored packets in the same class. Finally, the effect of scheduling in inter-class fairness has been thoroughly discussed elsewhere and is better understood than intra-class fairness.

AF Backbone Router

**Figure 13: Active Queue Management (AQM) mechanism.**

### 3.1.3 Objective

Our subject of study is “intra-class fairness”, as explained in the previous section. We want to know what fairness in bandwidth allocation does AF offer to marked flows of the same class. In order to better understand the idea of fairness, we’ll take the definition presented in [12]:

*“In an under-subscribed network, all flows should get a share of the excess bandwidth proportional to their target rates (an equal share for equal target rates). In an over-subscribed network, all flows should experience throughput degradation proportional to their target rates (equal degradation for equal target rates)”.*

Intra-class fairness can be affected by many factors: the number of aggregates, target rates, subscription rate, the presence of unresponsive flows, packet size and heterogeneity of RTTs. Also, intra-class fairness can be evaluated at both the aggregate and micro-flow levels. For example, what happens if flows are individually marked, then aggregated and remarked? Is fairness assured for aggregates but not for micro-flows or vice-versa?

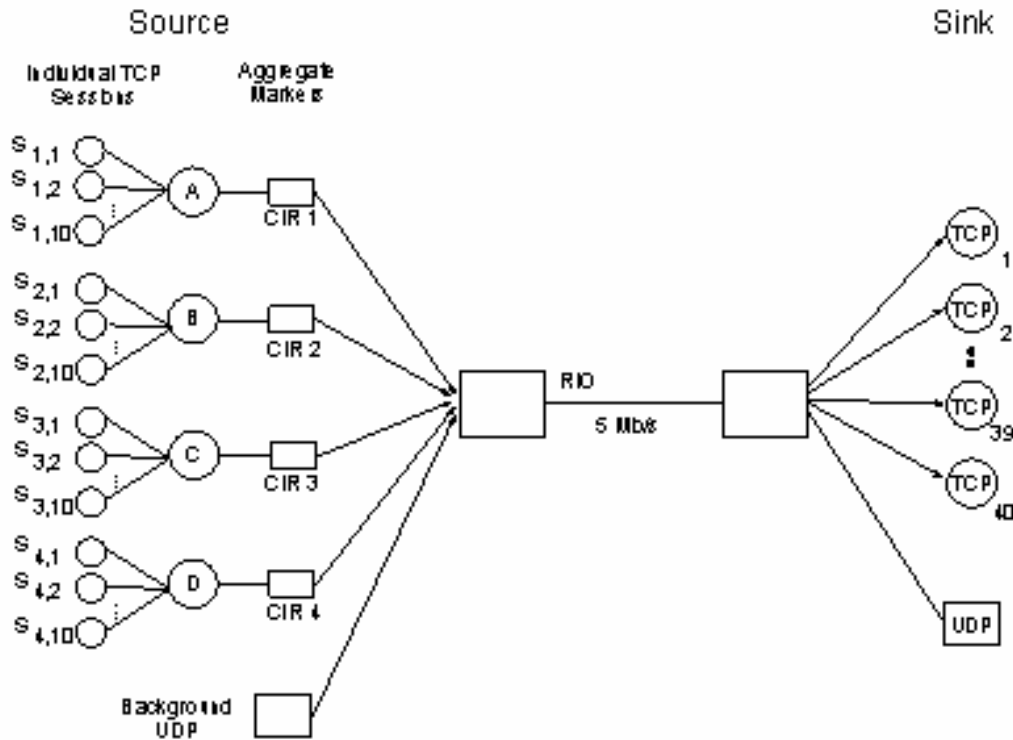
The variety of factors and approaches mentioned above dictate that several tests and scenarios are required to evaluate intra-class fairness in AF over IPv6 networks. Having this as our long-term goal, we have conducted a first phase of tests focusing on TCP aggregates and protection from unresponsive UDP flows. The specification, test-bed and results are presented in the following sections of the document. Further tests phases are in progress and will be reported in future versions.

## 3.2 Intra-class fairness for TCP aggregates

### 3.2.1 Overview

In this phase, our goal is to evaluate fairness between TCP aggregates marked with different profiles under different subscription and congestion scenarios. Figure 14 shows the logical topology used for the tests. Four TCP aggregates (10 flows each) and a UDP stream are fed into an AF domain. This AF domain is composed of a single backbone router. At the ingress interface, the router marks aggregates by means of multiple instances of the trTCM (two-rate Three Color Marker). At the egress interface, the RIO AQM mechanism performs the differentiated discard of packets. The aggregates are marked with different target rates (CIR  $i$ ,  $i = 1,..4$ ). The sum of these

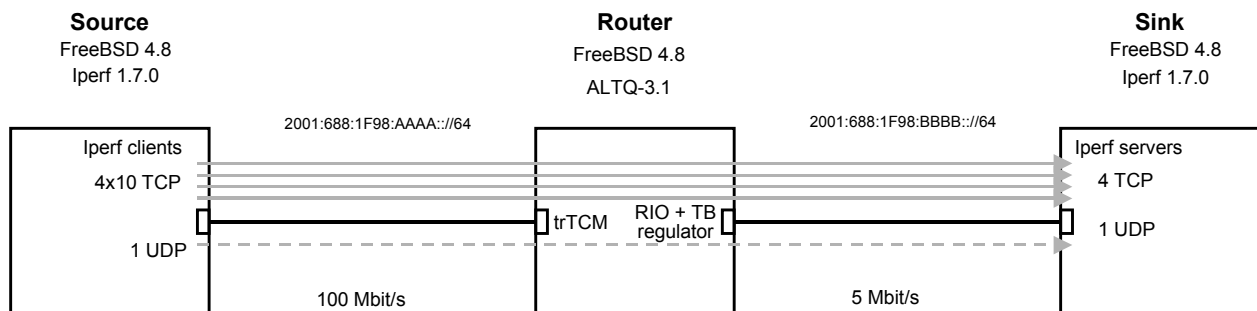
CIRs accounts for the total CIR. Several total CIRs will be tested, keeping the relative weights of individual CIRs. The throughput of each aggregate will be measured at destination for these scenarios and varying rates of the background UDP flow.



**Figure 14: Logical topology**

### 3.2.2 Test platform

Figure 15 shows the test platform used in these tests. The local IPv6 network is made up of three FreeBSD computers: two end-nodes and an AF router in the middle. The IPv6 stack is the native Kame implementation bundled in FreeBSD. The ALT-Q framework (see [10]) provides the required queuing functions for the AF router. Test flows are generated by the Iperf application (see [7]).



**Figure 15**

ALT-Q (ALternate Queuing) is a software framework for BSD that provides queuing disciplines (schedulers, AQM mechanisms) and other components required for QoS networking. ALTQ supports IPv6 and is fully integrated with the Kame stack. In our case, ALT-Q version 3.1 is used to support these functions: at ingress, a conditioner classifies and marks packets. Packets from the four aggregates are marked by four instances of the trTCM (two-rate Three Color Marker). At the egress interface, RIO3 enforces differentiated discard of marked packets and a token bucket regulator limits output capacity to 5 Mbit/s.

Iperf is a tool for measuring TCP and UDP throughput. It works as a client/server application in which, once the connection is established, test packets are sent from the client to the server, which records the amount of bytes received and calculates the average throughput of the test. Iperf works over IPv6 and allows the tuning of several parameters of the test flows, like packet size, test time and server's port number. Another important feature of Iperf is that each server process can handle multiple clients.

As seen in Figure 15, for these tests five simultaneous Iperf server processes run on the station labeled "Sink", each one in a different port. Four server processes are TCP and one is UDP. In the station labeled "Source", several client processes are launched to form the test aggregates. Ten flows are sent to each one of the four TCP server processes (identified by destination port number). A single background UDP flow is sent to the UDP server process.

### 3.2.3 Specification

#### 3.2.3.1 Test scenarios

In a first step (called Test 0) we want to see how bandwidth is distributed between TCP aggregates in plain Best Effort (no DiffServ AF) with background traffic UDP at growing rates. The UDP rates are set at 0, 30, 60 and 90% of the output link's capacity (0, 1.5, 3 and 4.5 Mbit/s, respectively).

Afterwards, we want to see how bandwidth is distributed between TCP aggregates when AF is active for different subscription rates, and also for different rates of background UDP traffic. The total subscription rate (CIR) in every case will be distributed unevenly between the aggregates in the following way: 40, 30, 20 and 10%. This distribution will be done by means of setting the appropriate marker parameters. The idea is to see if AF can effectively yield this weighted throughput. Four total subscription rates will be tested: 30, 60, 90 and 120% of output link's capacity. These four cases are designated Tests 1 to 4. Also, for every CIR, four sub-tests will be done with different rates for the background UDP flow (at 0, 30, 60 and 90% of the output link's capacity). Table 6 summarizes these scenarios.

Test	Total CIR		CIR per aggregate				Background UDP
			A 40%	B 30%	C 20%	D 10%	
	(% of link's BW)	Mbit/s	Mbit/s	Mbit/s	Mbit/s	Mbit/s	Percentages of link's BW
0	0 (Best Effort)	-	-	-	-	-	0, 30, 60, 90
1	30%	1,50	0,60	0,45	0,30	0,15	0, 30, 60, 90
2	60%	3,00	1,20	0,90	0,60	0,30	0, 30, 60, 90
3	90%	4,50	1,80	1,35	0,90	0,45	0, 30, 60, 90
4	120%	6,00	2,40	1,80	1,20	0,60	0, 30, 60, 90

Table 6

### 3.2.3.2 Configuration of the trTCM

The marker for each aggregate is a two-rate Three Color Marker [13]. This marker is composed of two cascaded token buckets:  $T_C$  (committed) and  $T_E$  (excess). Each token bucket needs two parameters, a fill rate ( $r$ ) and a bucket size ( $b$ ). For the committed bucket, we refer to them as  $r_C$  and  $b_C$ . In the same way, the parameters for the excess bucket are  $r_E$  and  $b_E$ . The configuration of each trTCM in these tests was done following a simple proportional rule presented in [15]. There, it is stated that for a particular TCP flow the burst size is proportional to the rate. Therefore, if  $r_C$  is known,  $b_C$  can be obtained by multiplying  $r_C$  by a scalar. On the other hand, the trTCM algorithm states that every time a green packet is produced, tokens are taken from both buckets. This implies that  $r_E$  should be at least equal to  $r_C$  (or larger, if we want to have packets marked yellow). Another scalar could then be used to set  $r_E$  as a function of  $r_C$ . This considerations lead to a simple parameter setting rule based on two scalar factors,  $\alpha$  and  $\beta$ , so that:

$$b_C = \alpha r_C$$

$$r_E = \beta r_C$$

Tests presented in [16] aimed at finding the best values of  $\alpha$  and  $\beta$  for intra-class fairness. The reported best values in the document are  $\alpha = 0.1$  and  $\beta = 2.5$ , and were chosen for our tests. Table 7 shows the values of the trTCM parameters for each aggregate in the four tests, calculated using the rule and values described above.

	Aggregate	CIR of aggregate		Committed		Excess	
		of total	bit/s	$r_C$ (bit/s)	$b_C$ (bit)	$r_E$ (bit/s)	$b_E$ (bit)
<b>Test 1</b> Total CIR 0,30 (of link's capacity) 1 500 000 bit/s	A	0,4	600 000	600 000	60 000	1 500 000	150 000
	B	0,3	450 000	450 000	45 000	1 125 000	112 500
	C	0,2	300 000	300 000	30 000	750 000	75 000
	D	0,1	150 000	150 000	15 000	375 000	37 500
<b>Test 2</b> Total CIR 0,60 (of link's capacity) 3 000 000 bit/s	A	0,4	1 200 000	1 200 000	120 000	3 000 000	300 000
	B	0,3	900 000	900 000	90 000	2 250 000	225 000
	C	0,2	600 000	600 000	60 000	1 500 000	150 000
	D	0,1	300 000	300 000	30 000	750 000	75 000
<b>Test 3</b> Total CIR 0,90 (of link's capacity) 4 500 000 bit/s	A	0,4	1 800 000	1 800 000	180 000	4 500 000	450 000
	B	0,3	1 350 000	1 350 000	135 000	3 375 000	337 500
	C	0,2	900 000	900 000	90 000	2 250 000	225 000
	D	0,1	450 000	450 000	45 000	1 125 000	112 500
<b>Test 4</b> Total CIR 1,20 (of link's capacity) 6 000 000 bit/s	A	0,4	2 400 000	2 400 000	240 000	6 000 000	600 000
	B	0,3	1 800 000	1 800 000	180 000	4 500 000	450 000
	C	0,2	1 200 000	1 200 000	120 000	3 000 000	300 000
	D	0,1	600 000	600 000	60 000	1 500 000	150 000

Table 7

### 3.2.3.3 Configuration of RIO

RIO (RED with In and Out) has been generalized to handle  $n$  packet priorities. In our tests, RIO is used with 3 priorities. The ALTQ implementation of RIO requires setting 10 parameters: two thresholds and a maximum discard probability per priority (9 parameters), as well as a weight factor (for the calculation of the weighted moving average). In order to set these parameters, we used the rules presented in [17], which in turn are derived from those originally proposed for Adaptive RED [11]. The rules from [17] define overlapped thresholds for all priorities. This reduces the number of parameters to six. Then, following the rules from [11], the value of the thresholds and the weight factor are a function of a user defined target delay and the link's bandwidth. This leaves us with only the maximum discard probability of each priority. We set the probabilities inversely

proportional to the priorities (i.e. a lower discard probability for a higher priority). The precise rules and the obtained values are described below.

- For a given target delay  $d$  (in seconds), and a link bandwidth  $C$  (in packets/s), the thresholds are given by:

$$th_{min} = d * C / 2$$

$$th_{max} = 3 * th_{min}$$

- The weight factor  $w_q$  is given by:

$$w_q = 1 - \exp(-1/C)$$

- In our case:

$$C = 625 \text{ packets/s (5 Mbit/s and a packet size of 1000 bytes)}$$

$$d = 0.05 \text{ s (we define a target of 50 milliseconds).}$$

- Thus,

$$th_{min} = 15 \text{ packets}$$

$$th_{max} = 46 \text{ packets (due to round up)}$$

$$w_q = 0.0016.$$

The ALTQ implementation of RIO requires an indirect setting of  $w_q$ . In the expression:  $w_q = 1 / 2^n$ , it's  $2^n$  (an integer) that is required by ALTQ. Following this requirement, the best approximation to our target  $w_q$  is given by  $2^n = 512$  ( $1 / 512 = 0.002$ ).

The maximum discard priorities are set at 0.02, 0.1 and 0.2 for green, yellow and red packets, respectively

#### 3.2.3.4 Test runs

We specified a total of 20 scenarios: five tests (0 to 4) and four UDP rates per test. For each scenario, we launched 10 different runs. In each run, the 41 individual flows (4x10 TCP and one UDP) start at random times in a period of 10 seconds and they last for two minutes. The results of the tests are shown and commented in the following section.

### 3.3 Results

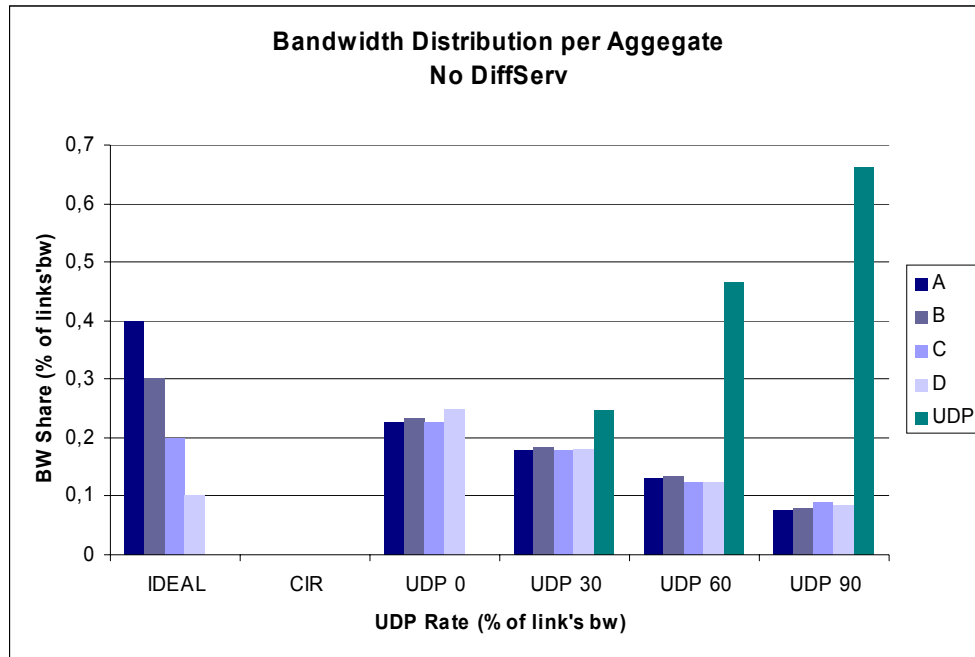
#### 3.3.1 Bandwidth distribution

The plots in Figure 16 to Figure 20 show the observed bandwidth distributions for the five tests. In each plot there are six sets of bars. The first set of bars (left to right) shows the ideal bandwidth distribution that we seek. The second set of bars represents the committed rate for each aggregate.



The following sets show throughput per aggregate for the different rates of the background UDP flow. The bandwidth of each aggregate (each bar) is expressed as a percentage of link's total bandwidth.

Figure 16 is test 0, the Best Effort case. As expected (RTTs and packet sizes are the same), it can be seen that bandwidth is evenly distributed between the four aggregates. We also see that the bandwidth available for TCP aggregates gets scarcer as the UDP rate increases. Obviously, since neither marking nor differentiated discard are active, the ideal weighted bandwidth distribution is not observed in any case in this test. The bars of the CIR set are null for the same reason.



**Figure 16**

In each plot in Figure 17 to Figure 20, we see that bandwidth is effectively distributed in a weighted fashion thanks to AF mechanisms. Nevertheless, the ideal distribution is never fully attained. In most cases, the sum of the four aggregate's bandwidths is not 100%. On the other hand, TCP aggregates are effectively protected from unresponsive UDP flows.

We can state other general (and informal) comments based on the data plotted in these figures. First, we observe that all aggregates get at least their committed rate in low-subscription scenarios (total CIRs of 30% and 60%). However, the share of excess bandwidth obtained by an aggregate is not proportional to its committed rate. That is one of the reasons for the difference between the observed and ideal throughputs. In these cases, we see that aggregates with higher committed rates are more penalized than aggregates with lower committed rates. For example, for total CIR= 30% (Figure 17) and UDP = 0, the observed throughput of aggregate A (33%) is 17.5% lower than the ideal (40%). At the same time, aggregate D gets 14% of bandwidth, 40% more than the ideal of 10%.

In high subscription scenarios, things seem to go worse for high-rate aggregates. First, as seen in Figure 19 and Figure 20, aggregates A and B do not get their committed rates. In addition, when the network is over-subscribed (CIR = 129% and UDP = 0, Figure 18), aggregate A gets a smaller share of bandwidth than in previous tests. In general, we can say that the tandem of RIO and trTCM favors aggregates with lower committed rates: they get a bigger share of excess resources in under-subscribed scenarios and are less penalized in over-subscription.



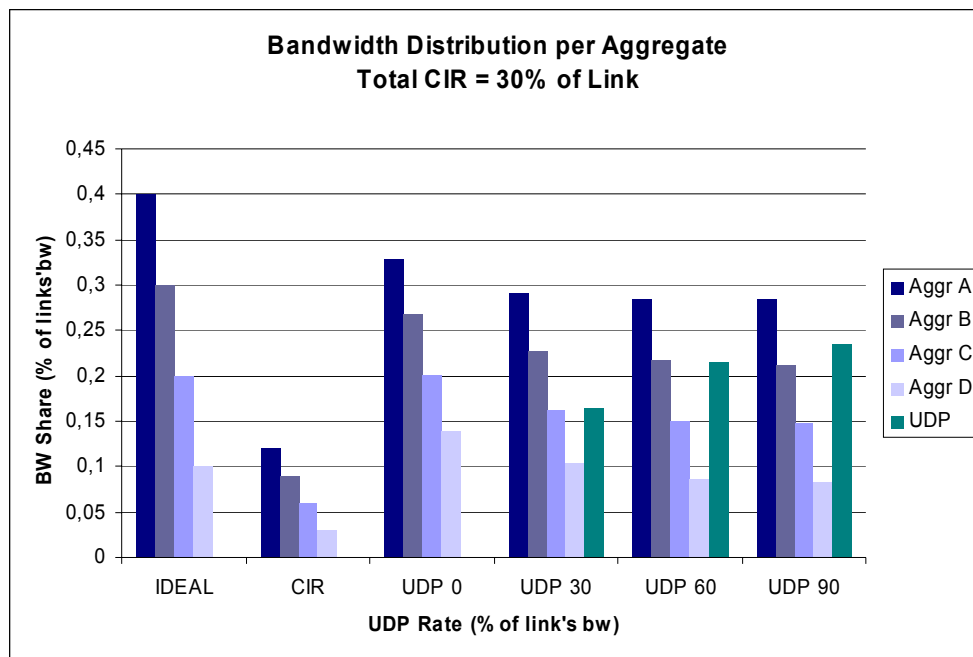


Figure 17

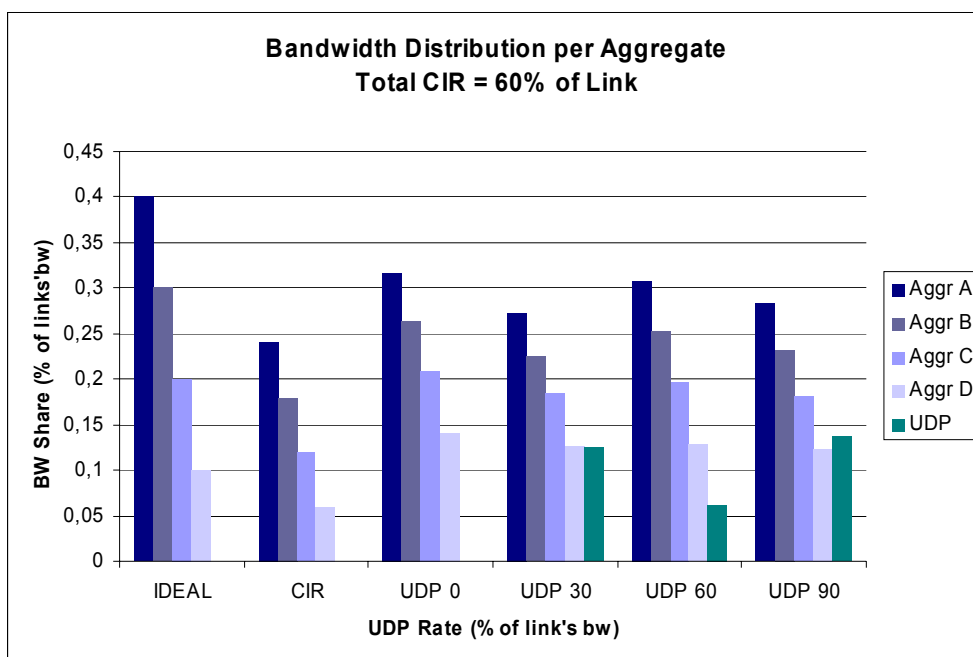


Figure 18

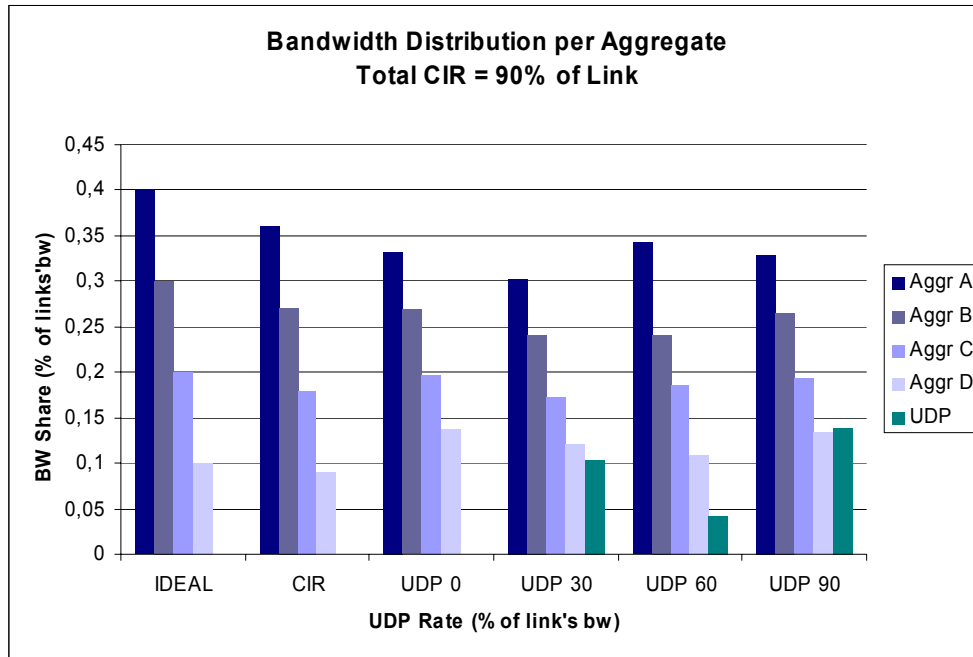


Figure 19

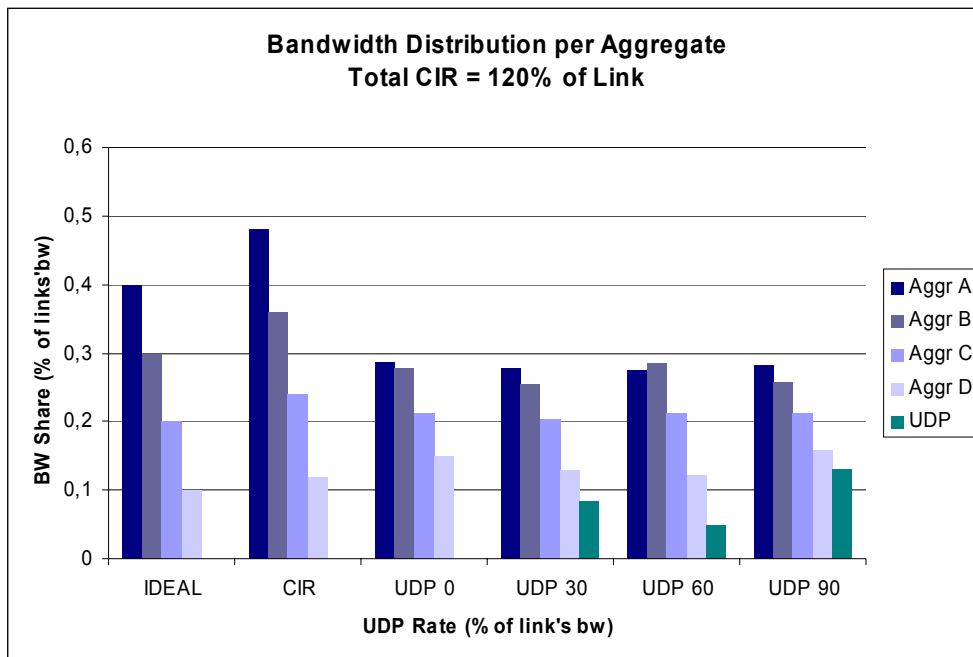


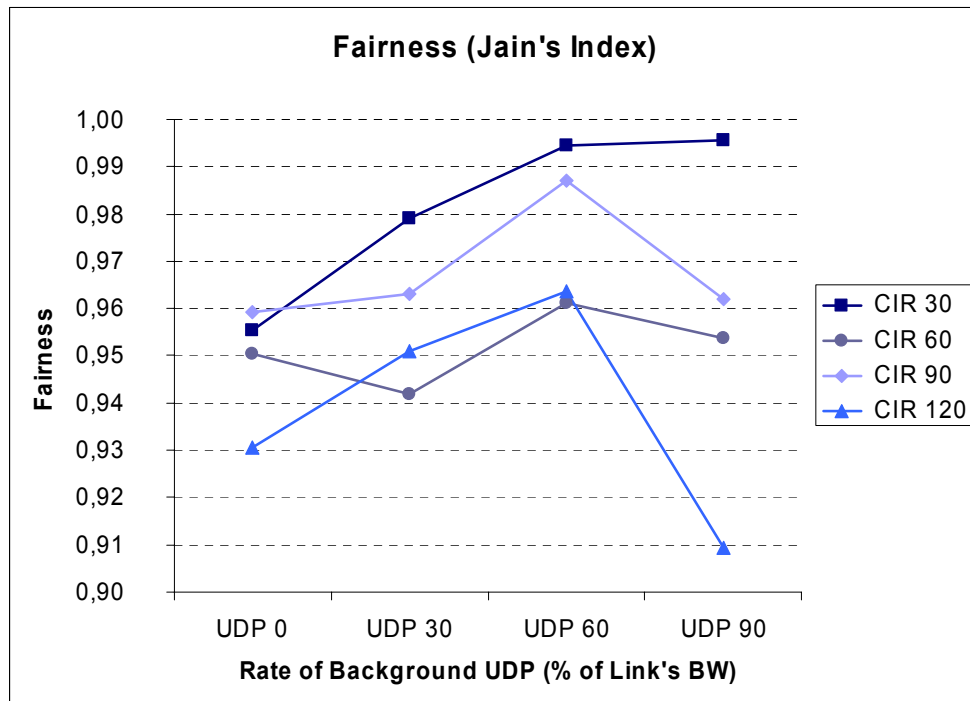
Figure 20

### 3.3.2 Fairness index

The plots shown in the previous section give a visual and intuitive idea of bandwidth distribution. Nevertheless, we would like to have a more precise idea of which scenario yields a better bandwidth distribution. In order to verify this, we use Jain's fairness index [14]. This performance metric defines fairness as a function of the variability of throughput across users. For a given set of measured throughputs ( $T_1, T_2, \dots, T_n$ ), if the fair (ideal) throughputs are known ( $O_1, O_2, \dots, O_n$ ), Jain defines the normalized throughput as  $x_i = T_i / O_i$  and fairness as:

$$\text{Fairness Index} = \frac{(\sum x_i)^n}{n \sum x_i^2}$$

This index gives a value between 0 and 1, where 1 is the maximum fairness (all measured throughputs are equal to the ideal throughputs). We have calculated Jain's fairness index for each scenario and the results are plotted in Figure 21. Each line presents the fairness values of one test (a total CIR) for the different rates of the background UDP flow. We see that the highest fairness values occur consistently for a total CIR of 30%, followed by a CIR of 90%. We also see a tendency of fairness increasing as UDP rate increases for a given CIR. This tendency is observed for UDP rates of 30 and 60% of link's bandwidth. After that, fairness stalls (CIR = 30%) or begins to decrease with varying slopes according to the CIR. Fairness increases when UDP is present at certain rates because differentiation is better. This is due to the fact that RIO, the mechanism that enforces discrimination, is active only when there is a certain amount of congestion.



**Figure 21**

Fairness values have been sorted in Table 8. The left column contains the scenario in the form (x,y) where x designates the total CIR and y, the UDP rate. The corresponding fairness values are in the right column. The highest fairness occurs in the following scenarios: (30,90) and (30,60). These cases are followed by (90,60). The first two cases occur in a low-subscription scenario. The third one occurs in a scenario when subscription is close to the link's capacity. All three cases occur when UDP rate is high. We could say that a trade-off must be found between subscription and congestion to maximize discriminations and, thus, fairness. A service provider might directly find the subscription rate of 90% more attractive. This CIR gives the best fairness when there is no UDP, its other fairness values are always the second best and it would signify the best profit for a given bandwidth resource. The main disadvantage is that some aggregates fail to get their committed rate.

Scenario (CIR, UDP rate)	Fairness Index
(30,90)	0,9955
(30,60)	0,9944
(90,60)	0,9871
(30,30)	0,9789
(120,60)	0,9636
(90,30)	0,9632
(90,90)	0,9620
(60,60)	0,9611
(90,0)	0,9591
(30,0)	0,9554
(60,90)	0,9536
(120,30)	0,9508
(60,0)	0,9505
(60,30)	0,9419
(120,0)	0,9306
(120,90)	0,9093

**Table 8**

### 3.4 Conclusions and future work

The issue of intra-class fairness of Assured Forwarding over IPv6 was evaluated experimentally in a local test platform. We found evidence that Assured Forwarding does not offer complete throughput fairness for aggregates forwarded in the same class. On the other hand, AF can effectively be used to protect TCP flows from unresponsive UDP flows forwarded in the same class.

The conclusions expressed above are far from definitive, because only a small set of conditions were tested. First, our observations were done using a particular implementation of AF by means of two mechanisms: the RIO AQM algorithm and the two-rate Three Color Marker. Also, we used specific settings for RIO parameters. One possible extension of our tests is then the evaluation and comparison of different marking and AQM mechanisms, as well as the impact of RIO parameters in intra-class fairness.

It is widely known that RTT is a factor that greatly affects fairness in Best Effort networks. The study of the effect of RTT in AF's intra-class fairness is then another possible extension of the work presented here. The RTT issue is very related to another aspect: we have focused in intra-class fairness for aggregates, but did not look at what happened at the level of individual flows. Simulation studies are already available, but experimental results would definitely add value in this subject.

## 4 University of Crete (UoC): A Testbed Investigation of QoS Mechanisms for Supporting SLAs in IPv6

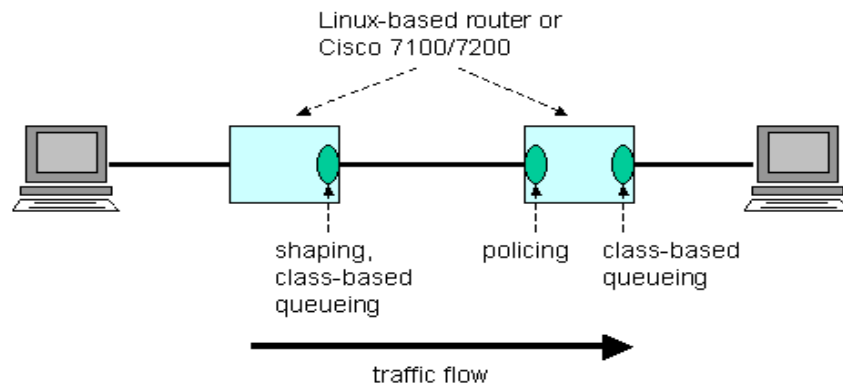
### 4.1 Introduction

In this report, we present and discuss results of investigating the combined operation and interaction of Quality of Service mechanisms, namely traffic policing, traffic shaping and class-based queuing.

All experiments were performed in an IPv6 testbed for TCP traffic. These mechanisms will be applied on edge routers at the interface between a user organization and his network provider, hence their tuning is important for effectively supporting Service Level Agreements (SLAs). Our results show how various parameters of the mechanisms affect the overall performance, hence can provide guidelines for appropriately selecting their values.

## 4.2 Testbed analysis

Our experiments were performed using the testbed topology shown in Figure 22. The two routers were either Cisco routers (7100/7200 series) or Linux-based routers. The Cisco routers ran IOS 12.3(a) Release with IPv6 Quality of Service support, while the Linux routers ran Red Hat Linux 9.0 with the IPv6 module for IPv6 support loaded. The edge PCs also ran Red Hat Linux 9.0, acting one as client (sending traffic) and one as server (receiving traffic). The devices were connected with a Fast Ethernet links at 100 Mbps.



**Figure 22: IPv6 testbed topology**

The traffic is generated from left to right (as shown in Figure 22), first being shaped/class-based queued at the first-hop router and then policed/class-based queued at the second-hop router. Shaping and class-based queuing was applied at the egress interfaces, while policing was applied at the ingress interfaces. Also, the IPv6 Cisco Express Forwarding (CEF) mechanism was enabled at the Cisco router.

## 4.3 Traffic Generators

In this chapter we analyze the traffic generators that were used in our experiments. Namely, we used Iperf 1.7.0 for Linux.

### 4.3.1 Iperf 1.7.0 for Linux

This version of Iperf for Linux systems gives the opportunity of generating TCP and UDP traffic. The most significant features are:

- TCP
  - Throughput measurement

- 
- Time / data-length based traffic generation
  - Support of variable segment and congestion window size
  
  - UDP
    - Throughput measurement
    - Delay jitter measurement
    - Packet loss ration measurement
    - Time / data-length based traffic generation
    - Support of specific rate traffic generation

Iperf supports IPv6 address assignment (using the `-V` option)

#### 4.3.2 A Simple Example

**Server side:**

```
# iperf -s -V
```

**Client side:**

```
# iperf -c fec2::11 -V
```

The client generates TCP traffic for the default amount of time (10 secs) and the server reports the achieved throughput.

Note: fec2::11 is the remote (server) IPv6 address

### 4.4 Quality of Service mechanism analysis

In this chapter, we present the QoS mechanisms used throughout our experiments. The Cisco Traffic Policing and Generic Traffic Shaping are supported in IOS 12.3 Release for IPv6. Also, the Linux Traffic Policing and Shaping are supported in Red Hat Linux 9.0 (kernel version 2.4).

#### 4.4.1 Token Bucket

A token bucket is a mechanism used to define a certain rate for a transfer. It has three components:

- The *mean rate*, which specifies how much data can be forwarded per unit time on average. The mean rate is measured in bits/second.
- The *burst size* or Committed Burst (Bc), which specifies the amount of data that can be sent within a time unit. The burst size is measured in bits.
- The *time interval* (sec) or *measurement interval*, which specifies the time quantum. It is measured in seconds.

Between the above three the following relationship is valid:

$$Bc = \text{mean\_rate} * \text{time\_interval}$$

In metaphor, tokens are put into the bucket at a certain rate. The bucket has a maximum capacity, and when it becomes full newly arriving tokens are discarded. To transmit a packet the mechanism removes from the bucket a number of tokens equal to the packet size. If the remaining tokens are not enough, then the arriving packet has either to wait for new tokens by being placed at the queue/buffer (shaping) or is discarded (policing).

#### 4.4.2 Cisco Traffic Policing

Traffic policing is used to drop traffic that does not conform to the specified token bucket parameters. The Traffic Policing mechanism manages the maximum rate of traffic that is sent or received on an interface through a token bucket algorithm. Below we show the commands used in Cisco routers in order to attach a Traffic Policing mechanism:

```
Router(config)# ipv6 access-list access-list-id
Router(config-acl)# permit transport-protocol source-addr destination-addr range
port-num port-num
Router(config-acl)# exit
Router(config)# class-map class-map-id
Router(config-cmap)# match access-group name access-list-id
Router(config-cmap)# exit
Router(config)# policy-map policy-map-id
Router(config-pmap)# class class-map-id
Router(config-pmap-c)# police bps bc burst-normal be burst-max conform-action
action exceed-action action violate-action action
Note: burst-normal and burst-max should be in bytes
```

We then apply the policy we created to the desired interface:

```
Router(config-if)# service policy input/output policy-map-id
```

Below we show a specific example of the attachment of Traffic Policing mechanism on the input interface of a Cisco router with the parameters:

Traffic Polcing Rate = 50 Mbps,

Burst Size = 2 Mbits,

Non conforming traffic should be dropped.

```
Router(config)# ipv6 access-list list_1
```

```

Router(config-acl)# permit tcp any any range 4000 9000
Router(config-acl)# exit
Router(config)# class-map class_1
Router(config-cmap)# match access-group name list_1
Router(config-cmap)# exit
Router(config)# policy-map policy_1
Router(config-pmap)# class class_1
Router(config-pmap-c)# police 50000000 bc 250000 be 250000 conform-action transmit
exceed-action drop

Router(config)# interface FastEthernet 0/0
Router(config-if)# service-policy input policy_1

```

#### 4.4.3 Cisco Generic Traffic Shaping (GTS)

Generic traffic shaping is used to delay packets that do not conform to the specified token bucket regulator. These packets are placed in a buffer, and wait until a sufficient amount of tokens become available in order to be transmitted. Below we show the commands that should be used in order to attach a Traffic Shaping mechanism:

```

Router(config-if)# traffic-shape < rate | group access-list-id > bit-rate [burst-size
[exceess-burst-size]] [buffer-limit]

```

Using the above commands, the Traffic Shaping mechanism will be applied to the specified interface (shaping is by default applied to the output of an interface) with the specified token bucket parameters.

#### 4.4.4 Linux Traffic Policing

Red Hat Linux 9.0 (with a kernel version 2.2/2.4) provide the opportunity to manage traffic using Policing and Shaping mechanisms, similar to a Cisco router. Linux implements a Traffic Policing mechanism, which can be defined using the following commands:

```

# tc qdisc add dev eth1 handle 1:0 root dsmark indices 64
# tc filter add dev eth1 parent 1:0 protocol ipv6 prio 1 u32 \
    match ip6 src fec0::11/64 police rate 50mbit burst 2.0mbit drop flowid 1:1

```

OR

```

# tc qdisc add dev eth0 handle ffff: ingress
# tc filter add dev eth0 parent ffff: protocol ipv6 prio 1 u32 divisor 1
# tc filter add dev eth0 parent ffff: protocol ipv6 prio 1 u32 \
    match ip6 src fec0::11/64 police rate 50mbit burst 2.0mbit drop flowid 1:1

```



The above series of commands police the traffic coming from host with IPv6 address fec0::11/64, with rate 50 Mbps and burst size 2 Mbits.

#### 4.4.5 Linux Traffic Shaping

To apply traffic shaping we can use the Token Bucket Filter mechanism which implements a token bucket algorithm. More specifically we used the commands below:

```
# tc qdisc add dev eth1 root tbf rate 50mbit burst 1.25mbit limit 9.25mbit
```

The above command attaches a Token Bucket Filter (TBF) mechanism to the output of the interface eth1, with parameters:

Traffic Shaping Rate = 50 Mbps,

Burst Size = 2 Mbits,

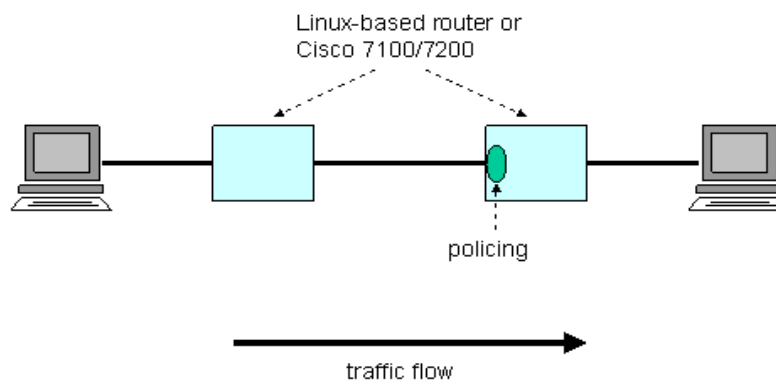
Buffer limit = 8 Mbits

### 4.5 Experimental Results

In this chapter we present the results of our experiments, using the topology described in 4.2 and the mechanisms described in 4.4.

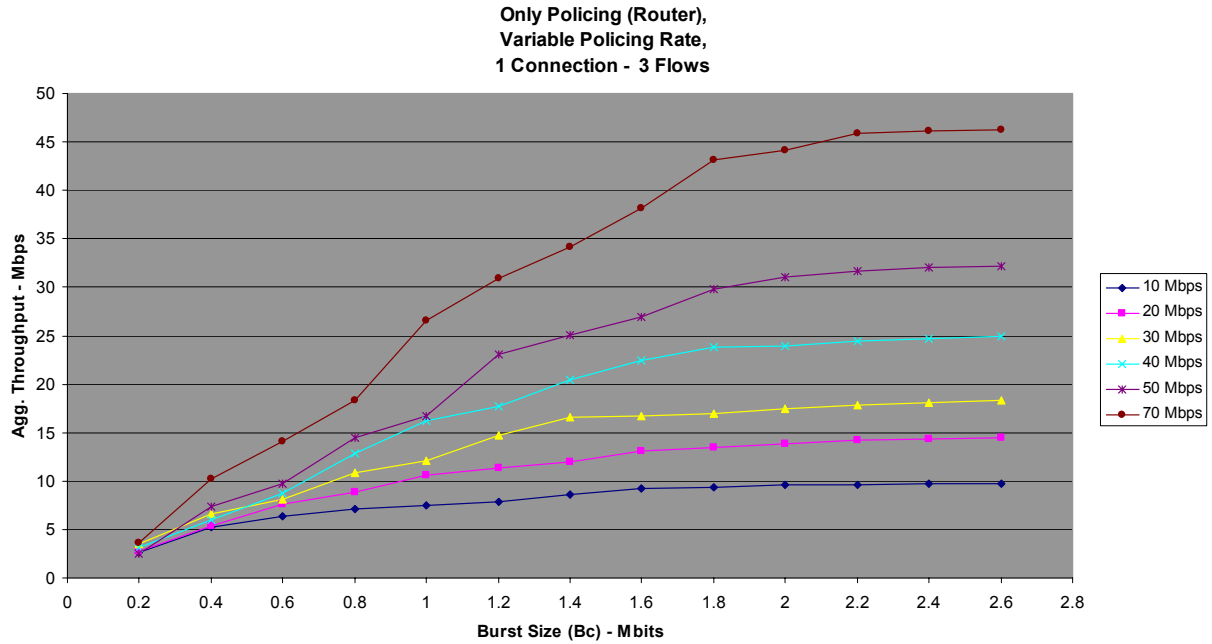
#### 4.5.1 Policing tests

We first considered the case where only the policing mechanism is applied. This is clearly shown in Figure 23.



**Figure 23: Policing testbed**

In Figure 24, we show the aggregate throughput as a function of the bucket size  $B_p$  for different policing rates  $R_p$  for a number of 5 TCP flows. As expected the aggregate throughput increases with the bucket size, but always remains less than the policing rate  $R_p$ . In addition, we notice that for higher policing rates the aggregate throughput is proportionally smaller: for  $R_p = 10$  Mbps the aggregate throughput is 9.7 Mbps, while for  $R_p = 70$  Mbps the aggregate throughput is only 46 Mbps.



**Figure 24: Aggregate throughput as a function of bucket size for different policing rates**

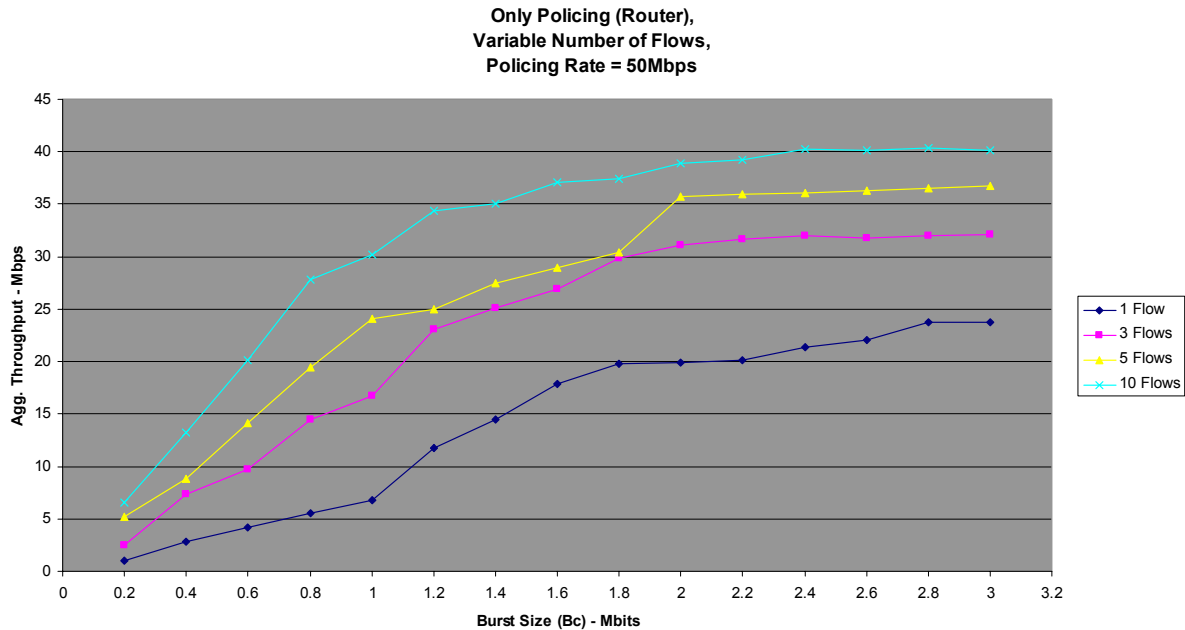
The dependence of the aggregate throughput on the bucket size exhibits a “knee”, i.e. for buckets sizes up to a specific value the aggregate throughput steadily increases, whereas for bucket sizes larger than that value the increase is very small. An appropriate selection for the bucket size is the value at which the “knee” appears. Based on our experimental results, this value  $B_p$  is approximately given by the following two relationships:

$$B_p = 0.6 \text{ Mbit} + 0.03 \text{ sec} * R_p, \text{ if } R_p < 40 \text{ Mbps}$$

**and,**

$$B_p = 1.5 \text{ Mbit} + 0.01 \text{ sec} * R_p, \text{ if } R_p > 40 \text{ Mbps}.$$

Figure 4 shows the aggregate throughput as a function of the bucket size for a different numbers of TCP flows, for a fixed policing rate (50Mbps).



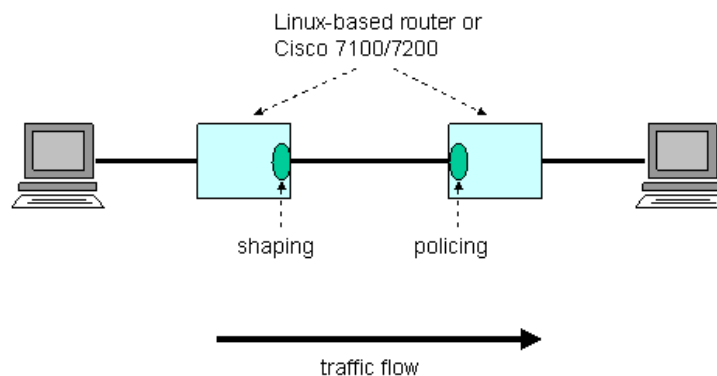
**Figure 25: Aggregate throughput as a function of bucket size for different number of flows**

As shown in Figure 25, the optimum value of the bucket size  $B_p$ , for which we are on the “knee” appears to be independent of number of TCP flows. Furthermore, the aggregate throughput increases with the number of flows.

The above experiments were conducted using traffic policing on a Linux-based router. Experiments with traffic policing on a Cisco router gave identical results, hence we can conclude that the operation and performance of traffic policing on Linux-based and Cisco routers is the same.

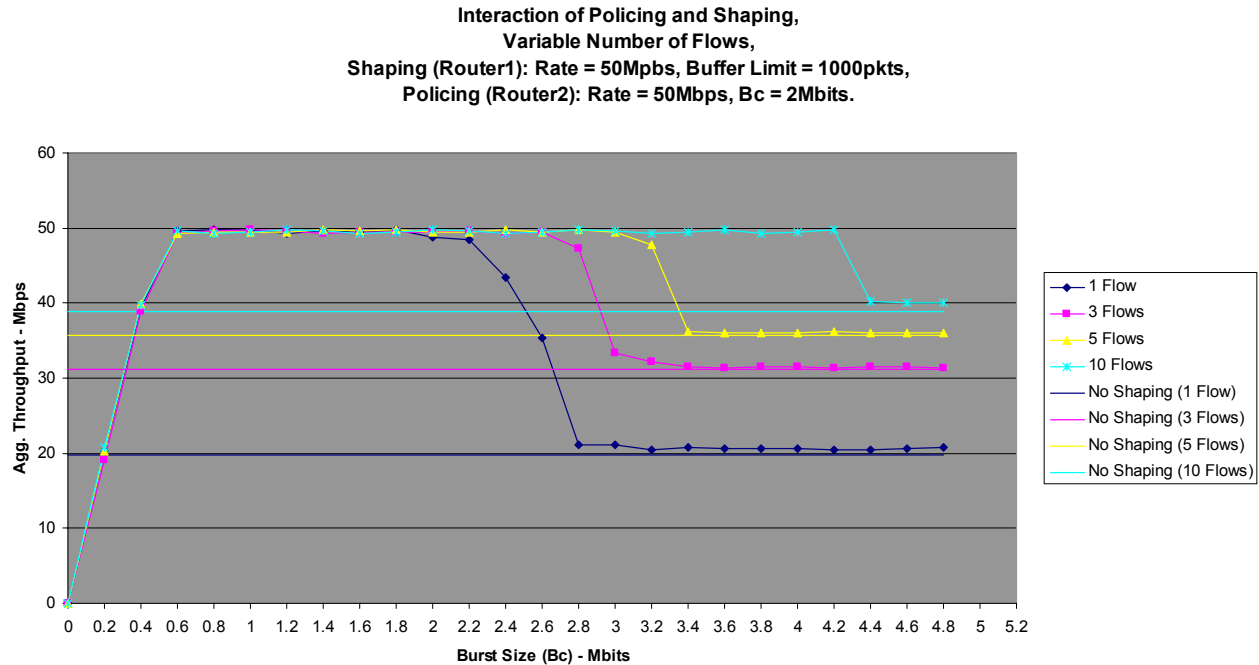
#### 4.5.2 Interaction of Policing and Shaping

In this section we consider the case where both policing and shaping are applied. As shown in Figure 26, , shaping is applied at the egress interface of the first-hop router while policing is applied at the ingress interface of the second-hop router. For this experiment both routers used where Linux-based routers (results for Cisco routers will be presented latter).



**Figure 26: Interaction of policing and shaping**

Figure 27 shows the aggregate throughput as a function of the shaper's bucket size for a different numbers of TCP flows. We chose fixed values for the policing parameters (policing rate at 50 Mbps and bucket size at 2 Mbits).



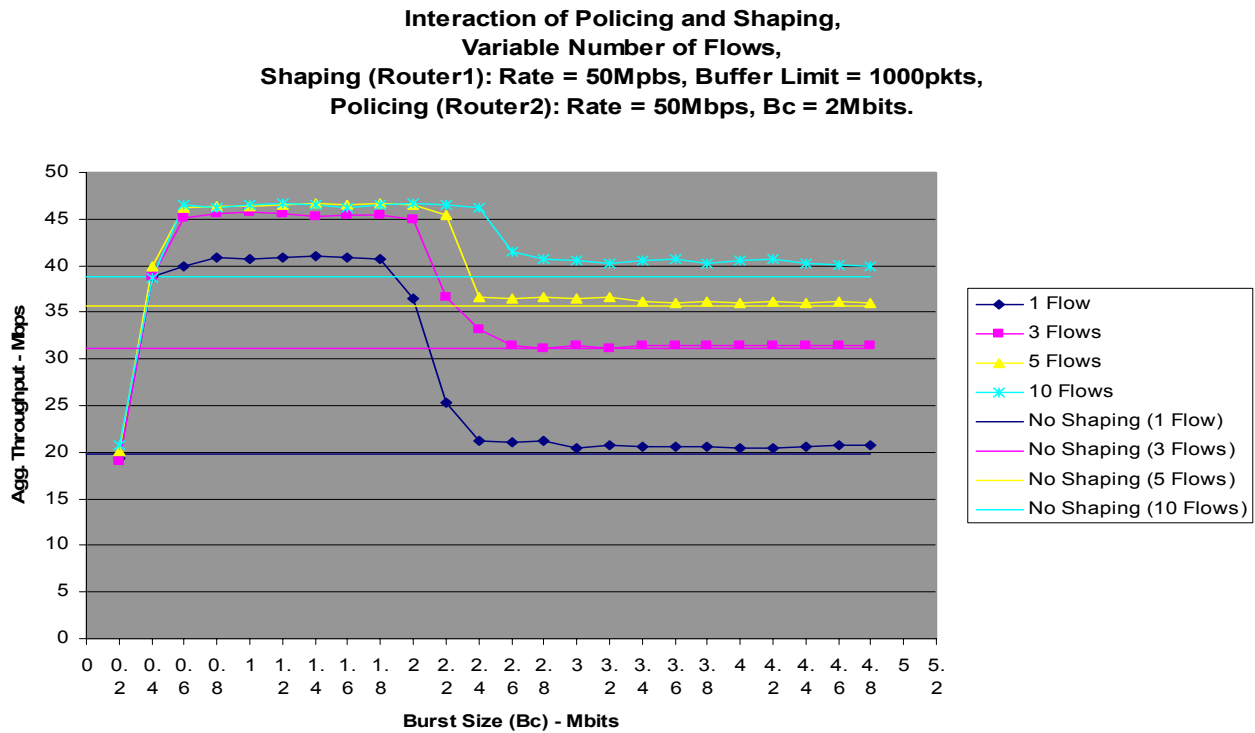
**Figure 27: Aggregate throughput as a function of shaping bucket size for different number of TCP flows (2 Linux Routers)**

Figure 27 shows that with appropriate selection of the shaper's bucket size we can achieve throughput nearly equal to the policer's rate, hence higher than the throughput achieved with policing alone; indeed, the throughput achieved is approximately equal to the policing rate. This suggests that the use of traffic shaping can be beneficial in terms of increased aggregate throughput.

Moreover, regarding the dependence of the throughput on the shaper's bucket size, we observe that for small values of the bucket size the aggregate throughput is low. We attribute this behaviour to the fact that a small bucket limits the amount of data that can be sent. On the other hand, for large values of the bucket size, the aggregate throughput is small, and approaches the throughput when shaping is not used. This is because a large bucket size allows large bursts into the network, which lead to drops because of the policing mechanism, hence diminishes the smoothing effect of shaping.

Finally, Figure 27 shows that the number of TCP flows affects the range of values of the bucket size for which the maximum throughput is achieved. Indeed, in case of 10 flows we see that the range of values of Bs for achieving high throughput is greater than the case of one flow.

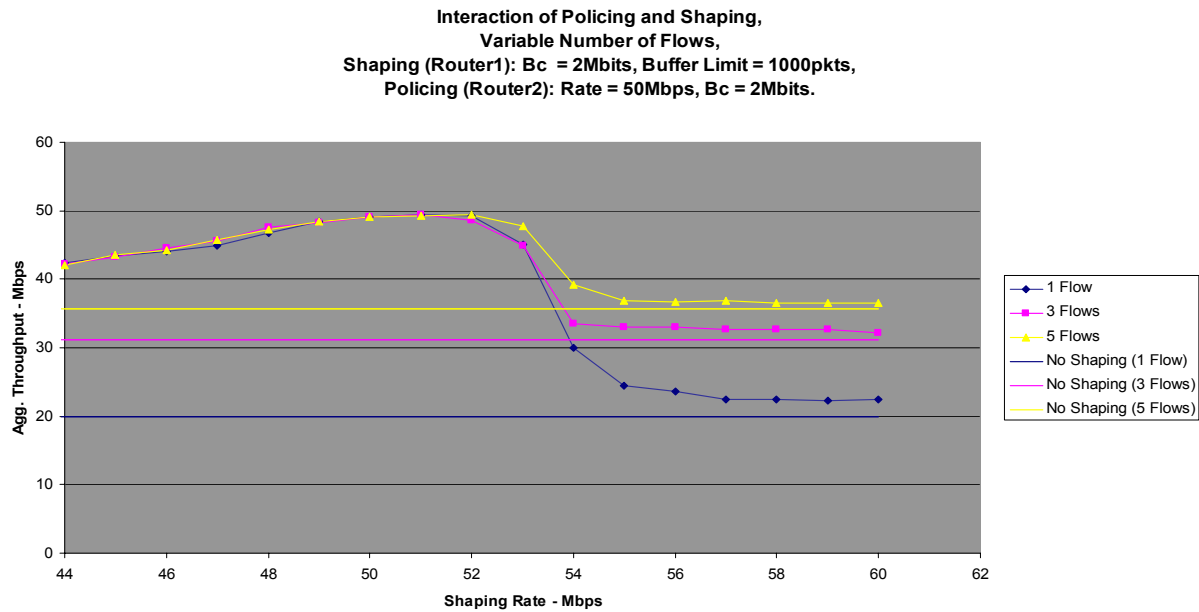
Figure 30 shows the dependence of the aggregate throughput on the bucket size when policing is performed by the Cisco router, and shaping is performed at the Linux-based router.



**Figure 28: Aggregate throughput as a function of shaping bucket size for different number of TCP flows (1 Linux, 1 Cisco Router)**

The figure above is qualitative the same with the one with Figure 27, where both routers were Linux-based. The difference lies upon the range of values for which we achieve the maximum aggregate throughput. We observe that the highest allowable value for the bucket size  $B_s$  for the shaper is now smaller.

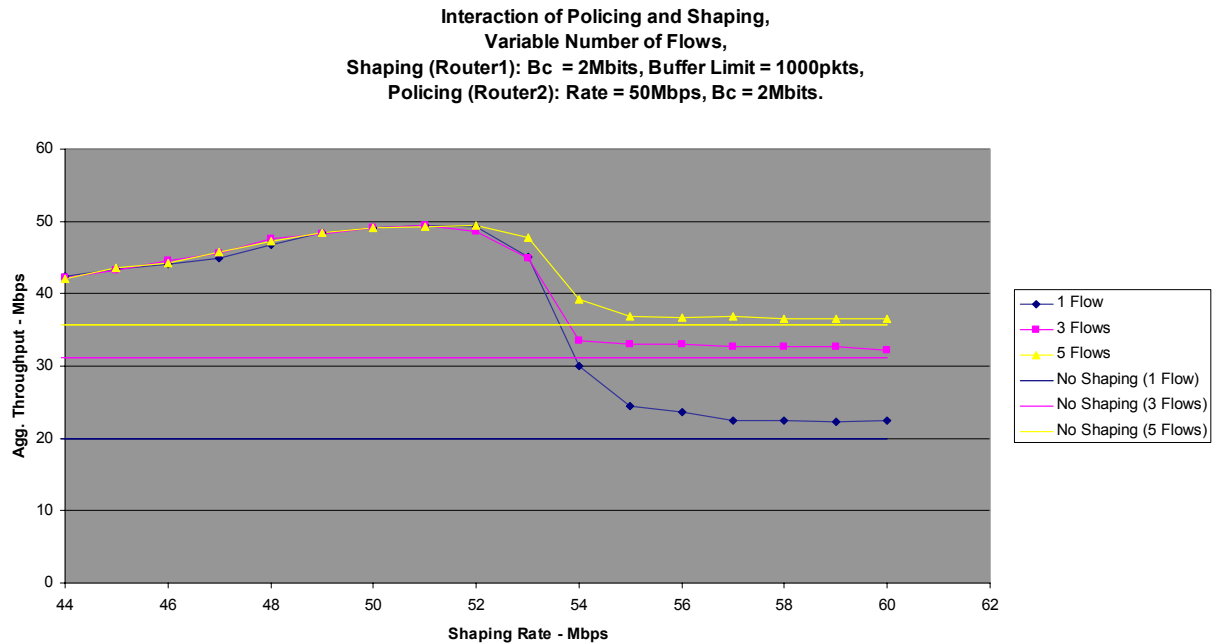
Our next test concerned the best selection of the shaper's shaping rate. We kept fixed values for the policer ( $R_p = 50$  Mbps,  $B_p = 2$  Mbits) and measured the aggregate throughput as a function of the shaper's rate for a different number of flows. The results in the case of two Linux-based routers are shown in Figure 30.



**Figure 29: Aggregate throughput as a function of shaping bucket size for different shaping rates (2 Linux Routers)**

As we can see from the above figure, the maximum throughput is achieved for a shaping rate somewhat higher than the policing rate; nevertheless, the increase of the throughput compared to the throughput when the shaping rate is equal to the policing rate is small. Also, observe that when the shaping rate further increases, then the throughput starts to decrease, and eventually approaches the throughput when no shaping is applied. Another observation in Figure 29 is that the dependence of the throughput on the shaping rate is similar for a different number of flows.

Next, we repeated the same experiment using one Linux-based router, which performed shaping, and one Cisco Router, which performed policing. The results are shown in Figure 30 below.



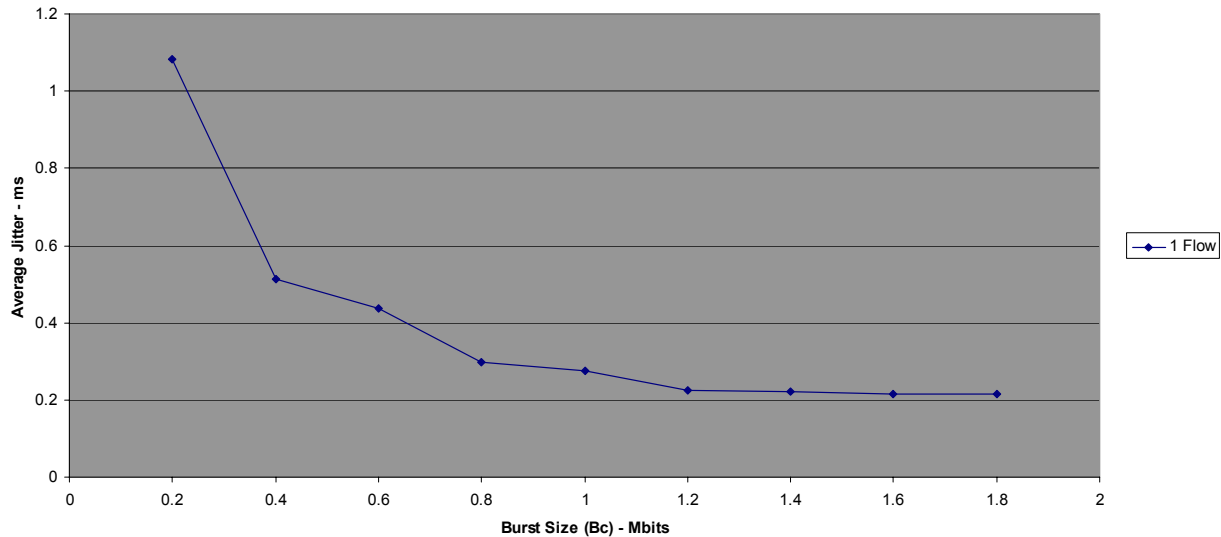
**Figure 30: Aggregate throughput as a function of shaping bucket size for different shaping rates**

Again, we see that there is no qualitative difference when we use Cisco routers. Observe however that the aggregate throughput starts to decrease for values of the shaping rate smaller than the corresponding values for the case of two Linux-based routers.

The above experiments lead us to the conclusion that the qualitative behaviour of the interaction between shaping and policing is the same for both Cisco and Linux-based routers.

As we saw in Figure 28, there is a range of values for the shaper's bucket size Bc for which the aggregate throughput is maximized. For example, in the case of one flow we can see that this range of values is 0.6 – 2 Mbits. Next we investigate the average delay jitter that packets experience for different values of the bucket size Bc. The results are shown in Figure 31. Observe that the average delay jitter decreases when the bucket size increases. Hence, we can conclude that the optimal setting of the bucket size, which both maximizes the aggregate throughput and decreases the average delay jitter, is a high value of the bucket size that still remain within the interval suggested by Figure 28.

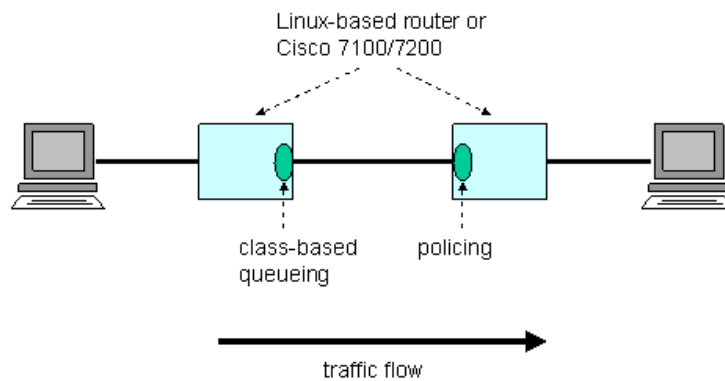
Interaction of Policing and Shaping,  
 Shaping (Router1): Rate = 50Mbps, Buffer Limit = 1000pkts,  
 Policing (Router2): Rate = 50Mbps, Bc = 2Mbits.



**Figure 31: Interaction of policing and shaping**

#### 4.5.3 Interaction of Policing and Class Based Queuing

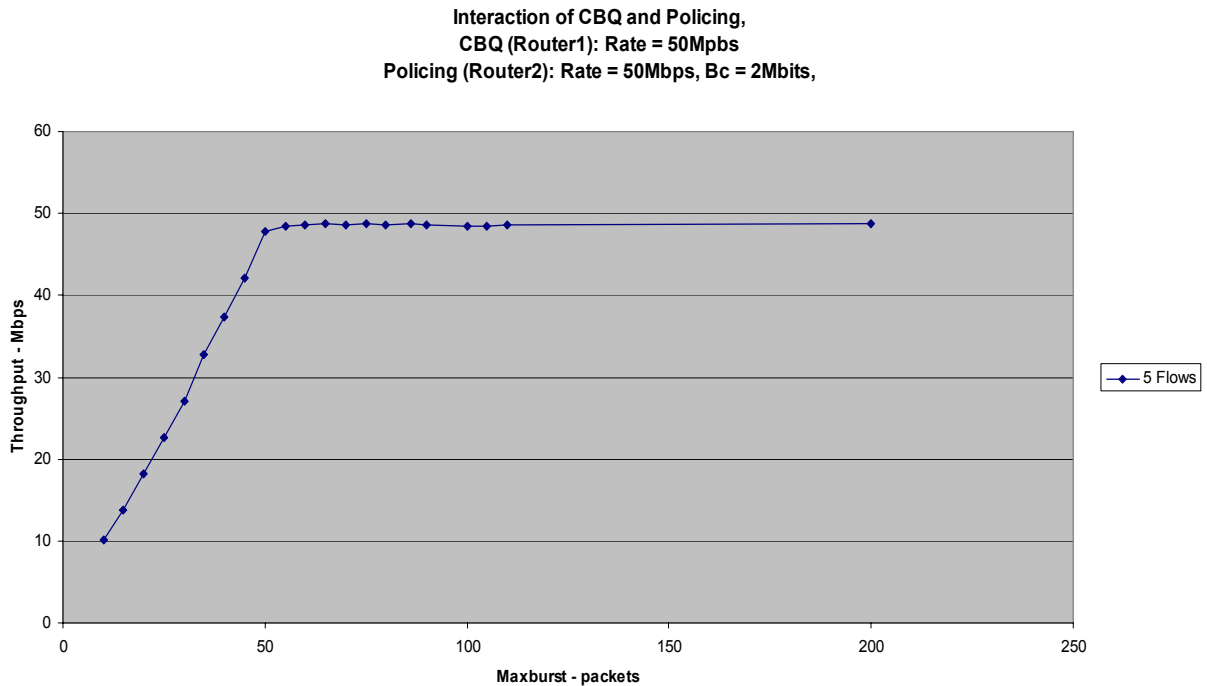
In this section we present results illustrating the interaction of Class Based Queuing with Policing. As shown in figure below, class based queuing is applied at the egress interface of the first-hop router while policing is applied at the ingress interface of the next-hop router.



**Figure 32: Interaction of Class Based Queuing and Policing**

Figure 33 shows the aggregate throughput as a function of the parameter *maxburst* of the Linux CBQ mechanism.





**Figure 33: Throughput as a function of *maxburst* CBQ**

As we can see from the above figure the interaction of Class Based Queuing with policing differs from the interaction of shaping with policing, for large burst sizes. We parallelize the *maxburst* parameter with the *Committed Burst (Bc)* parameter of the token bucket mechanism used by Traffic Shaping. For small values of *maxburst* we have a low aggregate throughput (as in the interaction of shaping and policing), but for large values of *maxburst* the throughput remains the same, and does not decrease as it did in the case of traffic shaping.

## 5 CSC /Funet: Testing LBE in IPv6 Network

### 5.1 Previous work in IPv6/QoS

CSC has previously tried and tested many of the IPv6 related QoS features at the Juniper and Cisco routers in a lab environment, as reported in deliverable D4.4.1. Tests were mainly focused on Juniper routers and final results showed that classification, queuing and marking mechanisms worked as expected.

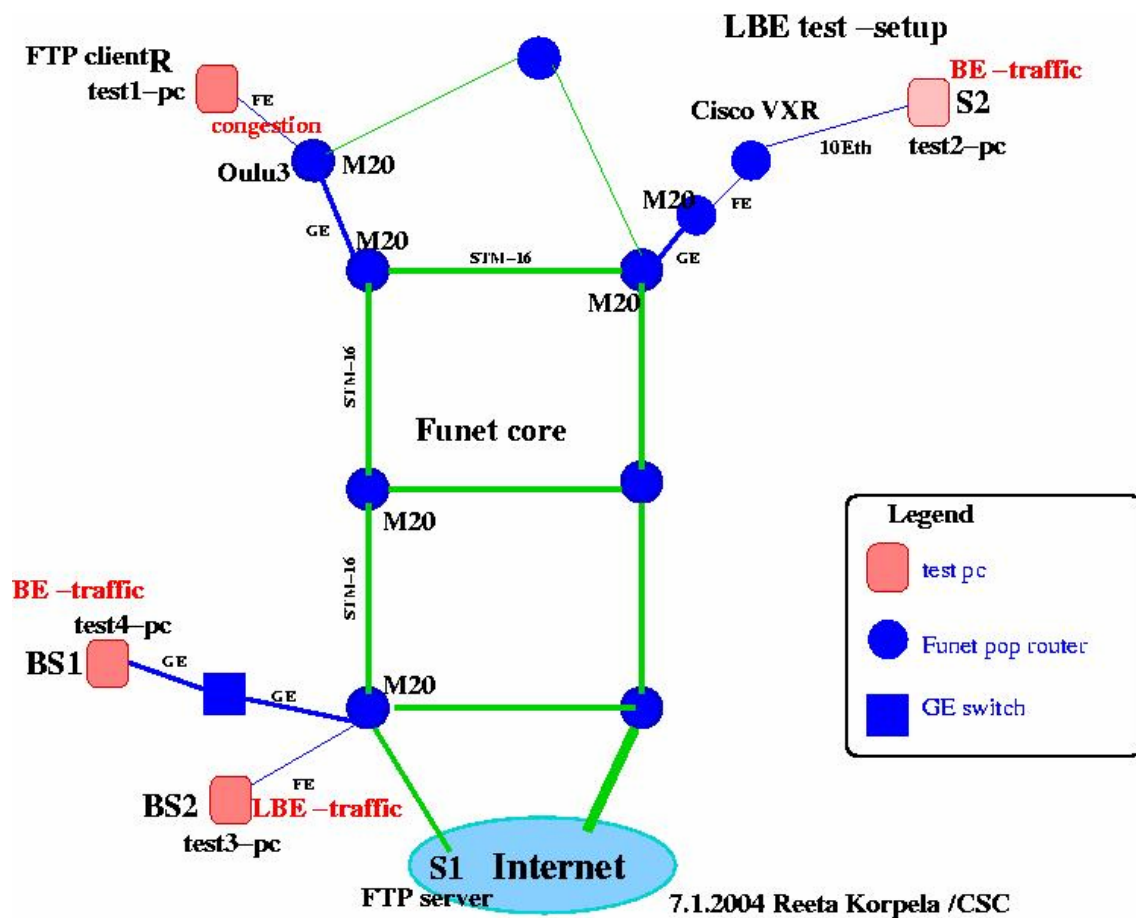


Figure 34: CSC network topology – LBE tests

## 5.2 Report on testing of the LBE concept in Funet production network

### 5.2.1 The goal of this test

The idea of the performed test was to examine the LBE concept as a whole, in a larger scale environment using a real application. Our intention was to measure the effect of the LBE marked traffic with some real network application in case of congestion. During the tests, we used FTP because it was easy to measure the time required for a complete file transfer from the server to the client. Note that the impact of LBE traffic on BE traffic has been previously tested in the GEANT backbone [18].

### 5.2.2 Topology

The test topology is depicted in Figure 34. The test was performed in Funet production network, which consists mainly of Juniper routers, though few Cisco routers exist. The Funet core interconnection links are STM-16/PoS. Link capacities in the access networks varied as shown in the topology figure (Figure 34). Note that the most part of Funet core network is IPv6 capable and the test traffic was IPv6, except the traffic generated with the source S2 as it was behind an IPv4-only router.

### 5.2.3 Equipment

#### Routers

The core routers were Juniper M20 with JunOS 5.6R2.4. Only one of the routers used was a Cisco7200 VXR. This router was deployed next to test2-pc (S2).

#### Traffic generators

There were four test PCs, labelled as “test[1-4]-pc”, located in different PoPs of the network (Figure 34). These PCs were used only for test purposes. “Test[1-3]-pc” were rack-mountable Dell PowerEdge 2450 (Intel P3 728 MHz, 126 MB RAM). “Test4-pc” was an IBM x-Series 342 8669-4RX server (Intel P3 1,26GHz, 64bit PCI bus).

For the test traffic generation, we used the Iperf[7]. Iperf is a tool able to measure IP bandwidth using UDP or TCP protocols. It also supports IPv6.

All the traffic was destined to one receiver in Oulu (test1-pc). The congestion took place at the Fast Ethernet link between Oulu3 router (M20 Juniper) and the labelled “test1-pc”. Thus, the amount of traffic needed for congestion was reduced. Also, this way the production traffic wasn't disturbed during the tests. There was not congestion at the core links between Funet PoPs. The traffic load from the sources towards the destination was 5-15 % of the core link capacity.

### 5.2.4 Test procedure

During test, we established multiple traffic flows to a single receiver and examined one particular flow, the FTP transfer flow. The receiver was fetching data from an external IPv6-only server.

A big file, approximately 30Mbytes, was used so as the transfer time was long enough to minimise inaccuracies at the measurements.

Also, we performed several measurements under the same network conditions and we noticed that the transfer time was almost constant in all cases. So, we concluded that the momentary load of the server or congestion in the server side did not affect the results.

The LBE configuration in the Oulu3 router was the following, queuing using Weighted Round Robin:

- 94% for the best-effort traffic (BE)
- 5% for the network control traffic
- 1% for the less-than-best-effort traffic (LBE)

The class-of-service configuration of the Oulu3 router is provided in appendix B.

#### Appreviations:

- R=receiver
- S= source
- BS= bulk source

---

R: test1-pc

Traffic flows:

S1: external FTP server	-> R IPv6 TCP FTP BE
S2: test2-pc	-> R IPv4 TCP iperf BE
BS1: test4-pc	-> R IPv6 UDP iperf BE
BS2: test3-pc	-> R IPv6 UDP iperf LBE

All the traffic flows were destined to the test1-pc. The traffic flow from test2-pc was IPv4.

The test consisted of three parts I, II and III. In part I we measured the file transfer time without the disturbing bulk-traffic from bulk sources BS1 or BS2. In part II, besides the TCP traffic S1, S2, there was a bulk traffic flow BS1 from source test4-pc (BE). In part III, the bulk traffic originated from the bulk source BS2 (LBE). All parts of the test were repeated multiple times.

#### 5.2.5 Part I: Sources

Two TCP traffic flows:

S1 IPv6	BE
S2 6.80Mb/s	BE

The time of the file transfer was 28 sec +- 1 sec. The accuracy of the measurement was 0.1 sec.

#### 5.2.6 Part II: Sources

S1: IPv6	BE
S2: TCP 6.80Mb/s	BE

in addition:

BS1: 100Mb/s	BE
--------------	----

In this case the FTP file transfer was never completed.

#### 5.2.7 Part III: Sources

S1: IPv6	BE
S2: 6.80Mb/s	BE

in addition:

BS2: 100Mb/s	LBE
--------------	-----

The result in this case was that the LBE marked bulk traffic did not affect the file transfer time at all within the measurement accuracy. From the router interface, it could be noticed that BE traffic passed the router without any losses and the rest of the FE link was filled with LBE bulk traffic.

---

### 5.3 Conclusions

As a conclusion, the concept of the LBE PHB worked fine in this case of FTP transfer. Also, the result was exactly the same as measured with IPv4 protocol. In addition, this test gives valuable information to the ongoing Funet QoS implementation.

### 5.4 Future work

Next thing to do could be testing with other essential network protocols. Also measuring the effect of excess LBE-marked traffic on quantities like delay and jitter would be interesting.

## 6 6NET Quality of Service Framework

### 6.1 Introduction

Several approaches for supporting QoS in best effort IP networks have been developed and described in the literature, and a number of research projects could demonstrate their viability in the context of IPv4 networks. In principle, the same results should apply in the case of IPv6. 6NET is addressing this issue by actually demonstrating QoS support in IPv6 networks.

The goal of the IPv6 QoS activity is to show that approaches for supporting advance services, which had been demonstrated for IPv4, smoothly migrate to the IPv6 environment. At the first phase of the project several QoS mechanisms were examined in local trials. It was demonstrated that QoS mechanisms in IPv6 routers work as expected and, hence, they can be taken as basis for wide-scale deployment of QoS services. The next step is inevitably the deployment of QoS services in the actual 6NET testbed. It has to be validated that adequate QoS support can be realized in the 6NET network and the special requirements of various applications, e.g. real time applications, can be fulfilled.

The QoS mechanisms to be applied in the 6NET backbone will adhere to Differentiated Services (DiffServ) architecture, which ensures scalability and efficiency required in this portion of the network. Based on the DiffServ architecture, 6NET QoS activity proposes a suitable QoS framework, which defines a small set of QoS classes that will be deployed in the core network.

In its initial stage, the deployed QoS service model will comprise of three different service classes. A high priority service, based on EF-PHB, will be provided to a small portion of network traffic, especially for traffic generated with real time applications. BE (best effort) and LBE (less than best effort) services can be used by elastic applications that do not require delay or bandwidth guarantees.

An initial test plan for 6NET is presented in Appendix A. Multiple district tests are defined in order to verify that basic QoS mechanisms, such as traffic policing, are performing as expected in the 6NET WAN environment. Most of the tests are “objective”, which means that standardised methodology is followed in order to evaluate the performance of the network. Software tools are used to generate testing (and background) traffic and collected data is analysed according to the IETF IPPM WG standards. Furthermore, a small set of tests will be performed with real time applications, e.g. videoconference, and end-users will be asked to “subjectively” score the QoS performance services in the 6NET network.

## 6.2 The DiffServ QoS Architecture

The 6NET backbone will adhere to the Differentiated Services (DiffServ) framework in order to offer Quality of Service (QoS) to different portions of traffic passing through the access and core network.

The DiffServ architecture minimizes the number of actions to be performed on every packet at each node and builds a configuration that does not use a signaling protocol. Individual DiffServ mechanisms are applied on traffic aggregates rather than individual flows.

DiffServ operates on a per-hop basis, with each router examining the IPv6 Traffic Class bits in each packet header to obtain the DiffServ code point (DSCP) for that packet. When the packet is ready to be queued for transmission, the router places the packet into an appropriate queue according to the DSCP value.

Service definitions can be specified with relative simplicity, and therefore allow for the service deployment in a short timescale. The end-to-end path is composed of parts that belong to consecutive Diffserv-enabled domains. No particular knowledge is required about internal topologies, physical structure or transmission technology of each of the external domains.

Where DiffServ mechanisms are deployed across network borders, close co-ordination is needed to ensure that each domain is aware of the DSCP values used for each class of service from other domains. If the DSCP value for the same service is different in two neighbor networks, all traffic passing through the network boundaries must have the DSCP value changed (remarked) to the correct local value.

A number of decisions are required prior to implement a DiffServ-enable QoS network; how many classes of service should be provided, and how and where should the bandwidth limit for each class be set? If a limit is exceeded, should out-of-profile packets be dropped or be placed in the lower quality queue? On the one hand, if out-of-profile packets are dropped, network resources may be underutilised. On the other hand, if high-priority (EF) out-of-profile packets are placed in a low quality queue, then packet reordering may take place and bandwidth may also be wasted. (Note that real time applications usually drop out-of-sequence packets).

Use of high-priority CoS should be strictly controlled to prevent their use by unauthorised data flows. For example, a commercial service provider would not accept traffic from a customer that is not paying for premium services. This implies that some form of traffic control is needed, to prevent abuse of preferential queueing services. Traffic control can be "undertaken" by the sending side (e.g. use of "shaping") or by the network receiving the traffic (e.g. use of "policing").

Policing admits traffic up to a certain bandwidth rate limit, and then (i) either drops packets (usually the default), or (ii) places them in a different queue or (iii) (re)marks packets with a higher discard precedence. As policing drops packets indeterminately, it can have a serious effect on throughput and deteriorate services to the preferential traffic class. In most cases, the sending side shapes the traffic, using buffering techniques. This smoothes out bursts of traffic, keeping the traffic rate within the limit so that packets are not dropped by policing.

Policing is usually performed according to three parameters: IP source, IP destination prefixes and agreed sending rate. Policing is performed by means of a token bucket. The token bucket depth is chosen larger than one MTU and varies according to the provided services. For example, high priority services usually have smaller bucket size in order to minimize jitter. Also, bucket size is usually larger in the access than in the core interfaces. The size of the bucket size may influence the packet loss, at the price of a small increase of delay variation. This can be experimentally verified.

At the initial policing point, packets successfully admitted to the service are marked with an appropriate DSCP value. It is possible for the policing configuration to simply enforce the bandwidth rate limit, and trust the DSCP values sent from the other domain. This would obviously be open to abuse from unauthorised sources marking their packets for preferential service.

Where DiffServ is in use between two network service providers, it is common for each provider to police ingress traffic to a reasonable limit, to prevent mis-configurations or denial of service (DoS) attacks from degrading every class of service on the network.

In many cases, the sending host or source should be required to shape flows it sends according to its allowed sending rate. While shaping by the router at the edge of the DiffServ domain should prevent traffic being policed in the next domain, shaping on a per-host basis should help to ensure fair access to the class of service between several hosts in the same domain.

### 6.3 6NET QoS Model

In the context of the QoS activity, WP4 partners will try to deploy a limited number of traffic services in the core network and, when it is required, to extend them to the access networks. On the one hand, the number of services supported in the network is intentionally kept small in order to minimise complexity but, on the other hand, it is adequate for testing equipment functionality and supporting targeted applications in an IPv6 environment.

Therefore, 6NET will initially support three classes of service (in descending order of quality):

- IP Premium (IPP)
- Best Effort (BE)
- Less than Best Effort (LBE)

IP Premium is a service that provides minimum latency and negligible packet loss to traffic and is suitable for real-time applications that require strict QoS guarantees. It may also be used to offer the equivalent of an end-to-end virtual leased line service at the IP layer across multiple administrative domains. IP Premium is a service defined at the IST SEQUIN project [21] and currently implemented for QoS treatment of IPv4 traffic over GEANT (the pan-European Research Network).

Best Effort (BE) is a service that does not provide any qualified guarantees to traffic and is suitable for elastic Internet applications, such as email applications. BE is based on the model of fair resource sharing where “sufficient” resources are available and applications are not starved by other greedy applications running in the background.

Less than Best Effort (LBE) is a service that exploits network resources without risk of jeopardizing or negative impact other traffic classes in the network. Typically, LBE service is used for background operations or bulk data transfer applications that require large scale network resources. Having less strict timing requirements, LBE applications behave in a “greedy” way, i.e. trying to expand as long network resources are available. However, LBE applications can quickly back down to a minimum share if other (higher priority) traffic is present. Note that a minimum bandwidth share is allocated to LBE traffic to prevent complete starvation of LBE flows, which would break established connections.

Apart from the above three traffic CoS, 6NET may attempt to support a DiffServ Assured Forwarding (AF) service in the future. The deployment of the AF-based service will follow the successful implementation of the aforementioned CoS and under the condition that there is explicit demand from (application) end users.



## 6.4 Class Specifications

### 6.4.1 IP Premium - Expedited Forwarding (EF)

The Premium IP implementation is based on the Expedited Forwarding (EF) Per- Hop Behaviour of the DiffServ framework and IPP packets are forwarded immediately by the backbone routers, provided that IPP traffic does not exceed its maximum acceptable rate. IP Premium traffic is marked with DSCP value of “46” and it will be mapped to a dedicated queue at each output interface in the backbone. In 6NET, IPP traffic will be allocated up to 30 Mbps at the core links, which is approximately 20% of the available bandwidth at the STM-1 core links. As this class of service is intended for the use of real-time audio and video traffic, policing will be configured to drop out-of-profile traffic, i.e. traffic exceeding the permitted access rate of a partner. An alternative action for out-of-profile traffic would be to remark it as best effort traffic. However, this would lead to packet reordering that significantly impacts the performance of TCP flows. Also note that out-of-order packets are of no use in real time applications.

IP Premium provisioning model is realised in three steps. Initially, a partner requests to use a portion of traffic that enters and exits from two specific 6NET access routers. The 6NET NOC will estimate whether there is enough resource across the network and, if yes, it will install the appropriate policers at the 6NET edge interfaces. Obviously, the provisioning model follows the “destination aware” approach and the admission control is performed manually by the 6NET network administrators.

### 6.4.2 Default/Best Effort (BE)

Best effort traffic is marked with DSCP value of “0”. No policing of BE traffic is configured, so BE traffic has access to all bandwidth not occupied by higher priority traffic classes.

Note that all traffic not authorised for accessing other QoS classes will be marked as BE by the 6NET access routers.

### 6.4.3 Less than Best Effort (LBE)

LBE traffic will be marked with DSCP value of “8” and it is allocated a minimum portion at the core links, approximately 3% of the total link capacity. Although some networks make no minimum reservation for LBE traffic, a small reservation such as this should enable large data flows over TCP to continue at a low bit rate.

## 6.5 QoS Semantics – DSCP values

The “Traffic Class - TC” one-byte field in the IPv6 packet header has the same semantics with the “Type of Service – ToS” field in the IPv4 packet header. The six (6) most significant bits in the TC byte define the DSCP value of the packet. For the IPP, BE and LBE CoS, the DSCP values in the IPv6 packet headers should be set according to the following table (Table 9):



DSCP bits						Decimal value of DSCP	Description
1	0	1	1	1	0	46	IP Premium
0	0	0	0	0	0	0	Best Effort
0	0	1	0	0	0	6	Less than Best Effort (LBE)

**Table 9: DSCP values for IPP, BE and LBE services.**

The following mechanisms should be configured at the input interfaces of 6NET edge routers (in the direction from the NRENs towards to 6NET):

- Police IPP traffic up to a predefined level according to already granted partners requests. Exceeding traffic will be discarded
- Traffic with invalid DSCP values is remarked as best effort.

The following mechanisms should be configured at internal input interfaces of 6NET domain:

- Police IPP traffic up to 30Mbps in the STM-1 links. Exceeding traffic will be discarded

The following mechanisms should be configured at internal output interfaces of 6NET domain:

- IPP traffic is serviced via a strict priority or a high priority queue.
- LBE traffic is granted at least 3% of the link capacity during heavily congested periods.

## 6.6 Traffic Metrics

There are multiple performance metrics proposed to measure the services provided in a QoS-enabled networks. Most of them are defined by IETF IP Performance Metric working group [20].

During the 6NET tests, the following parameters will be used to qualify the QoS services provided:

- One-way or round trip delay
- Inter-packet delay variation (jitter)
- Packet loss
- Packet reordering

One-way delay is defined as the time needed by a packet to be transmitted and fully received by the destination. The overall time consists of the propagation delay, e.g the time to transmit a bit over long-distance circuits, and the transmission time, e.g. the time to transmit a bit over a specific-speed circuit. As one-way delay measurements require strict synchronization among the monitoring systems, round-trip delay measurements may be performed, instead.

Inter-packet delay variation is measured for packets belonging to the same packet stream and shows the difference in the one-way delay that packets experience in the network. Large values for jitter usually reveal queuing delays in the network.

Packet loss is measured as the portion of packets transmitted but not received in the destination compared to the total number of packets transmitted. Large values of packet loss usually shows highly congested networks or frequent sharp increases of the traffic load. Packet loss is measured as the portion of packets lost in the network (or delayed more than a specific time threshold) compared to the number of packets successfully delivered their destination. Packet loss usually reveal congestion in the output queues of the routers.

Packet reordering is measured as the portion of packets that are delivered to the destination in wrong order compared to the total number of packets. There are multiple reasons that lead to packet reordering; parallel forwarding engines in high performance routers, per packet load balancing on parallel physical links, routing path changes, etc. There is a significant impact to the TCP (application) performance even for small packet reordering values.

## 6.7 Application Profiles

The IP Premium (EF) class of service is likely to suit interactive, real time voice and video applications, which are not tolerant of packet loss or variations in one-way delay (jitter). Therefore, the final testing (after service's validation) will be performed using the Gnomemeeting videoconferencing application[19]. GnomeMeeting is based on the OpenH323 library, a full featured, interoperable, Open Source implementation of the ITU-T H.323 teleconferencing protocol that can be used by personal developers and commercial users without charge. It is an H.323 compatible videoconferencing and VOIP/IP-Telephony application that allows you to make audio and video calls to remote users with H.323 hardware or software (such as Microsoft Netmeeting). It supports all modern videoconferencing features, such as registering to an ILS directory, gatekeeper support, making multi-user conference calls using an external MCU, using modern Quicknet telephony cards, and making PC-To-Phone calls. Finally, Gnomemeeting as well as OpenH323 library has been ported to IPv6 and tested in the concept of Work Package 5 of 6NET project.

As already noted, LBE is primarily expected to be used for large scale data transfers, where the time constraints for the transfer are very relaxed, provided the transfer does actually complete.

## 6.8 Monitoring and Measurement Requirements

Subscription to a premium class of service implies a Service Level Agreement (SLA) is signed between the customer and the service provider. On 6NET, provided SLAs are clearly not commercial agreements but only define performance guarantees that are experimentally provided to portions of traffic.

Monitoring and measuring activities in 6NET should demonstrate the network infrastructure is able to provide service guarantees to portion of traffic. For example, traffic marked as EF receives preferential treatment compared to BE and LBE, under congestion conditions in the core or the access networks. Also, LBE traffic should also be reduced to a minimum level when other traffic is present.

This may be possible to measure using Cisco's Service Assurance Agent (SAA). SAA runs on Cisco routers and allows particular traffic tests to be configured between a sending (the "sender") and a receiving router (the "responder"). For example, one test will send bursts of ICMP echo requests (or pings), which can be used to monitor latency or one-way delay. Another test allows bursts of UDP packets to be sent, and could be used to simulate a voice over IP (VoIP) call. Results of the tests can be obtained using SNMP.

It should be noted that SAA is supported only for IPv4 traffic. A work-around solution is to perform SAA measurements over IPv4 in IPv6 tunnels, which causes extra latency for the traffic using these tunnels, but this should not be important. In most cases, the information of interest in QoS monitoring is the relative changes in delay statistics or simply the packet loss occurring when traffic uses different classes of service and not their actual values. Another problem, which is related to the tunnelling requirement, is that there needs to be a mechanism that is able to mark the TC field of IPv6 packets according to the ToS field of the IPv4 packets. If this is not possible in 6NET routers, SAA measurements may be impossible or very cumbersome to be performed. Furthermore, in a CEF<sup>1</sup>-enabled router, such as GSRs, SAA traffic does not follow the default forwarding paths but it follows an alternative path that passes through the CPU. Therefore, SAAs measurements may become inaccurate during CPU-intensive periods. As a workaround solution, shadow routers co-located with backbone routers at testing points may be used. In such case, latency effects due to CPU loading are avoided. For example, JANET has used Cisco 805 shadow routers for network performance monitoring in both QoS and content delivery trials.

QoS measurements can also be performed with software tools that are able to generate traffic with pre-defined characteristics and measure the performance of the network. A tool that is already extensively used in measurements is *iperf*, which is able to provide accurate throughput and jitter measurements for flows under test. Other software tools are also available and may be used if required. Furthermore, the 6NET partner FOKUS has built various network monitoring applications that are suitable for passive monitoring/measuring the performance of the 6NET network. These applications will provide accurate information at packet level for the network behaviour and it will leverage the analysis of collected results.

The exact methodology for monitoring the provided services in the 6NET network is not finalised up to now as technology obstacles have been identified. However, possible workaround solutions are already been proposed but need to be further investigated.

---

<sup>1</sup> Cisco Expedite Forwarding.

## 7 References

- [1] “Enhancing the DiffServ architecture of a simulation environment”, C. Bouras, D. Primpas, A. Sevasti, A. Varnavas, 6th IEEE International Workshop on Distributed Simulation and Real Time Applications, Fort Worth, Texas, USA, October 11 – 13, 2002
- [2] K. Thomson, G.J. Miller and R. Wilder, “Wide Area Internet Traffic Patterns and Characteristics” in IEEE/ACM Transactions on Networking, pp 10-23, 1997
- [3] <http://www.cisco.com>
- [4] <http://ouranos.ceid.upatras.gr/diffserv/start.htm>
- [5] <http://ouranos.ceid.upatras.gr/openh323/>
- [6] <http://www.isi.edu/nsnam/ns/>
- [7] <http://dast.nlanr.net/Projects/Iperf/>
- [8] S. Vegesna, ‘IP Quality of Service: the complete resource for understanding and deploying IP quality of service for Cisco networks’, Cisco Press, 2001
- [9] C. Bouras, A. Gkamas, D. Primpas, K. Stamos, “Performance Evaluation of an IPv6 –capable H323 Application” The 18th International Conference on Advanced Networking and Applications (AINA 2004), Fukuoka, Japan, March 29-31 2004 (to appear)
- [10] <http://www.csl.sony.co.jp/person/kjc/kjc/software.html>
- [11] S. Floyd, R. Gummadi and S. Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED. Technical Report, to appear (2001).
- [12] M. A. El-Gendy and K. G. Shin. Assured Forwarding Fairness Using Equation-Based Packet Marking and Packet Separation. Computer Networks The International Journal of Computer and Telecommunications Networking. 41(4): 435-450. March 2003.
- [13] J. Heinanen and R. Guerin. A Two Rate Three Color Marker. Informational RFC 2698, IETF, September 1999.
- [14] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems, DEC Research Report TR-301, September 1984.
- [15] O. Medina. Etude des algorithmes d'attribution de priorité dans un Internet à Différentiation de Services. Ph.D. Thesis, Université de Rennes I, Rennes, France, March 2001.
- [16] O. Medina, J. Orozco, D. Ros. Bandwidth Sharing under the Assured Forwarding PHB. Technical Report IRISA, No1478, September 2002.
- [17] J. Orozco, D. Ros. An Adaptive RIO (A-RIO) queue management algorithm. 4th COST 263 International Workshop on Quality of Future Internet Services (QoFIS), Springer, Lecture Notes in Computer Science, 2003.
- [18] <http://www.cnaf.infn.it/~ferrari/tfngn/lbe/results/lbe-geant/>
- [19] GnomeMeeting Home page, <http://www.gnomemeeting.org/>
- [20] IP Performance Metrics (IPPM) Working Group, <http://www.ietf.org/html.charters/ippm-charter.html>, IETF.
- [21] Service Quality across Independently managed Networks, <http://archive.dante.net/sequin/>

## **8 Appendix A: 6NET Quality of Service Testing Plan**

### **8.1 Introduction**

This section defines the Quality of Service (QoS) tests for validating the most significant QoS mechanisms, e.g. marking, policing and shaping, at 6NET access and backbone routers. The test will mainly try to verify that IPv6 traffic can be given the same performance guarantees as IPv4 traffic, which means that QoS mechanisms are performing as expected for IPv6 traffic. A secondary objective for the tests is to validate that prioritisation can be provided to portions of traffic crossing the 6NET core (or access) network.

Note that the following proposed tests (a) do not investigate the performance of QoS mechanisms, i.e. what kind of guarantees may a mechanism provide to traffic, (b) do not compare the performance of similar purpose mechanisms and (c) do not investigate under which conditions QoS guarantees are met or violated in a large scale network.

### **8.2 Implementation of Framework**

The implementation of the QoS framework requires configuration at the 6NET core and access routers, at least for the portion of the network that is used during the QoS tests. QoS testing will be a complex and time consuming process, so it will be necessary to book the 6NET network for testing for two weeks.

To prevent problems caused by misconfiguration, the policing and DSCP marking/re-marking will be configured first, on the 6NET access routers. If the preferential queueing were to be enabled in the core before the edge policing, all packets marked with the EF DSCP value would start to be forwarded in the priority queue. If the EF queueing were misconfigured in some way, this would have the potential to cause disruption on the network.

Once the edge policing and marking has been established, preferential queueing can be enabled on the core. Configuration within partners' networks can be done at any time leading up to the start of testing.

### **8.3 Traffic generators**

In order to perform the QoS tests, the most suitable traffic sources have to be identified. Traffic sources should be able to generate data streams that “simulate” backbone traffic and/or specific application traffic profiles, e.g. VoIP calls. Also, traffic sources should be able to generate traffic that can congest the network and stress the mechanism enabled at the routers. For 6NET tests, there are the following alternatives:

- Embedded software in routers
- Software tools
- Hardware traffic generators
- Real applications

Cisco's Service Assurance Agent (SAA) is embedded software in Cisco IOS that can be used to generate UDP streams and perhaps to simulate VoIP calls. SAA allows the DSCP value to be

configured on generated streams, although the DSCP value could also be set as the packet is forwarded through a router interface. SAA do not generate traffic that could congest 6NET network. However, SAA may be used to monitor provided performance guarantees during QoS tests. SAA could be run on the 6NET routers or partners may also be able to make other SAA agents available during testing.

There are many software tools, such as *iperf*, that can be used to generate traffic in large amounts. A common server equipped with a gigabit Ethernet interface could be able to create congestion to 6NET STM-1 core links. Packet loss and bandwidth measurements are easily performed with the *iperf* tool.

Hardware traffic generators may be used for performing the QoS test. There are few vendors that already support IPv6 protocols and could be used during testing. However, there are no traffic generators available for testing in 6NET.

Finally, real-time applications may be used to perform the QoS tests, especially for qualitative assessing the performance of the network. Also, bulk transfer applications, e.g. ftp, may be used for estimating the throughput guarantees provided.

Note that JANET (UKERNA) has two Linux servers intended for this generating traffic; other partners may also have suitable equipment. These traffic generators can be located at partner sites; it should be possible to route traffic streams from these generators to different locations on 6NET, thereby causing congestion on desired links.

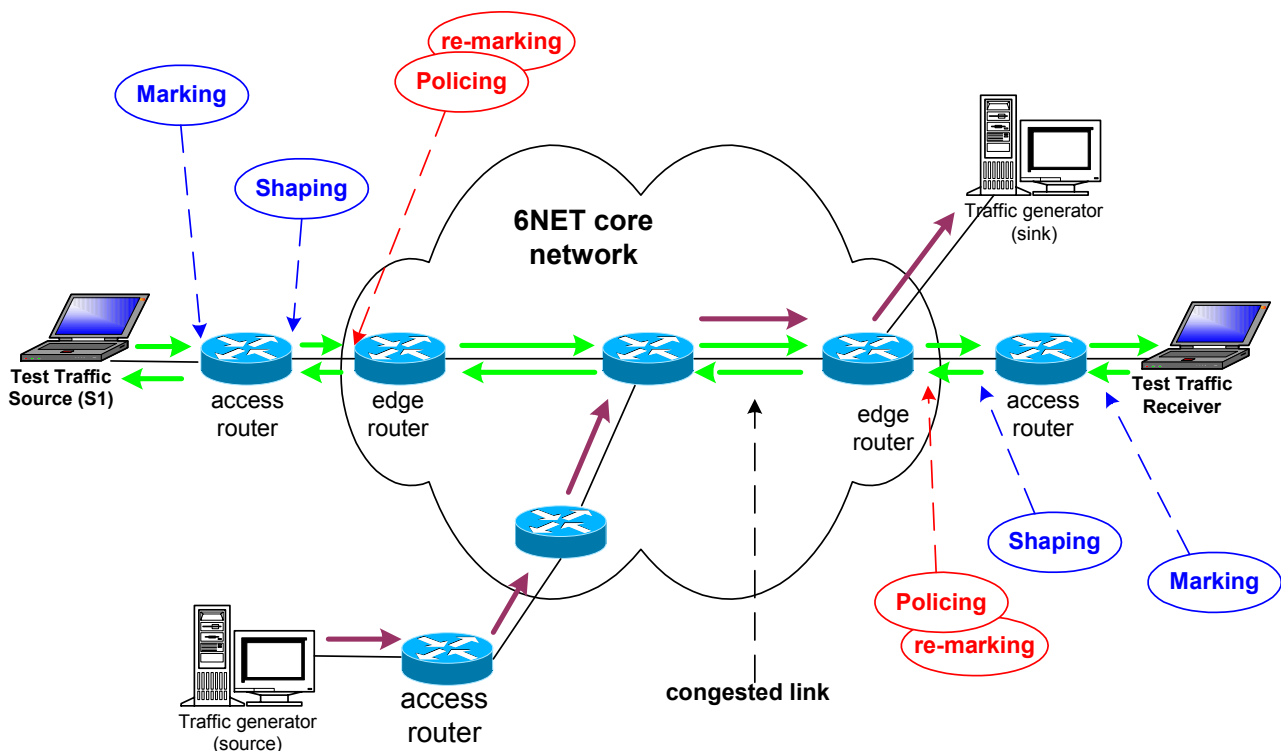


Figure 35: 6NET test network – QoS sources and mechanisms

---

## 8.4 Passive Monitoring

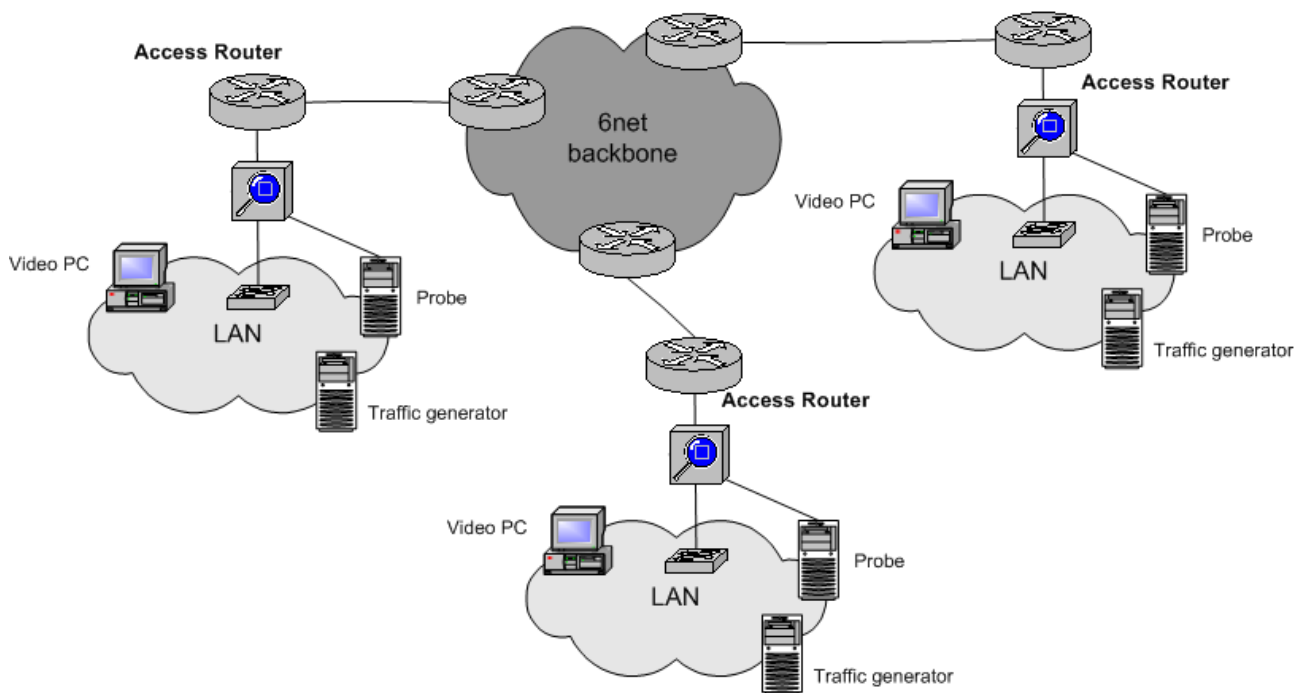
The QoS trials in 6NET will be supported by passive monitoring probes, which will be installed in the network. In contrast to active measurements, which are based on the principle of injecting test packets into the network, there is no need for special test traffic in the passive measurement approach. Passive measurements observe the traffic that is already in the network and directly derive QoS metrics from the genuine application traffic.

In general, active and passive methods are complementary and in many contexts both approaches will be needed, e.g. in trouble-shooting, where passive monitoring can help in fault detection, but for isolating the exact fault location controlled probing and testing using active methods will also be needed.

The passive approach is a particularly valuable tool when auditing SLAs and validating QoS. Operating directly on the application traffic there is less risk of skewed measurements as compared to active methods where QoS metrics could potentially deviate from each other as effect caused by differences in the statistical flow properties (packet size, distributions, rates,...) between application traffic and test traffic.

In the 6NET QoS trials involving user applications for the various QoS classes, a passive measurement application (OpenIMP developed by FOKUS) will be used to measure the application traffic and in order to validate that the service provided to the various classes is according to the class specification. The probes can be standard PCs with FreeBSD or Linux OS. The probes have additional network cards installed which are connected to a network tap. The taps have to be installed at the FastEthernet-Links at the ingress/egress of the partners access routers (see figure below). If there are no taps available one can configure an ethernet switch to double the network traffic to a monitor port and connect the probe to that port. The passive measurements can determine bandwidth, packet one-way-delay and loss of the captured traffic flows. In addition, changes of DSCP values are monitored. The precision of the calculated one-way-delays depends on the synchronization method used and the local clocks of the probes.





**Figure 36: Passive monitoring test network.**

## 8.5 Testing QoS mechanisms

### 8.5.1 Marking

Objectives: Validate that marking mechanisms are correctly implemented at access and core routers.

Test No1: Use a traffic generator, e.g. *iperf*, to create a flow with BE packets. Mark packets with IPP or LBE DSCP values at either the access (Ra1) or the core (Rc1) routers.

Test No2: Use a traffic generator, e.g. *iperf*, to create a flow and mark packets with an invalid DSCP value at the access router. Re-mark packets as BE traffic at the core router.


Assessment: Check router statistics via CLI or SNMP.

Success condition: Verify that packets have the correct DSCP value after passing through the routers.

### 8.5.2 Traffic Policing

Objectives: Validate that policing mechanisms are performing as expected at input interfaces. In general, input policing takes place at the upstream providers edge routers towards the customers direction.



32603	Deliverable D4.2.2v1	
-------	----------------------	---

Test No1: Use a traffic generator to create IPP traffic that exceeds the bandwidth rate limit configured at the input interface at access router. Exceeding traffic is discarded.

Test No2: Use a traffic generator to create BE traffic that exceeds the bandwidth rate limit configured at the input interface at access router. Exceeding traffic is remarked as LBE.

Assessment: Verify that packets are dropped using the router CLI or SNMP and/or observe egress traffic rate over time to ensure that it remains within the limit.

Success condition: Traffic is accepted up to the agreed rate limit; traffic exceeding this limit is dropped or remarked.

### 8.5.3 Shaping

Objectives: Validate that shaping mechanism is functioning as expected at output interfaces of partner's access routers and at the input interfaces of the 6NET core routers. In general, output shaping takes place at the customer's side, preferable as much as close to the source. Input shaping is performed by the upstream provider as an additional service to his/her customers, usually when customers are not able to perform output shaping in their routers.

Test No1: Use a traffic generator to generate bursty UDP traffic. Apply shaping at the access router and verify that traffic forwarded to the core domain remains within the bandwidth limit.

Assessment: Verify that the received traffic rate at the next hop remains at a steady rate, and is not policed. CLI or SNMP at the router may be used.

Success condition: Traffic is forwarded always within the rate limit. Packet loss may occur at the output interfaces for some periods.


## 8.6 Test with non-BE Traffic Classes

### 8.6.1 Premium/Expedited Forwarding

Objectives: Verify that IP Premium traffic is forwarded up to its allocated bandwidth when the network is congested.

Test No1: Use a traffic generator, e.g. *iperf*, to congest the network links with UDP traffic. Artificial traffic produced with *iperf* is marked as BE and it used as background traffic. Introduce one TCP flow and mark its packets as IPP. Note the achieved throughput.

Test No2: Use a traffic generator, e.g. *iperf*, to congest the network links with UDP traffic. Artificial traffic produced with *iperf* is marked as BE and it used as background traffic. Introduce one TCP flow and mark its packets as IPP. Monitor traffic levels to verify that IPP traffic takes precedence over other traffic. Verify that there is not packet loss at the traffic policers on the access or core routers using CLI statistics. Note the achieved throughput.

32603	Deliverable D4.2.2v1	
-------	----------------------	---

Test No3: Use a traffic generator, e.g. *iperf*, to congest the network links with UDP traffic. Artificial traffic produced with *iperf* is marked as BE and it is used as background traffic. Introduce one UDP flow and mark its packets as IPP. Gradually increase the packet rate and note the maximum delivered rate at the destination.

Assessment: Compare the achieved throughput at Tests 1 and 2. Also, at Test 3 compare the maximum achieved rate with the minimum policer rate in the network.

Success condition: IP Premium traffic is forwarded in favour of BE traffic, up to its bandwidth limit.

### 8.6.2 Less Than Best Effort (LBE)

Objective: Verify that LBE traffic backs off to its minimum bandwidth limit when links become congested.

Test No1: Use a traffic generator, e.g. *iperf*, to create UDP traffic that occupies most of the network capacity. Gradually, introduce BE traffic to the network and measure the packet losses for the BE traffic. Note the bandwidth that LBE traffic occupies in the most congested link.

Assessment: Compare the throughput achieved from LBE traffic with the available bandwidth in the most congested link.

Pass: The LBE traffic falls to its minimum bandwidth limit when the network is congested. In all the other cases, it tries to occupy all the available bandwidth.

## 8.7 Application Tests

Objectives: Verify that video/audio streaming during a videoconference is improved when IPP service is used as compared to BE service.

Test No1: Use a traffic generator, e.g. *iperf*, to congest the network links with UDP traffic. Artificial traffic produced with *iperf* is marked as BE and it used as background traffic. Start a videoconference session using GnomeMeeting application. Streaming flow should pass through at least one of previously congested network links. Streaming packet should also be marked as BE. Use *traceroute* to identify the flow path and to measure two-way packet delay. Create another UDP flow with *iperf* and mark traffic as BE. Use this flow to measure quantitative performance guarantees in the network and, thus, note measured delay jitter and packet loss estimated with *iperf*.

Test No2: Use a traffic generator, e.g. *iperf*, to congest the network links with UDP traffic. Artificial traffic produced with *iperf* is marked as BE and it used as background traffic. Start a videoconference session using GnomeMeeting application. Streaming flow should pass through at least one of previously congested network links. Streaming packet should also be marked as IPP traffic (DSCP 46). Use *traceroute* to identify the flow path, and to measure two-way packet delay. Create another UDP flow with *iperf* and mark traffic as BE. Use this flow to measure quantitative performance guarantees in the network and, thus, note measure delay jitter and packet loss estimated with *iperf*.

Test No3: Repeat the previous test but without background traffic.

Assessment: Request a group of people to give their opinion regarding received video / audio quality (qualitative measurements). Also, compare *iperf* measurements for different set of tests.

Success condition: Video and possibly audio quality should be improved with IPP service. Also, packet loss and jitter should also be reduced. Performance metrics for Tests 2 and 3 should be comparable.

## 8.8 Schedule

From	To	Activity Description
	Mon 29/3	Circulate QoS framework and testing plan to WP mailing lists for comments.
	Mon 29/3	Identify individuals that will participate to the tests. Collect contact information.
	Fri 2/4	Identify hardware and software requirements.
	Mon 5/4	Prepare a detailed testing plan, e.g. configuration examples, traffic patterns, etc.
	Mon 19/4	Prepare a prototype QoS configuration for access / core routers
	Mon 26/4	Install the measurement equipment and software.
	Mon 30/4	Prepare a prototype configuration for SAA
	Fri 30/4	Create IPv4 over IPv6 tunnels – Enable SAA monitoring
	Mon 3/5	Enable 6NET router QoS configuration in access
	Mon 10/5	Enable 6NET router QoS configuration in core
	Mon 10/5	Define final time schedule
Tue 11/5	Fri 21/5	Perform a' set of tests - Collect results
Mon 24/5	Mon 31/5	Perform b' set of tests - Collect results
	Wed 2/6	Evaluate monitoring infrastructure data (SAA)
	Tue 5/6	Analyse results
	Wed 30/6	Document

## 8.9 6NET partners involved to the tests


The following 6NET partners have already expressed their intention to participate to the tests: Cisco, GRNET, UKERNA, UCL, ENST, TELIN, CTI, FOKUS, OULU, Uninett, UoLancaster.

---

## 9 Appendix B: Router configuration in LBE tests

The class-of-service -configuration in Juniper M20 routers:

```
class-of-service {
  classifiers {
    dscp backbone-classifier {
      import default;
      forwarding-class best-effort {
        loss-priority low code-points [ ef af11 af12 af13 ];
      }
      forwarding-class less-than-be {
        loss-priority low code-points cs1;
      }
    }
  }
  forwarding-classes {
    queue 0 best-effort;
    queue 1 expedited-forwarding;
    queue 2 less-than-be;
    queue 3 network-control;
  }
  interfaces {
    ge-0/0/0 {
      scheduler-map MAP-BACKBONE;
      unit 0 {
        classifiers {
          dscp backbone-classifier;
        }
      }
    }
    fe-0/3/3 {
      scheduler-map MAP-BACKBONE;
      unit 0 {
        classifiers {
          dscp backbone-classifier;
        }
      }
    }
  }
}
```

32603	Deliverable D4.2.2v1	
-------	----------------------	---

---

```

scheduler-maps {
    MAP-BACKBONE {
        forwarding-class best-effort scheduler sch-best-effort;
        forwarding-class less-than-be scheduler sch-less-than-be;
        forwarding-class network-control scheduler sch-network-ctrl;
    }
}
schedulers {
    sch-best-effort {
        transmit-rate percent 94;
        buffer-size percent 94;
        priority high;
    }
    sch-less-than-be {
        transmit-rate percent 1;
        buffer-size percent 1;
        priority low;
    }
    sch-network-ctrl {
        transmit-rate percent 5;
        buffer-size percent 5;
        priority low;
    }
}
}

```