Journal of Network and Systems Management, Vol. 12, No. 4, December 2004 (© 2004)
DOI: 10.1007/s10922-004-0672-5

Performance Evaluation of the Impact of QoS Mechanisms in an IPv6 Network for IPv6-Capable Real-Time Applications

Ch. Bouras,[1,2,3] A. Gkamas,[1,2] D. Primpas,[1,2] and K. Stamos[1,2]

This paper describes a Quality of Service (QoS) service on an IPv6 domain that aims to service aggregates of real-time traffic with minimum delay, jitter, and packet loss. It contains results from the tests that were performed in order to configure and evaluate the QoS mechanisms. As an actual example of real-time traffic, we have used the OpenH323 project, an open source H.323 implementation that has been ported to IPv6. The QoS mechanisms in IPv6 networks is still a field that has not been researched adequately, and we therefore present the 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.

KEY WORDS: Quality of Service; IPv6; OpenH323, real-time applications.

1. INTRODUCTION

The new version of the Internet Protocol (IP), IPv6 [1], is a solution designed to overcome the limitations of IPv4 and fill the future needs in the Internet. While increasing the address space and solving a major obstacle for the further growth of the Internet, IPv6 also offers improvements like security support embedded in the protocol's definition, address autoconfiguration, no checksum at the IP header, and more flexibility and extensibility than IPv4. The widespread adoption of the new Internet Protocol will fuel innovation and make possible the creation of many new networking applications. It will also allow the replacement of the Network Address Translation (NAT) solutions that have been implemented today in order

[1]Research Academic Computer Technology Institute, Riga Feraiou 61, Patras, Greece.
[2]Computer Engineering and Informatics Department, University of Patras, GR-26500 Patras, Greece.
[3]To whom correspondence should be addressed at Research Academic Computer Technology Institute, Riga Feraiou 61, GR-26221 Patras, Greece. E-mail: bouras@cti.gr

to workaround the lack of IPv4 addresses. NAT introduces a number of problems to network applications that need knowledge of the IP address of the host machine or want to take advantage of Quality of Service (QoS) mechanisms, like Voice over IP (VoIP) implementations.

On the other hand, Internet traffic consists of flows generated by different applications, which all receive the same treatment from the network, as the network can only provide best-effort service. This treatment causes many problems to real-time applications, because they are sensitive on parameters such as delay, packet loss, or jitter. The implementation of QoS techniques is therefore necessary and their usage is widely experimented. The QoS guarantees can be measured, using a number of specific metrics that include the bandwidth that a traffic class uses and the delay that the packets of each class experience. In addition, packet loss and jitter (the fluctuation of the arrival time between packets) are the other two metrics that are used to provide QoS guarantees. During the last years, two architectures have been proposed in order to provide QoS and some services have already been deployed. Until now, significant research activity has been done on QoS in IPv4 networks [2], as there are several publications, but QoS in IPv6 networks is still a relatively open issue. 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. Since the usage of the IPv6 protocol [1] increases and many domains have been IPv6 enabled or are expected to become so in the next few years, the investigation of the supported QoS mechanisms on IPv6 and the deployment of a QoS service should be a basic goal for existing networks as they migrate to tomorrow's IPv6 networks.

To evaluate the QoS mechanisms, we used as real-time traffic the traffic generated by an H.323 videoconferencing application based on the library developed by the OpenH323 project. The OpenH323 project [3] develops an open source implementation of the H.323 standard in the form of a central library, the OpenH323 library, which is also based on another open source library called PWLib. The OpenH323 library can be used for the rapid development of applications that wish to use the H.323 protocol for multimedia communications over packet-based networks. The library and most of the applications it supports have now been ported to IPv6 [4, 5].

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 Differentiated Services (DiffServ) architecture. Section 3 describes the IPv6 testbed that we use and explains the QoS techniques and traffic patterns that have been used for the experiments. Section 4 presents the experiments with best-effort service and with the QoS mechanisms, as well as the results from each one. Finally, Section 5 describes our conclusions from the experiments and the future work that we intend to do in this area.

2. THE QUALITY OF SERVICE TECHNIQUES

The two main architectures for Quality of Service that have been proposed are Integrated Services (IntServ) [6] and Differentiated Services (DiffServ) [7, 8]. They follow different philosophies 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] which has a complicated operation and also inserts significant network overhead. In particular, all the messages that are exchanged in order to initialize, terminate, or update the status of a resource reservation, create a network overhead and additionally a computational overhead to the nodes of the network.

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 every class needs. In the DiffServ architecture, two different types have been proposed (per hop behaviors [9]), the expedited forwarding (EF) [10], and the assured forwarding (AF) [11]. Their difference is the packet-forwarding behavior. The former, expedited forwarding, aims to provide QoS on a traffic class by minimizing the jitter and generally aims to provide more strict guarantees. This type of DiffServ architecture tries to simulate the virtual leased lines and the policy behavior should therefore be very tight. The latter, assured forwarding, inserts at most four classes with at most three levels of dropping packets. Every time the traffic of each class exceeds the policy criteria then it is marked as a 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). To do this it can use several criteria, such as the IP addresses of source and destination, port numbers in source and destination, or special protocol characteristics. In our case, the classification is done using the DSCP field (Differentiated Service CodePoint [7]). This field exists in IPv4 and also in IPv6 packet headers. In IPv4 it is part of the Type of Service field (TOS) while in IPv6 [1] that our work focuses on, it is part of the Traffic Class field. The Traffic Class field has an 8-bits length and declares the service that is provided on the network. In addition, in the IPv6 packet header, there is the Flow label field (20 bits) that declares the traffic flow that the packets belong to. This field is however still experimental and its use has only been recently standardized [9].

The operation of services based on the DiffServ architecture uses several mechanisms that act on every aggregate of flows. In particular, these mechanisms are packet classification, packet marking, metering, and shaping. In addition, in

order to provide QoS guarantees it is necessary to properly configure the queue management and time routing/scheduling mechanism. The packet classification and marking mechanisms classify the packets in aggregates of traffic. This is achieved by marking the packets using the DSCP field in the IPv6 packet header. Next, the queue management mechanism is configured in order to provide the preferential packet treatment for the appropriate traffic classes. Then the DiffServ architecture makes use of the marking and metering mechanisms. These two mechanisms operate together by using token bucket algorithms. In particular, they try to measure the profile of the traffic classes and compare it with the preagreed levels. This way they make the decision of how to classify the packets based on whether they exceed the policy profile or not. The token bucket algorithm is believed to be the most suitable algorithm for this operation and the network administrator should choose the appropriate parameters for the token bucket for the service he wants to provide. The last step is the shaping/dropping mechanism that is used in conjunction with the marking/metering. This mechanism acts when a percentage of the traffic class exceeds the policy profile and therefore the shaping/dropping mechanisms are applied for the exceeded packets. The action that will take place is the one specified by the network administrator, and it could either be the drop of all the "exceeded" packets or alternatively, the shaping mechanism could be used to normalize the traffic class, hoping that it will then agree with the policy profile. The shaping mechanism is preferably used when the traffic class contains significant bursts that cause the exceeding packets from the policy profile. The most known shaping mechanism is the Leaky Bucket algorithm. Generally all the above mechanisms are called traffic conditioning mechanisms.

The DiffServ architecture should only specify the forwarding mechanisms for each traffic class, in order for the traffic class to have the preferential treatment. This forwarding mechanism is closely related to the queue management mechanisms that are applied on the network. The most common approaches are the Priority Queue, Weighted Fair Queue, or Modified Deficit Round Robin [11, 12]. The first one, Priority Queue, simply creates a priority queue and all the packets that are inserted into it have strict forwarding priority on the network. The second one is the Weighted Fair Queue, which is a mechanism that creates several queues giving a specific weight to each one. According to this weight, the network administrator can provide several services with different levels of delay and jitter. The Modified Deficit Round Robin [12] is the most modern mechanism and has two versions. The first one creates a priority queue, with specific thresholds, and when this queue is empty it forwards packets from the other queue with the Round Robin approach. The second one creates a semi-priority queue and forwards packets sequentially from the priority queue and from one of the other queues.

The main problem that has to be taken into account is that not all of the above-described mechanisms have been implemented to operate on IPv6 domains yet.

In particular, many vendors only provide Weighted Fair Queue or Priority Queue and this makes it more difficult to provide efficient QoS services. But generally, this fact can be considered temporary and it is expected that most vendors will shortly implement more mechanisms in their products as the usage of the IPv6 protocol increases.

3. THE IMPLEMENTED QoS SERVICE ON IPv6 TESTBED

The QoS services are extremely useful for real-time applications, where the packet loss or the delay reduces their quality. In addition, on next generation networks where the number of users is expected to be significantly larger, the need for QoS services will increase. Here, we describe the implementation and testing of a QoS service that will be applied on IPv6 networks and will service flows from real-time applications. 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 testbed IPv6 network that has been created internally in our organization, the Computer Technology

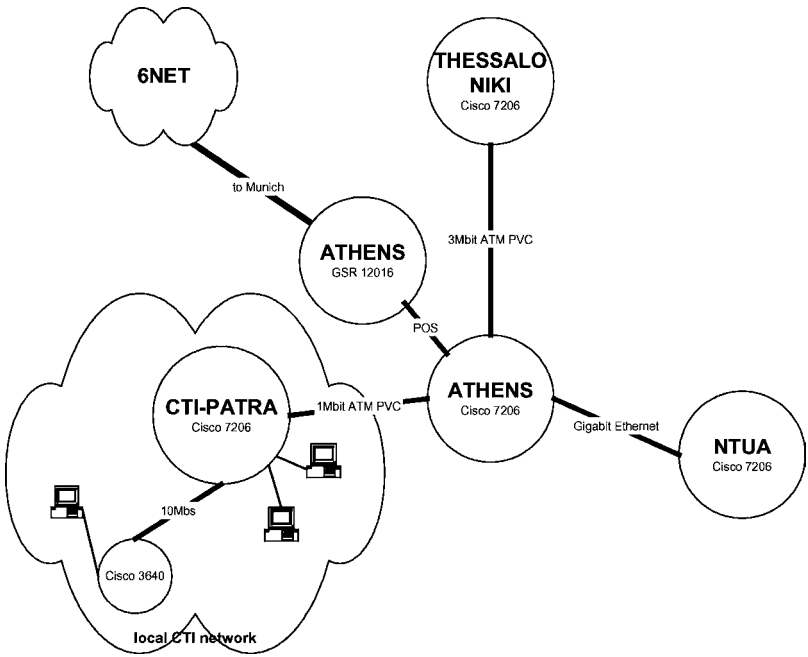

Fig. 1. The greek part of the 6NET network.

Institute (CTI) and is presented in Fig. 1. The software version that this testbed uses for the operating system of the routers is the CISCO IOS 12.2(13) T [12].

The QoS service that was implemented aims to provide prioritization to traffic coming from real-time applications. The QoS service first has to classify the packets that belong to the real-time applications. Next, those packets experience preferential treatment (forwarding with strict priority) with the use of a "priority queue." The rest of the traffic on the router will be treated with best-effort service. So, the rest of this paper focuses on presenting the implemented service, evaluating its performance as well as investigating the correct operation of the prioritization mechanism.

The service has been implemented using the Class-Based Weighted Fair Queuing mechanism (CBWFQ) [11]. This mechanism is software based, extends the classic Weighted Fair Queuing (WFQ) mechanism and can provide strict packet priority. The strict priority feature implements the Low Latency Queue (LLQ) mechanism, which is a method to enqueue packets on a "priority" queue in order to guarantee them low latency and jitter. The service follows the classic guidelines of the DiffServ architecture, so the packets are classified at the ingress point of the IPv6 domain. The experiments have been done using the 7206 and 3610 Cisco routers, with various PCs connected to them using two network switches and hubs (Fig. 1 presents the Greek part of 6NET's network, a research IPv6 network connecting academic and commercial partners around Europe, and Fig. 3 presents CTI's internal testbed in more detail). Our goal was to make the experimental scenarios as realistic as possible. So, we inserted background traffic in the network with the cross-connect method that is a mix of TCP and UDP traffic generated with the Iperf [13] 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 UDP traffic (generated with a traffic generator) and RTP (Real-Time Transport Protocol) traffic [14] (generated by an IPv6-enabled OpenH323 application [3]).

To mark the background and foreground traffic, we have applied a marking mechanism on the network devices. 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 [10, 11]. Then, the output interfaces of the network devices have been configured so that they send the packets that have been marked with DSCP 101110 (Expedited Forwarding) with strict priority.

The experimental procedure that has been followed contains several stages. Initially we evaluate the IPv6 version of the OpenH323 library, and particularly in comparison with the IPv4 version using best-effort service. In Ref. 15, we document how both versions behave when a link in the transmission path is congested because of the simultaneous transmission of other UDP and TCP traffic. Here we document our next step, which was to perform many tests in order to investigate and report the operation of the QoS mechanism under different

conditions (traffic load, congestion etc.). As we finished the investigation, we tried to test this QoS service with traffic generated by real-time applications (OpenH323) and report the results.

After this experimental procedure, we implemented a second testing scenario, where the network devices had also been configured in order to apply Weighted Random Early Detection (WRED) mechanism [11] for congestion avoidance on the background traffic. In this case the goal is to measure whether the existence of WRED has any impact on the QoS guarantees for the foreground packets. The operation of the WRED mechanism is quite straightforward as the administrator simply specifies the minimum and maximum threshold for the queue size and the packet's dropping probability when the queue size is between the thresholds. If the queue size is smaller than the minimum threshold then the packets are forwarded without any impact. If the queue size is between the minimum and maximum threshold, then incoming packets are dropped with probability equal to the dropping probability. Finally, if the queue size exceeds the maximum threshold, then all the packets are dropped.

To create background traffic, we used the Iperf tool [13], which is capable of producing TCP and UDP traffic in both IPv4 and IPv6 packets. For retrieving and studying the transmission/reception data we used both the RTP/RTCP (Real-Time Transport Protocol/RTP Control Protocol) feedback from the OpenH323 library, and the Ethereal network monitoring tool [16].

Transmission of video data was made using the OpenH323 built-in H.261 codec with CIF (Common Intermediate Format) resolution. Audiotransmission was achieved using G.711 (muLaw variation), a PCM (Pulse Code Modulation) scheme that operates at the rate of 64 Kbps. The main application used for the tests was the OpenPhone GUI client.

4. EVALUATING THE DiffServ MECHANISM

4.1. Best-Effort Traffic

Our first set of experiments, which are described in detail in Ref. 4, aimed at observing and understanding the behavior of real-time traffic when no QoS mechanism is applied, which means that real-time traffic is treated as best effort. Our first experiment in this set was to create real-time traffic with an OpenH323 videoconferencing application on an uncongested network, in order to compare the bandwidth requirements for the IPv4 and the IPv6 versions of the application. The result was that IPv6 has a slightly bigger need for bandwidth, because of its larger header (on the average case where IPv4 uses no additional header fields). In particular, the IPv6 transmission rate was around 7% higher than the IPv4 transmission rate under similar circumstances, as can be seen in Fig. 2.

Higher bandwidth requirements lead to more immediate needs for quality guarantees, in order to avoid reduced application quality due to network

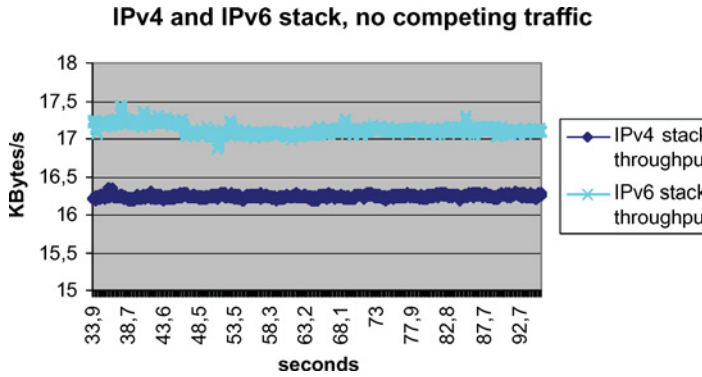

Fig. 2. Real-time OpenH323-based application bandwidth consumption.

congestion. Therefore, our next step was to simulate network congestion by introducing background traffic. We experimented both with UDP and TCP background traffic for the IPv4 and IPv6 versions of OpenH323. Our experiments showed that while background traffic was causing congestion problems at the network, the transmission rate and the quality of the real-time application was significantly reduced. Other kinds of traffic, especially TCP traffic that includes congestion avoidance mechanisms with the AIMD (Additive Increase, Multiplicative Decrease) algorithm tend to suffer from congestion also. These results clearly demonstrated the need for QoS mechanisms for real-time applications that compete for limited network resources with similar or different kinds of traffic, both in IPv4 and in IPv6 networks. This conclusion led us to the second set of experiments, this time taking advantage of DiffServ mechanisms for QoS guarantees.

4.2. Investigation of the Prioritization Mechanism

For the evaluation of the DiffServ mechanisms, we inserted foreground and background traffic with cross-connect method on the network. The background traffic is a mix of TCP and UDP traffic and is created with Iperf traffic generator [13]. The foreground traffic is a mix of UDP traffic and real-time traffic (RTP).

The first point was to investigate the operation of the classification and prioritization mechanisms in our testbed (Fig. 3). 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 101110 (EF). To make sure that

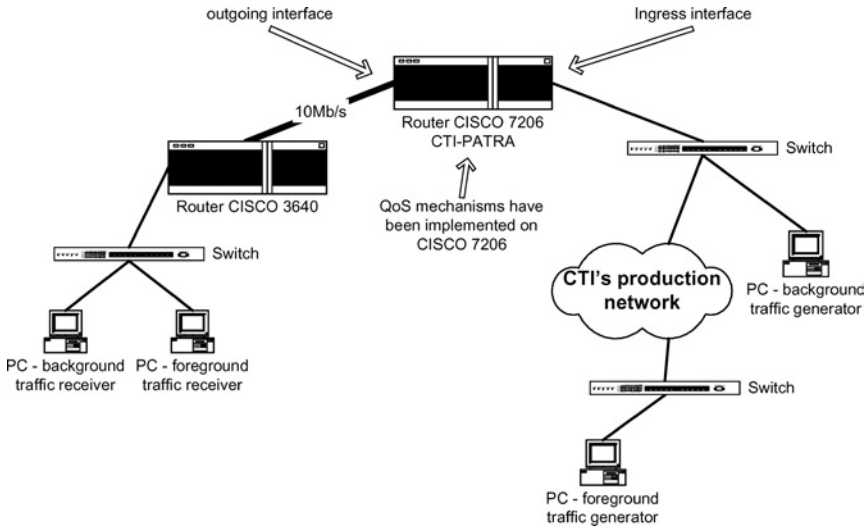

Fig. 3. CTI's testbed.

the configured mechanisms operate as expected, a number of tests were done and their results are presented in this section.

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 was UDP with average rates of 12 and 1.5 Mbps, respectively. The backbone link is 10 Mbps 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 behavior when it only has best effort traffic (UDP or both UDP and TCP). At this stage, we performed several tests that are summarized in Tables I and II. First, we performed tests using only UDP traffic, generated by the Iperf traffic generator. Table I presents the experimental results.

Table I. Best-Effort Experiments (Only UDP)

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

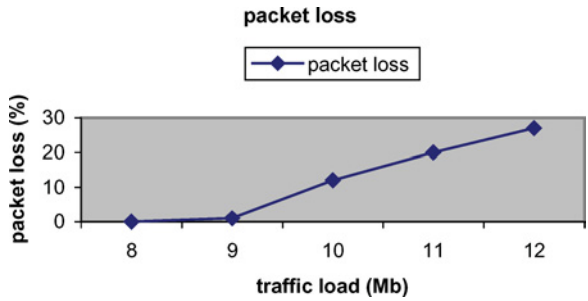

Fig. 4. The packet loss when network usage increases.

As we can see from the results, the actual throughput that can fill the backbone links is 8.80 Mbps. In addition, as the network usage increases, the packet loss and jitter also increase as Figs. 4 and 5 show.

Next, we repeated the same scenarios using both TCP and UDP traffic generated by the Iperf traffic generator. Table II 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, these experiments run for 3 min, so the results for the TCP can be considered quite stable and accurate.

According to the above results, we concluded that for almost 8 Mbps 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 seemed capable of meeting our traffic generation requirements. It produced the actual UDP traffic that consumes its specified percentage of the bandwidth

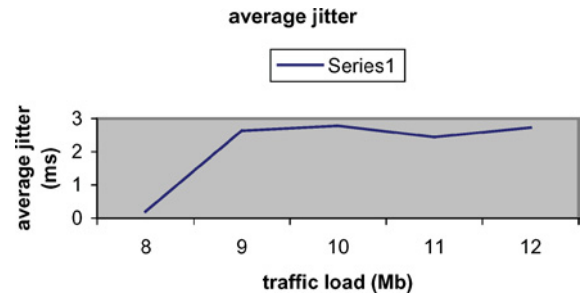

Fig. 5. The average jitter when network usage increases.

Table II. Best-Effort Experiments (UDP and TCP)

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

(because of the protocol's nature that does not include a congestion avoidance mechanism) 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 Table III.

As can be seen in Table III, the prioritization mechanism seems to work efficiently. On each testing scenario the packet loss that the foreground traffic experienced was significantly low, in contrast with the background traffic that suffered significantly larger packet losses. Regarding the jitter, Table III shows that the foreground traffic has larger average jitter (and in some cases, packet loss also has strange behavior). This point is actually due to the fact that the foreground traffic follows a longer path as it also crosses a part of CTI's internal network before reaching the testbed. 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 it is almost 4.3 ms. 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 Low Latency Queue (LLQ) mechanism. This mechanism uses low latency queues for the classified packets that provide low delay and we therefore expect significantly lower jitter. Figures 6 and 7 present the packet loss and jitter metrics for the foreground and background traffic for the above scenarios.

Afterwards, some extra experiments were performed aiming to investigate the behavior of the Low Latency Queue (LLQ) mechanism in conditions where the foreground traffic exceeds the percentage of bandwidth that has been allocated for it. In particular, as we had configured the policy map, the strict priority could only use 20% of the total bandwidth, in other words only 2 Mbits. For this reason, we also performed the experiments to record Low Latency mechanism's behavior when it approaches its upper bound.

Looking at the results from the above experiments (Table IV), we conclude that when the rate approaches or exceeds the upper bound that the strict priority mechanism can guarantee, then its behavior seems to change (for faster rates of

Table III. Testing of Classification and Prioritization Mechanism

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

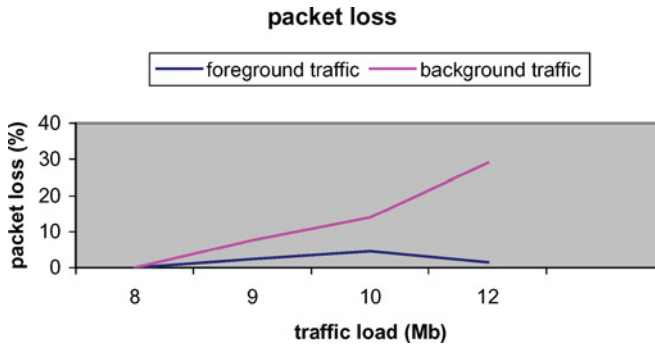

Fig. 6. The packet loss for various background traffic loads (foreground = 250 Kbps).

foreground traffic than 1 Mbps, the packet losses increase). This can be better demonstrated at Fig. 8.

According to Fig. 8 the packet loss increases proportionally for traffic load equal or larger than 1 Mbps. 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 extends the upper bound (in our case 2 Mbps). Instead, Fig. 8 shows that packets are still transmitted, but with bigger packet loss.

4.3. 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 was presented earlier, the used mechanism is the strict priority (which implements the Low

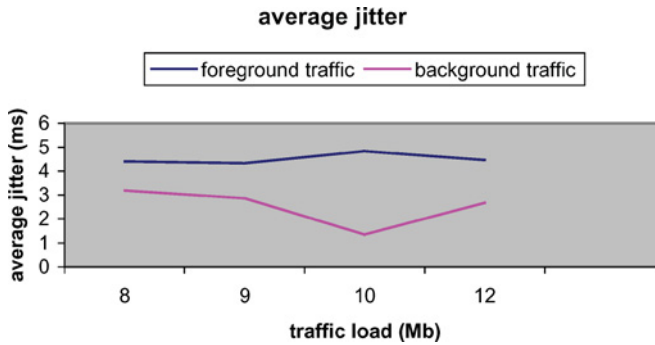

Fig. 7. The average jitter for various background traffic loads (foreground = 250 Kbps).

Table IV. Experiments for Testing the Upper Bound of Strict Priority

Traffic load (bps)		Actual throughput (bps)		Packet loss		Average Jitter	
Background traffic	Foreground traffic	Background traffic	Foreground traffic	Background traffic	Foreground traffic	Background traffic	Foreground traffic
5 M	1.5 M	4.99 M	1.5 M	0.0037	0.047	0.007 ms	3.376 ms
5 M	2.5 M	4.89 M	2.5 M	2.2%	0.094%	0.743 ms	2.276 ms
6 M	4 M	4.24 M	3.16 M	29%	21%	5.779 ms	1.773 ms
8 M	0.5 M	7.98 M	500 K	0.043%	0.047%	3.018 ms	3.761 ms
8 M	1 M	7.72 M	990 K	3.3%	1%	3.092 ms	4.641 ms
8 M	1.5 M	6.93 M	1.43 M	13%	4.6%	4.630 ms	3.005 ms
8 M	2 M	6.16 M	1.8 M	23%	10%	2.989 ms	3.346 ms
8 M	2.5 M	5.3 M	2.16 M	34%	13%	2.390 ms	4.500 ms

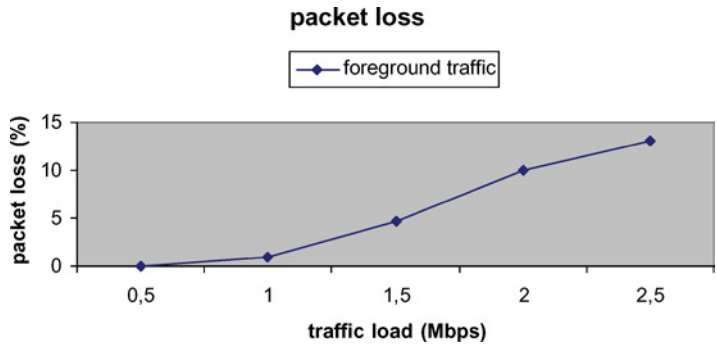

Fig. 8. Strict priority's behavior on upper bound.

Latency Queues) and the reason why we used and investigated the behavior of this mechanism is because it is extremely suitable for real-time applications that need low delay, packet loss, and jitter.

The next step was to simulate realistic conditions of traffic load and measure the performance of real-time applications that are preferentially treated by those mechanisms. For this reason, we used an actual application based on the OpenH323 library for implementing the following testing scenarios.

4.3.1. Scenario 1: Real-Time Foreground Traffic

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 long enough 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. This test was performed for more than 5 min and we recorded the packets that were exchanged. The background traffic consisted of 5 Mbps UDP traffic and TCP traffic that tried to occupy as much bandwidth as possible, as specified by the TCP implementation. The results demonstrated 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. On the other hand, the TCP background traffic was strangled by the strict priority mechanism. Figures 9 and 10 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 s it obtains its steady state. In addition, the foreground traffic has an average throughput of almost 300 Kbps and very good video quality.

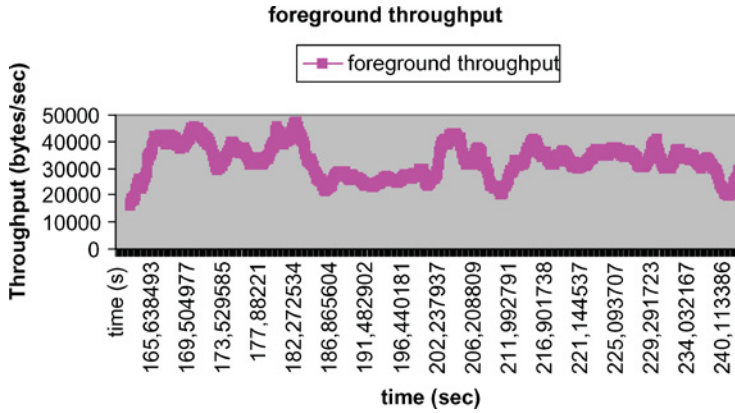

Fig. 9. Throughput of foreground traffic.

4.3.2. Scenario 2: Foreground Traffic Is a Mix of Real-Time and UDP-Generated Traffic

Afterwards, 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 (300 Kbps), once again 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). Figures 11 and 12 present the throughput of the foreground traffic and TCP background traffic.

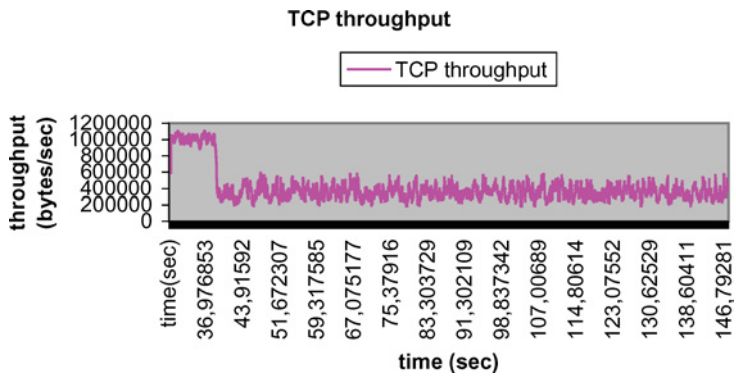

Fig. 10. Throughput of TCP background traffic.

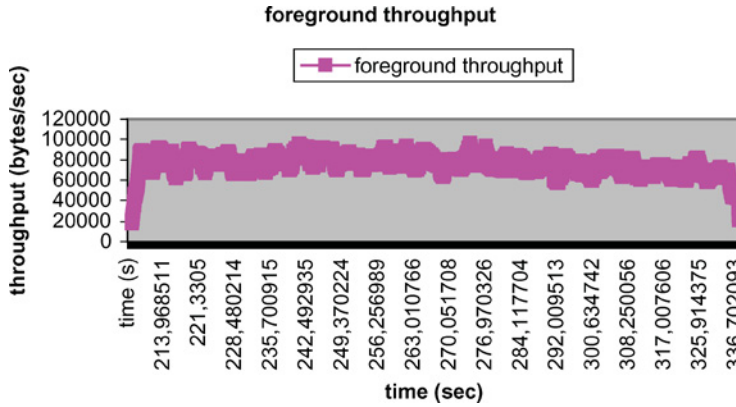

Fig. 11. Foreground throughput.

4.4. Investigation of WRED Mechanism

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

4.4.1. Scenario 1: Initial WRED Experiment

The WRED mechanism was set up 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 probability was 10%. According to this

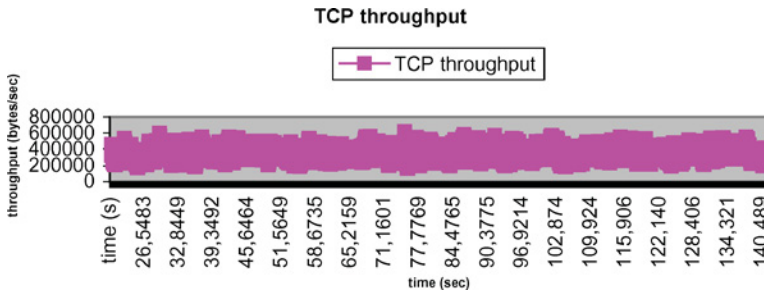

Fig. 12. TCP background throughput.

Table V. Results Testing the WRED Mechanism

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

configuration, we repeated a test that had also been performed earlier, in order to compare the results. So, there were background traffic that was a mix of TCP and UDP (5 Mbps) and foreground traffic that was a mix of UDP (700 Kbps) and RTP (OpenH323-based application). Finally, 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. The results are displayed in Table V and the TCP throughput is shown in Fig. 13.

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 look at the TCP throughput of the background traffic (Fig. 13) and compare it with the corresponding throughput on the previous section (Figs. 10 and 12). It is obvious that the throughput in this case is lower and that the WRED caused

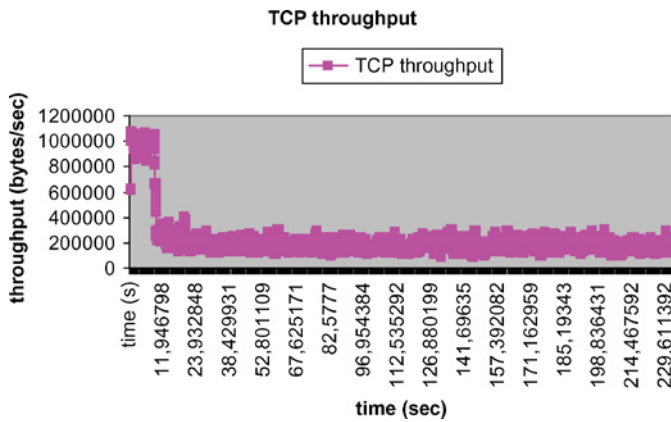

Fig. 13. TCP throughput when WRED has been applied.

this reduction of TCP transmission 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.

4.4.2. Scenario 2: Higher WRED Thresholds

After the first experiment, the same scenario was repeated except that we changed the thresholds of the WRED mechanism. This time we tried to approach the maximum queue size and configured the thresholds to be 55 and 75 packets, respectively. The drop probability 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 behavior as only 0.92% of UDP traffic packets were lost. In addition, the TCP traffic had a bigger average throughput (1.36 Mbps), as can be seen in Fig. 14. So, at this experiment we allowed the queues to be more filled and achieved a better performance for the background traffic. But, regarding the foreground traffic (for the QoS service that was tested) the existence of the WRED mechanism does not have any significant impact.

5. CONCLUSIONS AND FUTURE WORK

This paper presented a QoS service that was implemented on an IPv6 network, aiming to serve real-time traffic. The QoS mechanisms that were used were tested widely in order to make sure that they work as expected and additionally in order to investigate their performance. Finally, this QoS service was tested using simulated traffic (but generated in such a way that it would be as close to reality as possible) and providing QoS to traffic flows that belong to real-time applications.

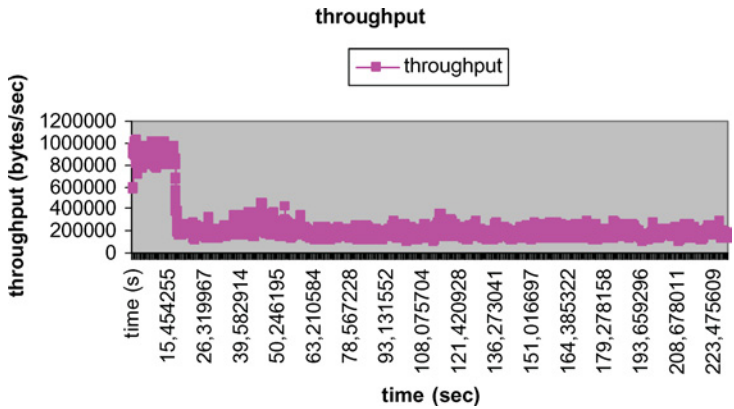

Fig. 14. TCP throughput with existence of WRED.

The real-time application under investigation was the OpenH323 library and the applications that have been developed on it. The results generally show that the level of QoS that the real-time traffic experiences is very good and gives to the real-time applications the ability to operate with high quality standards.

In addition, a second experimental phase was performed, applying the WRED mechanism for congestion avoidance on background traffic. The goal was to investigate its effect on the QoS service. The results from these 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 was affected from the WRED mechanism and the level of how much it was affected is proportional to the values that the thresholds of the WRED mechanism had been configured at. In particular, as the values of the thresholds approached the maximum queue size, the packet loss that the background traffic experienced, decreased. Generally, from those tests we can draw the conclusion that on a fully congested network the WRED mechanism should be configured for the best-effort service, but the values for the thresholds should be selected carefully.

Our next steps include plans to continue our work on this area by extending those experiments on the bigger topology of the 6NET network [17]. We also plan to test some other mechanisms such as policing at the ingress router of the DiffServ domain. A very 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 follow the accorded SLAs [18]. 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 preagreed rules (the mean rate that they send packets, their maximum bursts etc.).

In addition, we plan to implement a second QoS service on the same domain, based on assured forwarding per hop behavior (AF PHB) that will operate in conjunction with this expedited forwarding QoS service. Finally, we are also interested on testing additional QoS features that will be possibly introduced in later versions of CISCO IOS for IPv6 QoS.

ACKNOWLEDGMENTS

This work was partially supported by the 6NET project funded by the IST program of the European Commission (IST Contract No: 2001-32603) [17].

REFERENCES

1. S. Deering and R. Hinden, *RFC 2460, Internet Protocol, Version 6 (IPv6) Specification*, Internet Engineering Task Force, December 1998.
2. S. Vegesna, *IP Quality of Service: The Complete Resource for Understanding and Deploying IP Quality of Service for Cisco Networks*, Cisco Press, Indianapolis, USA, 2001.

3. *OpenH323 Project*, http://www.openh323.org and http://sourceforge.net/projects/openh323
4. C. Bouras, A. Gkamas, D. Primpas, and K. Stamos, *Performance Evaluation of an IPv6-Capable H323 Application*, The 18th International Conference on Advanced Information Networking and Applications (AINA 2004), Fukuoka, Japan, March 29–31, 2004, pp. 470–475.
5. S. Josset, C. Bouras, A. Gkamas, and K. Stamos, *Adding IPv6 Support to H323: Gnomemeeting/OpenH323 Port*, 11th International Conference on Software Telecommunications and Computer Networks (SoftCOM 2003), Croatia, Italy, October 7–10, 2003, pp. 458–462.
6. Integrated Services (IntServ) charter: http://www.ietf.org/html.charters/intserv-charter.html
7. Differentiated Services (DiffServ) charter: http://www.ietf.org/html.charters/OLD/diffserv-charter.html
8. H. A. Akhand and M. A. Bauer, Managing Quality-of-Service in Internet Applications Using Differentiated Services, *Journal of Network and Systems Management*, Vol. 10, No. 1, 2002.
9. B. Carpenter and K. Nichols, *RFC 3086, Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification*, Internet Engineering Task Force, April 2001.
10. V. Jacobson, K. Nicholsm, and K. Poduri, *RFC 2598, An Expedited Forwarding PHB*, Internet Engineering Task Force, June 1999.
11. F. Baker, J. Heinanen, W. Weiss, and J Wroclawski, *RFC 2597, Assured Forwarding PHB Group*, Internet Engineering Task Force, June 1999.
12. Cisco Systems, Inc.: http://www.cisco.com
13. Iperf: The TCP/UDP Bandwidth Management Tool, http://dast.nlanr.net/Projects/Iperf/
14. S. Casner, R. Frederick, V. Jacobson, and H. Schulzrinne, *RFC 3550, RTP: A Transport Protocol for Real-Time Applications*, Internet Engineering Task Force, July 2003.
15. C. Bouras, A. Gkamas, D. Primpas, and 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, pp. 470–475.
16. Ethereal: A Network Protocol Analyzer, http://www.ethereal.com/
17. 6NET project: http://www.6net.org
18. G. Fankhauser, B. Plattner, and D. Schweikert, *Service Level Agreement Trading for the Differentiated Services Architecture*, Swiss Federal Institute of Technology, Computer Engineering and Networks Lab, Technical Report No. 59, November 1999.

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.

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 has been working with Research Unit 6 of CTI.