Quality of Service Aspects in an IPv6 Domain

Ch. Bouras[1,2] A. Gkamas[1,2] D. Primpas[1,2] K. Stamos[1,2]
[1]Research Academic Computer Technology Institute, Riga Feraiou 61, GR-26221 Patras, Greece and
[2]Computer Engineering and Informatics Dept., Univ. of Patras, GR-26500 Patras, Greece
Tel:+30-(2)610-{960375, 960465, 996954, 960316 }
Fax:+30-(2)610-{996314, 960358, 960358, 960358}
e-mail: {bouras, gkamas, primpas, stamos}@cti.gr

Keywords: Quality of Service, IPv6, Open H323, real time applications, jitter, packet loss

Abstract
During the last years 2 Quality of Service architectures (IntServ and DiffServ) have been proposed and evaluated. The DiffServ architecture has proved its ascendancy and some services have already been proposed and deployed on IPv4 domains. But the usage of IPv6 protocol creates new challenges, as the QoS mechanisms that are currently supported for IPv6 in most implementations are fewer (or different) compared to IPv4 and in addition the whole network's behaviour is different. So, as a result the QoS services should be designed and evaluated again. This paper describes a QoS service on an IPv6 domain that aims to service aggregates of real time traffic with minimum delay, jitter and packet loss. In addition, it contains results from the experiments in our IPv6 network that took advantage of the QoS mechanisms. This QoS service uses the Modular QoS CLI (MQC) mechanism and especially the Low Latency Queue feature (LLQ) in order to treat packets from real time applications.

1 INTRODUCTION
Internet traffic consists of flows generated by different applications, which all receive the same treatment from the network, as most networks today can only provide best-effort service. This treatment causes many problems especially to real time applications (for example videoconferencing applications), because they are sensitive on parameters such as delay, packet loss or jitter. For this reason, the QoS techniques are necessary to provide service guarantees in today's congested and increasingly used networks. The QoS guarantees can be measured using a number of specific metrics. These metrics are the bandwidth that a traffic class uses, the delay that the packets of each class experience, packet loss and jitter. During the last years several architectures have been proposed in order to provide QoS and some services have already been deployed. In addition, significant research activity has been done in this area, as there are several publications [1] [2] [3] [4]. One of the next goals in this area, and the topic that we examine in this paper is the investigation of the deployment of a QoS service on an IPv6 domain. The usage of the IPv6 protocol [5] is increasing and many domains have already been IPv6 enabled. The investigation of the supported QoS mechanisms on IPv6 and the deployment of a QoS service, especially for real time applications, is our basic goal. The rest of the paper is organized as follows. Section 2 gives a brief description of the architectures for Quality of Service and concentrates on the DiffServ architecture that we use, presenting the main mechanisms and their operation. Section 3 describes the IPv6 testbed that we used for performing the experiments and explains the QoS techniques and traffic patterns that have been used for the experiments. Section 4 presents the experiments and the results from each one. Finally, section 5 describes the conclusions from those experiments and section 6 the future work that we intend to do on this area.

2 THE QUALITY OF SERVICE TECHNIQUES
During the last years 2 main architectures for Quality of Service, IntServ [6] and DiffServ [7] have been proposed. They follow different philosophy as they approach the topic of Quality of Service from different point of views.
The IntServ architecture tries to provide absolute guarantees via resource reservations across the paths that the traffic class follows. The main protocol that works with this architecture is the Reservation Protocol (RSVP) [6]. However, its operation is quite complicated and it also inserts significant network overhead. On the other hand, DiffServ architecture is more flexible and efficient as it tries to provide Quality of Service via a different approach. It classifies all the network traffic into classes and tries to treat each class differently, according to the level of QoS guarantees that each class needs. In the DiffServ architecture, 2 different types (per hop behaviours [8]) have been proposed, the expedited forwarding [9] and the assured forwarding [10], and their difference is on the packet

forwarding behaviour. Expedited forwarding (EF) aims at providing QoS for the class by minimizing the jitter and is generally focused on providing stricter guarantees. This type tries to simulate the virtual leased lines and its policy profile should be very tight. Assured forwarding (AF) inserts at most 4 classes with at most 3 levels of dropping packets. Every time the traffic of each class exceeds the policy criteria then it is marked as lower level QoS class.

The operation of the DiffServ architecture is based on several mechanisms. The first mechanism is the classifier that tries to classify the whole traffic into aggregates of flows (traffic classes), mainly using the field DSCP (Differentiated Service CodePoint [7]). This field exists in both the IPv4 and IPv6 packet headers. In IPv4 it was part of the field Type of Service (ToS) and in IPv6 that is our focus in this paper, it is part of the field Traffic Class. In addition, the IPv6 packet header also has the field Flow label (20 bits) but it is still experimental and its use has only been recently standardized [11].

The operation of services based on DiffServ architecture uses also several additional mechanisms that act on every aggregate of flows. These mechanisms are packet marking, metering and shaping. In addition, in order to provide QoS guarantees it is necessary to properly configure the queue management and the time routing/scheduling mechanism. The most common queue management approaches use the Priority Queue, Weighted Fair Queue or Modified Deficit Round Robin mechanisms [11][12].

Generally, the main problem that has been noticed is that not all of these mechanisms have been fully implemented to work for IPv6 domains yet. In particular, many vendors only provide Weighted Fair Queue or Priority Queue adding limitations to the administrators. In addition, various mechanisms for IPv6 networks (QoS, multicast etc) are not supported on the same version of router software, so the network administrators can not provide to their consumers advanced services simultaneously. Generally this fact is considered to be temporary and it is expected that will shortly be addressed as the usage of IPv6 increases.

3 THE IMPLEMENTED QOS SERVICE ON IPV6 TESTBED

In this paper we describe the implementation and testing of a QoS service on IPv6 networks. The service is based on the 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 IPv6 network that has been created internally on CTI and is presented in Figure 1. The testbed is also interconnected and uses infrastructure that has been deployed for the IST Project 6NET [13] (a pan-European IPv6 network). The software version that this testbed uses is the CISCO IOS 12.2(13) T [12].

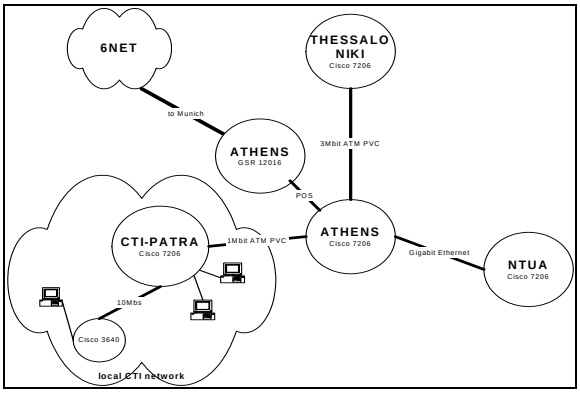

Figure 1. The 6NET's network including CTI's testbed

The QoS service that we implemented aims at providing prioritization for traffic coming from real time applications. Its operation is to classify the packets that belong to this application and use a "priority queue" for them. The rest of the traffic on the router will be treated as usual, with best–effort service.

The service has been implemented using the Class Based Weighted Fair Queuing mechanism [11]. This mechanism actually extends the classic Weighted Fair Queuing mechanism and can provide strict packet priority. The strict priority feature actually implements the Low Latency Queue mechanism, which is a method to en-queue packets on a "priority" queue in order to guarantee low latency and jitter. The service follows the classic guidelines of the DiffServ architecture. We have tried to make the experimental scenarios as realistic as possible by inserting background traffic in the network with the cross traffic method that is a mix of TCP and UDP traffic, generated by the Iperf [14] traffic generator. This traffic is classified with DSCP value 0 (default) and is treated as best-effort. In addition we inserted foreground traffic that simulates an aggregate of real time traffic. This traffic is also a mix of artificially generated UDP traffic and RTP traffic [15] generated by an application based on the OpenH323 library [16], which has been ported to IPv6 [17], [18]. On the network devices, we have applied a 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 with DSCP values: default (000000) and the predefined expedited forwarding (101110) respectively [9], [11]. Next, the output interfaces of the network devices have been configured in order to send the packets that have been marked with DSCP 101110 with strict priority.

The first step in the experimental procedure was to performed many tests in order to investigate and report the operation of the QoS mechanism under different conditions (traffic load, congestion etc). As soon as the investigation was finished, we tried to test this QoS service with traffic

generated by real time applications (OpenH323) and report the results.

After this experimental procedure, the next stage was the implementation of a second testing scenario, where the network devices have been additionally configured in order to apply the Weighted Random Early Detection mechanism [11] for congestion avoidance on the background traffic. In this case the goal is to measure if the existence of WRED makes any impact on the QoS guarantees that the foreground packets experience.

4 EXPERIMENTAL PROCEDURE

The experimental procedure that was followed for testing the QoS mechanisms and the whole service's performance has been applied on CTI's internal IPv6 testbed. This testbed uses infrastructure that has been deployed for IST 6NET project extended with additional CTI routers. The basic testbed that we used consisted of 2 routers, one router of CISCO 7200 series (local 6NET router) and one router of CISCO 3600 series. This testbed is interconnected with CTI's production network and can be shown in Figure 2 that presents it in detail.

The measurement of the network's metrics and the service's performance was done using specific tools. The first one was the statistics of the Iperf traffic generator for the traffic that it was generating. These statistics were produced at the server instance of the Iperf traffic generator [14] and included the average throughput and the average jitter of the UDP traffic and the average throughput of the TCP traffic. Another tool was the RTCP statistics that the videoconference tool was providing and finally the results from the Ethereal Network Protocol Analyzer [19] that captured all the packets at the receiver and provided us graphic representations of their throughput.

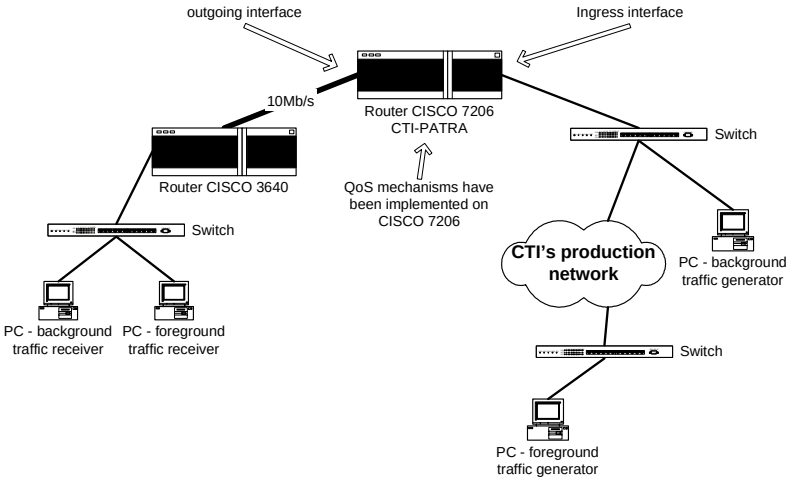

Figure 2. The testbed

Using this infrastructure, we performed many tests investigating the QoS mechanisms that are supported on IPv6. First of all, we tested the prioritization and classification mechanisms performing a large number of tests. Then, these tests became more complex and finally we applied all the mechanisms simultaneously to test the whole QoS service with real time data.

4.1 Investigation of the Prioritization Mechanism

The first tests aimed at investigating the operation of the classification and prioritization mechanisms. The classification mechanism was implemented using 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 (ef (101110) for the foreground traffic and default (000000) for the background). Next, on the output interface of the router, we configured a second policy class that gives strict priority to the packets that have been marked with the DSCP value for EF. In order make 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 was created with Iperf traffic generator and used the UDP protocol with average rate 12Mbps and 1.5 Mbps respectively. The backbone link is 10Mb/s 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 re-enable the QoS mechanisms and investigate the network's behaviour when it only has best effort traffic (UDP or both UDP and TCP). First, we performed tests using only UDP best-effort traffic, generated by the Iperf traffic generator.

Table 1. Best -effort experiments (only UDP traffic)

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

Table 1 presents the experimental results, which show that the actual throughput that can fill the backbone links is 8.80 Mbps. In addition, the packet loss and jitter increases when the network usage exceeds 8.80 Mbps.

Next, those tests were repeated, according to the same scenarios, using both TCP and UDP traffic generated by the Iperf traffic generator. Similarly, Table 2 presents the results. At this point we should notice that while for UDP traffic the traffic generator tries to send the 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.

Table 2. Best-effort (UDP and TCP traffic)

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

According to the 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 consumes its specified bandwidth and the remaining bandwidth is used by TCP.

The next step was to test the classification and prioritization mechanism by adding foreground traffic for different background 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 (with different traffic loads) and some of them are described in Table 3.

Table 3. Testing of classification and prioritization mechanism (UDP traffic)

Traffic load		Actual throughput		Packet loss		Average Jitter	
Background (Mbps)	Foreground (Kbps)	Background (Mbps)	Foreground (Kbps)	Background	Foreground	Background (ms)	Foreground (ms)
8	250	7.98	250	0.0011%	0%	3.191	4.404
9	250	8.58	244	7.6%	2.4%	2.864	4.333
10	250	8.57	239	14%	4.5%	1.357	4.830
12	250	8.49	246	29%	1.5%	2.678	4.449

As Table 3 presents, the prioritization mechanism seems to work efficiently. On each testing scenario the packet loss that the foreground experienced 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 (and in some cases, also the packet loss demonstrates strange behaviour). This point is actually due to the fact that the foreground traffic follows a longer path (Figure 2) and the measurement is end to end and not for the common path (between the 2 routers) for foreground and background traffic. We have measured the average jitter for packets that have been generated from the same source as the foreground traffic, until the PC that inserts the background traffic (this is the additional part of the path) and the jitter is almost 4.3ms. Taking this result into account, the jitter that the foreground traffic experiences on the real testbed (between the routers) 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 3 and Figure 4 present the measurements for packet loss and jitter for the foreground and background traffic.

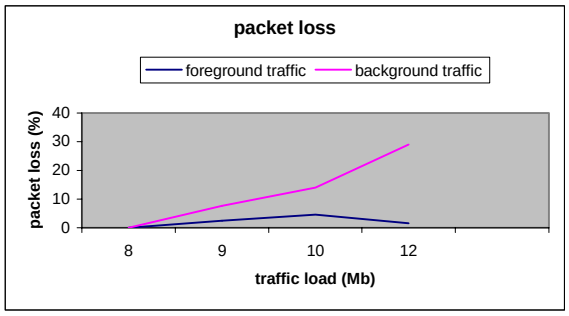

Figure 3. Packet loss for different traffic load

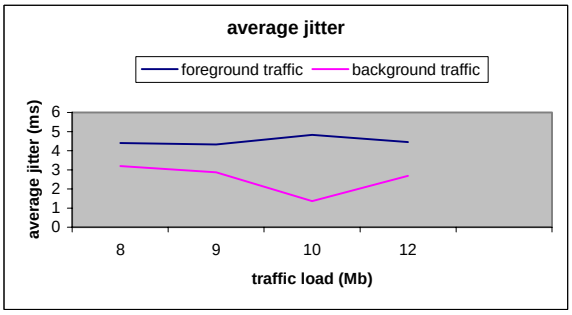

Figure 4. Average jitter for different traffic load

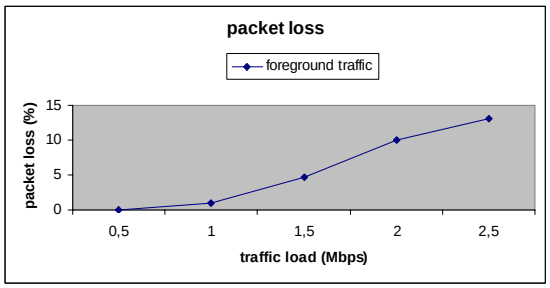

Figure 5. Strict priority's behavior on upper bound

Afterwards, some extra experiments were performed with the goal to investigate the behavior of the LLQ mechanisms in conditions where the percentage of bandwidth that foreground traffic 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 2Mbps. For this reason, we also performed the experiments to record its behavior when it approaches its upper bound. The results are presented in Figure 5. According to the results, we can see that the packet loss increases proportionally for traffic load equal or larger than 1Mbps. This figure also presents one more strange result, as theoretically the Class based Weighted Fair Queuing mechanism was supposed to perform strict policy and discard all packets when the rate exceeded the upper bound (in our case 2Mbps). Instead, Figure 5 shows that packets are still transmitted, but with bigger packet loss.

4.2 Experimental testing for real time applications

Finishing the above-described experimental stages, the QoS mechanisms that can provide QoS guarantees have been set up and evaluated. As we described earlier, the mechanism used is the strict priority (which implements the Low Latency Queues) and the reason why we used and investigated the behaviour of this mechanism is because it is extremely suitable for real-time applications that need low delay, packet loss and jitter. Therefore, we tried to simulate realistic conditions of traffic load and to measure the performance of real-time applications that experience the preferential treatment by those mechanisms. For implementing the following testing scenarios we used an application based on the OpenH323 library.

4.2.1 Testing scenario 1

Initially, the network was loaded 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 note that we have started loading the network before inserting the foreground traffic, in order for the TCP to obtain a stable state. Next, the foreground traffic that was generated by a videoconference (using the OpenPhone application based on the OpenH323 library) was inserted.

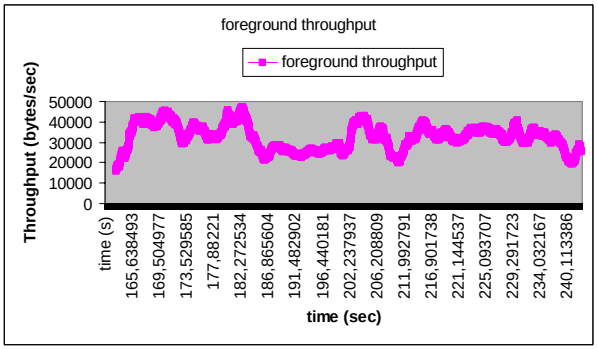

Figure 6. Throughput of foreground traffic

This test was performed for more than 5 minutes and we recorded the packets that were exchanged. The background traffic was 5Mbps UDP traffic and also TCP traffic that tried to occupy as much bandwidth as 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 straggled by the strict priority mechanism. Figure 6 and Figure 7 present the throughput of the foreground traffic as well as the throughput of the 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.

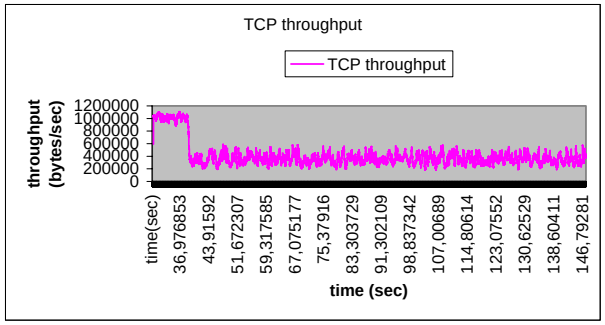

Figure 7. Throughput of TCP background traffic

4.2.2 Testing scenario 2

Similarly, a second test with the same characteristics and traffic load was performed. 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, as 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 8 and Figure 9 present the throughput of the foreground traffic and TCP background traffic.

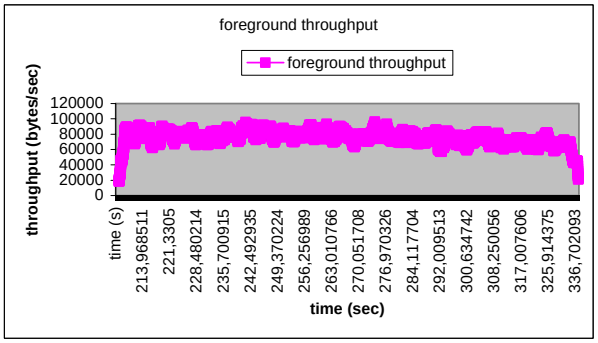

Figure 8. Foreground throughput

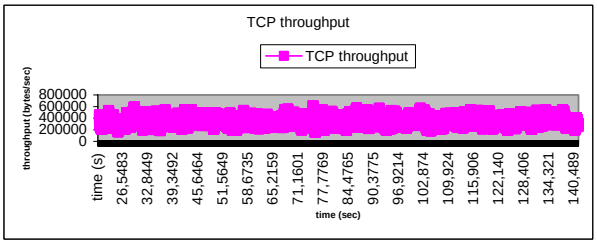

Figure 9. TCP background throughput

4.3 Investigation of WRED mechanism

Our next goal was 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. Our goal 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 2 separate testing scenarios that are described in the following sections were performed.

4.3.1 Testing scenario 1

Initially the WRED mechanism was set up in order to be applied on the background traffic using 30 and 50 packets in the queue as thresholds. In addition, the maximum queue size was 75 and the drop possibility was 10%. A test that we had also performed earlier was repeated with this new configuration, 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). The result was that the foreground traffic still only had 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 (UDP background traffic had almost 2% packet loss). So, 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 look at the TCP throughput of the background traffic (Figure 10) and compare it with the corresponding throughput 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. This can be explained if we consider that the WRED mechanism "created" packet losses earlier, so TCP thought that the network was congested and reduced its rate.

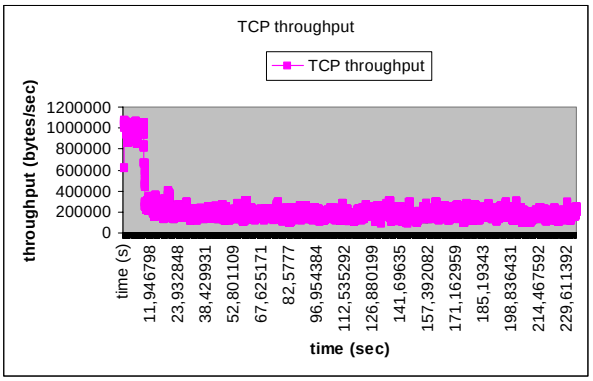

Figure 10. TCP throughput when WRED has been applied

4.3.2 Testing scenario 2

After the first experiment, the same scenario was repeated, but this time we changed the thresholds of the WRED mechanism. We tried to approach the max queue size and configured the min and max thresholds to be 55 and 75 packets respectively. The drop possibility was also 10%.

We observed similar results regarding the foreground traffic, as the packet losses were almost zero and the video quality was very good. This time the background traffic had better behaviour, as only 0.92% of UDP traffic packets were lost. In addition, the TCP traffic had a bigger average throughput (1.36 Mbps). So, at this experiment the queues were allowed to be more filled and achieved a better performance for the background traffic. But, regarding the foreground traffic (for the QoS service that is tested) the existence of the WRED mechanism does not have any significant impact.

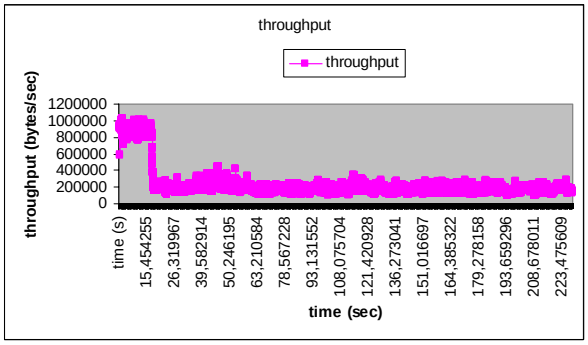

Figure 11. TCP throughput with existence of WRED

5 CONCLUSIONS

In this paper, we presented a QoS service that we implemented on an IPv6 network and examined its correctness. The QoS mechanisms that were used were tested widely in order to make sure that they work well and additionally to investigate their performance. The QoS service was tested using simulated traffic (but as close to reality as possible) providing QoS to traffic that belongs to real-time applications. The main feature that was tested was the MQC (Class based Weighted Fair Queuing) and specifically the low latency queue feature that it is implemented with the priority command. The operation of this feature is to "implement" a queue that provides very low latency for traffic that belongs to the specified class. So it is obvious that theoretically this feature is ideal for real-time traffic that needs high priority. Finally, seeing all the above experiments and their conclusions, the QoS service that we tried to implement and test, using the above mechanisms, worked efficiently as it reduced the packet loss, the delay and the jitter of the real-time data, consequently increasing the receiving quality for the applications that produced these data.

Additionally, some tests focused on the WRED mechanism that had been applied on the background traffic and aimed to investigate its effect on the QoS service. The results from those experiments indicate that there was not any significant impact on the foreground traffic from the existence of the WRED. On the other hand, the background traffic has been affected from the WRED mechanism and the impact is proportional to the values that the thresholds of the WRED mechanism had been configured at.

Finally, all the tests that were performed and are described on this paper have been repeated at least twice, with exactly the same characteristics, and the results were always the same. The overall conclusion is that the QoS service, with the use of the specific mechanisms that were tested, can provide prioritization on an IPv6 domain to the specified traffic and therefore the real-time application (that produces the traffic) can operate efficiently and with high quality.

6 FUTURE WORK

All the tests that were performed also indicated some points that need further research and investigation. Therefore, our future plans include extending these experiments on bigger network topologies. We also plan to test some other mechanisms like policing at the ingress of the DiffServ domain. Generally, the policing mechanism is implemented using the token bucket algorithm and its role 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). The policing mechanism is very crucial in order to provide QoS service to a domain, as its operation is closely related to the "implementation" of the SLAs [20]. The most interesting and open issue for research is the investigation of the way that the policy profile should be selected and configured for aggregate of flows of real time data in order to contract and follow the SLAs. Finally, we are also interested on testing additional QoS features that will possibly be

introduced in later versions of CISCO IOS for IPv6 QoS or on different router platforms.

7 REFERENCES

[1] C Bouras, M. Campanella, M. Przybylski and A. Sevasti, 2003, "QoS and SLA Aspects Across Multiple Management Domains: The SEQUIN Approach", Future Generation Computer Systems 19 pp. 313-326

[2] J Kielthy, R. Frisby and M O Foghlu, 2003, "An Initial Investigation into QoS Provisioning in a DiffServ Domain" Telecommunication System Software Group

[3] C. Bouras, D. Primpas, A. Sevasti and A. Varnavas, 2002, "Enhancing the DiffServ Architecture of a Simulation Environment", 6th IEEE International Workshop on Distributed Simulation and Real Time Applications, Fort Worth, Texas, USA, October 11–13

[4] S. Vegesna, "*IP Quality of Service: the Complete Resource for Understanding and Deploying IP Quality of Service for Cisco Networks*", Cisco Press, 2001

[5] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification" RFC 2460, December 1998

[6] Integrated Services (intserv) Working Group, Internet Engineering Task Force (IETF), http://www.ietf.org/html.charters/intserv-charter.html

[7] Differentiated Services (diffserv) Working Group, Internet Engineering Task Force (IETF),http://www.ietf.org/html.charters/OLD/diffserv-charter.html

[8] K. Nichols and B. Carpenter, RFC 3086, "Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification", April 2001

[9] V. Jacobson, K. Nicholsm and K.Poduri, "An Expedited Forwarding PHB", RFC 2598, June 1999

[10] J. Heinanen, F Baker, W. Weiss and J Wroclawski "Assured Forwarding PHB Group", RFC 2597, June 1999

[11] J. Rajahalme, A. Conta, B. Carpenter and S. Deering, RFC 3697, "IPv6 Flow Label Specification", March 2004

[12] Cisco Systems, Inc. home page, http://www.cisco.com

[13] 6NET Project homepage, http://www.6net.org

[14] Iperf homepage, http://dast.nlanr.net/Projects/Iperf/

[15] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996

[16] The OpenH323 project, http://www.openh323.org and http://sourceforge.net/projects/openh323

[17] S. Josset, C. Bouras, A. Gkamas and K. Stamos, 2003, "Adding IPv6 Support to H323: Gnomemeeting/OpenH323 Port", 11th International Conference on Software Telecommunications and Computer Networks (SoftCOM 2003), Croatia, Italy, October 7-10, pp. 458-462

[18] C. Bouras, A. Gkamas, D. Primpas and K. Stamos, 2004, "Performance Evaluation of an IPv6–capable H323 Application" The 18th International Conference on Advanced Networking and Applications (AINA 2004), Fukuoka, Japan, March 29-31, pp. 470-475

[19] Ethereal homepage, http://www.ethereal.com

[20] George Fankhauser, David Schweikert and Bernhard Plattner, 1999, "Service Level Agreement Trading for the Differentiated Services Architecture", Tech. Rep. 59, TIK

Christos Bouras obtained his Diploma and PhD from the Computer Science and Engineering Department of Patras University (Greece). He is currently an Associate Professor in the above department. Also he is a scientific advisor of Research Unit 6 in Research Academic Computer Technology Institute (CTI), Patras, Greece. His research interests include Analysis of Performance of Networking and Computer Systems, Computer Networks and Protocols, Telematics and New Services, QoS and Pricing for Networks and Services, e-Learning Networked Virtual Environments and WWW Issues.

Apostolos Gkamas obtained his Diploma, Master Degree and PhD from the Computer Engineering and Informatics Department of Patras University (Greece). He is currently an R&D Computer Engineer at the Research Unit 6 of the Computer Technology Institute, Patras, Greece. His research interests include Computer Networks, Telematics, Distributed Systems, Multimedia and Hypermedia.

Dimitris Primpas obtained his Diploma and Master Degree from the Computer Engineering and Informatics Department of Patras University (Greece). He works in the Research Unit 6 of Research Academic Computer Technology Institute (CTI). His research interests include Computer Networks, Telematics, Distributed Systems and Quality of Service.

Kostas Stamos obtained his Diploma and Master Degree from the Computer Engineering and Informatics Department of Patras University. He has worked for the Networking Technologies Sector of Research Academic Computer Technology Institute (CTI) , Patras, Greece from the end of 1999 until December 2000. Since July 2001 he works with Research Unit 6 of CTI.