Approach to the Dynamic Forwarding of Packets in a Differentiated Service Based Router

Goncalo Quadros, António Alves, Henrique Matos, João Silva, Edmundo Monteiro, Fernando Boavida

CISUC – Centro de Informática e Sistemas da Universidade de Coimbra
Departamento de Engenharia Informática
Polo II, Pinhal de Marrocos, 3030 Coimbra, PORTUGAL
{quadros, aalves, kikas, jonas, edmundo, boavida}@dei.uc.pt

ABSTRACT

In the LCT-UC[1] we are currently working on a new service model for Internet based communications. The main goal of this model is to provide QoS without too much complexity. For this, we need to reinvent and develop new mechanisms for routers, which effectively can differentiate the way packets are treated.

We assume that traffic can be roughly well classified in some limited number of different classes, characterized by different QoS needs – we follow the IETF differentiated service model. Classes are the only possible choice if we want to maintain systems simple. For the same reason we know that we must avoid traffic specifications and explicit resource reservation. The main idea behind our work is to measure the impact of degradations on the applications whose traffic we need to deal with and, according to that, to redistribute the communications resources when their availability fall to very low levels. We use a QoS metric developed in LCT for this purpose.

Among the main mechanisms that had to be reconstructed for the routers there were the packet scheduler and the related queuing system. As we were mainly using the FreeBSD operating system in our laboratory, we decided to use the ALTQ technology as the queuing system. Our first approach foresaw the use of the ALTQ *weighted fair queing* (WFQ) implementation as well. Nevertheless, we verified that the WFQ/ALTQ didn't behave as it should, so we decided to develop a new scheduler, taking advantage of the lessons learned during this process. This scheduler is able to dynamically schedule packets, according to the varying router capacity to forward packets and to the impact of performance degradation on applications.

This paper presents our scheduling approach, the first version of the prototype under construction, and some tests that prove its ability to efficiently differentiate traffic classes.

Keywords: IP QoS, packet scheduling with QoS

1. INTRODUCTION

If doubts did exist in the past, nowadays it is evident that the Internet is the natural way to interconnect computers. Given this, and the emergence of applications that generate new kinds of traffic, it is particularly relevant to re-think this network, conceived when computer utilization had a very different meaning. The idea is to selectively bring application needs (which will be highly variable) closer to the communication system offer.

To achieve this, new mechanisms must be conceived. In particular, mechanisms that perform admission control and implement better packet scheduling strategies will be essential in the future Internet. Ideally, packets should be dynamically scheduled according to the different sensitivities their applications have to the continuous change in QoS characteristics, such as transit delay and losses.

The IETF Differentiated Service model [DIFFSERV] proposes the classification of data packets in classes as a first step to give them a differentiated treatment in the network. This paper presents a new mechanism, to be used in a Differentiated Service core router, which enables the dynamic scheduling of packets taking into account the different applications' sensitivities to QoS changes. It also presents a prototype we have implemented, and the tests we made to evaluate the performance and the effectiveness of the mechanism in differentiating traffic.

The prototype uses a metric developed at the University of Coimbra to monitor QoS in packet switched networks, considering the diversity of applications' sensitivities to QoS changes. Its design integrates widely used technologies — it was conceived for a FreeBSD system and uses the ALTQ approach as an alternative to IP queuing system (the common queuing scheme implemented by BSD Unix, the tail-drop FIFO, is too simple and do not provide QoS guarantees) [Cho98, Cho99].

[1] Communications and Telematic Laboratory of the University of Coimbra.

This paper is structured as follows: section 2 introduces our approach to the dynamic forwarding of packets based on QoS needs. Section 3 presents the prototype we are developing and discusses the main challenges we faced when designing and implementing it. Section 4 refers a first phase of experiments with the prototype and presents the correspondent results. Finally, section 5 presents some conclusions and refers future work.

2. A NEW APPROACH TO DINAMICALLY SCHEDULE IP PACKETS

The work that is being developed at the LCT intents to define and to test a new service model for IP networks with a real capacity to provide QoS to applications, without requiring a substantial displacement from the current IP model. In fact, our main goal is to develop a very simple model that can be used easily wherever QoS is needed in IP networks.

There are some router's mechanisms that are crucial for the success of this task, such as the queuing system and the related packet scheduler. The scheduler shall be able to conveniently assign the resources of the router, according to a given criterion. As we are looking for simplicity, we decided to avoid traffic specifications and explicit resource reservations. Instead, we want a simple way to evaluate the impact of performance degradation at applications and to dynamically correct the performance given to the traffic according to that evaluation.

Our idea is to use the sensitivity that different *traffic classes* have to QoS degradation for the measuring and redistribution of QoS among the classes. This is actually done using the QoS metric developed at LCT [Quadros98]. The only QoS characteristics we consider in our model are the transit delay and the packets' losses, as we believe that we can install sufficiently interesting QoS capabilities at communication systems by evaluating and controlling these two QoS parameters.

For short, we are using the *congestion indexes* (CI) of our metric associated with delays and losses to manage the packet scheduler and the dropper mechanism in the QoS capable router. We recall that the system's main goal is the equality of the delay indexes for all the application flows[2]. When that happens, despite the different delays and losses affecting the various QoS classes, the impact suffered by all the involved applications is supposed to be equal, so the resources are fairly well distributed among them.

Figure 1 exemplifies the concept just referred. The graph on the left side corresponds to a class with low tolerance to additional delay (a delay beyond the expectable value in a slightly loaded network). In this case the impact of the degradation grows fast with the delay (the *degradation threshold* is very low, or the *degradation slope* is high). The second and third graphs correspond to classes that have successively less sensitivity to delay degradation (i.e., lower degradation slopes).

The system will try to maintain the same congestion index for all the classes, which is the *system congestion index* (represented as CI_t in the figure). Naturally, whenever the load grows the system index will grow accordingly. So it can be used to evaluate the global performance given by the system, or its congestion state.

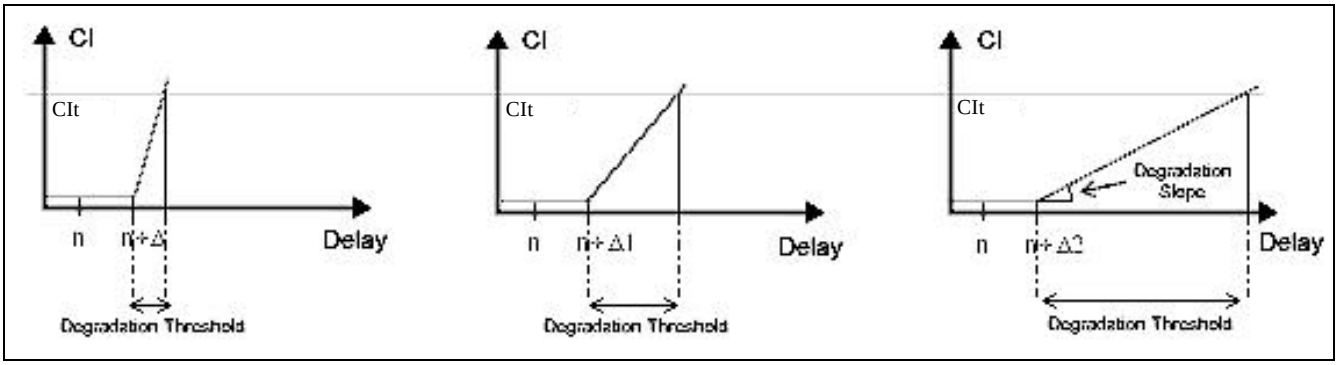

Figure 1 – Congestion Index *vs* IP transit delay (for traffic classes with different sensitivity to transit delay degradation)

The first thing that shall be considered in the scheduling of IP packets, taking into account the quality of service, is the router's queuing system. In fact, the traditional queuing mechanism based on a single FIFO tail-drop queue is inadequate for enabling QoS capacities in the router. We considered two alternatives:

- The use of different queues, each one with a given, fixed priority. Packets would be enqueued in a higher or a lower priority queue according to the congestion index of the respective class;

[2] The same for the losses indexes.

- The use of different queues, one for each class. Packets would be put in the correspondent (according to its class) queue.

In the first method the scheduler is essentially static, as it always gives a better treatment to packets that belong to queues with a higher priority. This implies the use of a mechanism that evaluates the classes congestion indexes all the time, deciding for each arriving packet which queue to use (for instance, the higher priority queue when the index is markedly higher than system CI[3], or the lowest priority queue when the opposite happens).

In the second approach the scheduler has a dynamic behavior. The packets are put in the queues based on the class they belong to. The role of the scheduler is to determine, in each iteration, how the packets in each queue shall be treated. For this, it uses the class congestion indexes.

The second approach is more flexible, as the scheduler is designed from scratch to dynamically adapt to varying operational conditions in the communication system. By opposition, the scheduler in the first approach shall be statically configured, and we can expect a poorer router performance level. Another and decisive advantage of the second approach is that it does not imply packet re-ordering. For all these reasons, we decided to adopt the second mechanism.

2.1 Scheduler architecture and main characteristics

The core of our prototype is logically illustrated in Figure 2, where we can see two nuclear components: the classifier and the packet scheduler.

One of the first problems we faced when designing our prototype was the need for an alternative queuing discipline other than the traditional FIFO queuing one. Rather than strongly modify a BSD UNIX kernel to handle a different queuing method, we decided to use the FreeBSD [FreeBSD] system with the ALTQ mechanism [Cho98,Cho99]. ALTQ is a queuing framework designed to support different queuing disciplines and it is implemented as an extension to the FreeBSD kernel. With this technology, we substituted the single IP output queue by several ones (as many as the number of classes) that actually support the packet scheduler operation. The ALTQ extension allowed us to concentrate on the implementation of the actual elements of the queuing architecture without spending too much time in other parts of the packet forwarding path of the kernel network code.

During the development phase our goal was to achieve a fully functional prototype with characteristics such as simplicity, efficiency and low overhead. The prototype was developed in a router with three ethernet 100BaseTX network interfaces.

In a first stage, we decided to use the WFQ (Weighted Fair Queuing) discipline implemented by the ALTQ team. In fact, we only needed a way to dynamically differentiate the treatment given to the different classes of traffic. The WFQ/ALTQ scheduler was available and we thought it would be interesting to take that challenge. This scheduler treats the classes (or the correspondent queues) in a round robin way, processing at each iteration a different number of packets per queue, according to their weights. As we can dynamically change the configured weight for each class, we admitted that we could effectively control the QoS given to each class. The idea was to use the measured congestion indexes to exert that control (figure 2 reflects the use of the WFQ/ALTQ discipline).

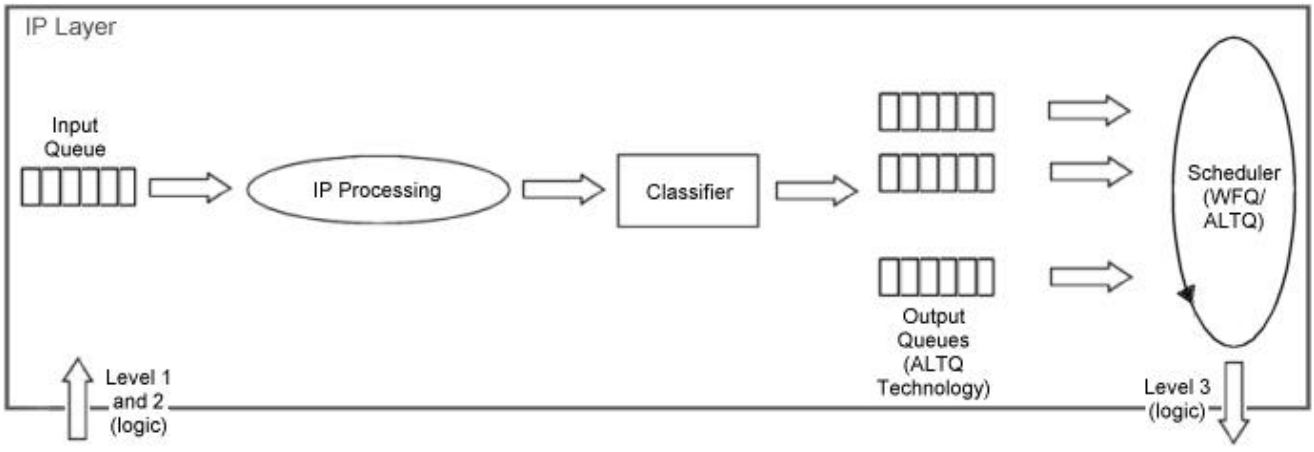

Figure 2 – Prototype logical architecture

[3] Meaning that the class needs a better treatment.

However, the experiments we did with the WFQ/ALTQ implementation revealed that it did not behave according to our expectation [Quadros99]. We realized that it would only be useful if we changed the packet drop mechanism. In turn, to use a simple tail-drop mechanism, we had to enlarge the ALTQ maximum queue lengths, which resulted in important additional packet delay. The problem was identified: the WFQ/ALTQ only works as expected when there are several packets simultaneously in the different queues. We verified by performing some experiments that this is not the normal case, even when the router is submitted to very high load conditions, built on UDP traffic.

Given that, and by the experience gained with the tests, we decided to use a different scheduler, designed and implemented at our laboratory. This scheduler, along with the classifier, are two fundamental components in our prototype, and are presented in the next section.

3. THE PROTOTYPE

This section describes our prototype, which is still under construction as referred before.

Several independent functional blocks have been developed, with the most important ones located at the packet forwarding path in the network node. Together they provide the necessary mechanisms for the desired forwarding behavior.

Basically, this set of modules represents a queuing architecture or, in other words, the elements of the traffic control. These elements include the packet classifier, the traffic meters, the very important packet scheduler, and the correspondent queuing mechanism.

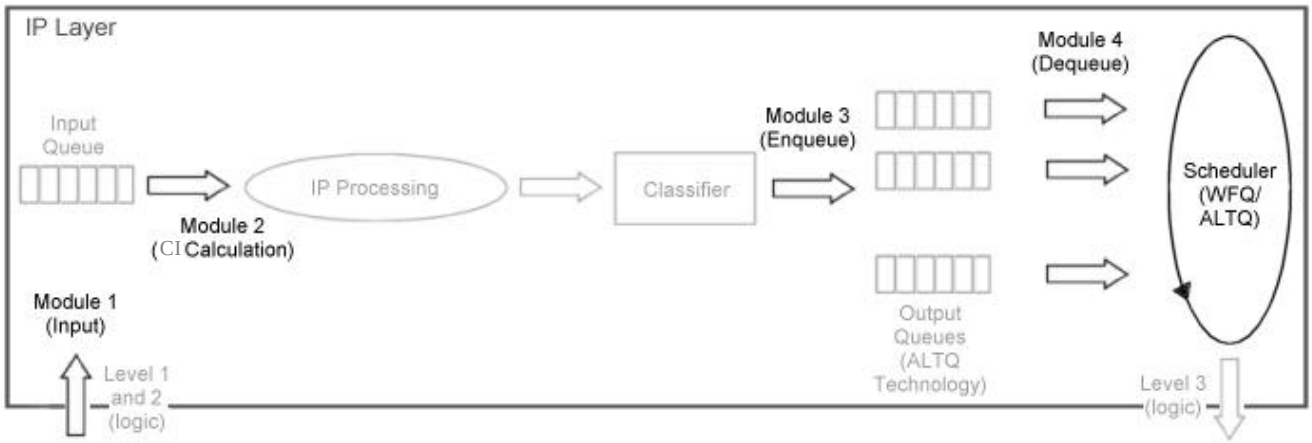

Figure 3 - The prototype modules

3.1 Principal prototype modules

Figure 3 shows the location of the principal modules in the forwarding path. The first module (Input) is located just before a packet is added to the IP input queue, and is responsible for the time stamping of the arrival time of packets. Module 2 (CI Calculation) is located inside the IP layer and does some calculations based on properties of the traffic streams. Modules 3 (Enqueue) and 4 (Dequeue) represent the enqueuing and dequeuing processes.

Module 1 – Input

This module time stamps the arrival time of the packets at the IP input queue. It works in conjunction with a sub-module of module 4 to calculate the average packet delay in the forwarding path. When designing this module we were faced with a major problem, as where to store the time stamp of each packet. We first thought of the mbuf structure. However, the mbuf structure of the FreeBSD kernel does not contain any extra field, and this way can not carry the needed information. Another option was to transport the information in the data area of the mbuf, although this is not a neat solution. Finally, we found a very simple scheme that just needed a few patchs to the original BSD mbuf handling code. Starting with the original size of 128 bytes for each mbuf, we increased it to 256 bytes. This way, each mbuf is still compatible with the original kernel code giving, at the same time, an additional 128 bytes to store the time stamp and other information about the packet. The only drawback of this scheme is that the kernel is allocating twice the original network memory buffers. However this

is not so problematic as memory is quite inexpensive these days. The actual stamping code represents just a few lines of code included in the IF_ENQUEUE() macro.

Regarding the clock source to time stamp the packet, we decided to use the 64-bit time stamp counter included in Intel Pentium processors, for efficiency reasons.

Module 2 – CI calculations

The code of this module is located inside the IP network layer running at *splnet*[4] [Wright96]. The decision of placing the code there relates to the intensive calculations needed to get the current CI for each class based on past average delay calculations. A large structure holds the current and average values for medium and long-term CI values for all active classes. The CIs for each class are recalculated after a pre-configured period of time. A sub-module in module 4 controls this period of time.

Module 3 – Enqueue

This module is conceptually very large, and includes the classifier, the marker and the dropper mechanisms. In terms of code, however, it is quite small compared, for example, with the dequeue process (in the current implementation).

Classifier and Marker: the classification of packets into classes is done by comparing information present at the IP packet header, such as the source and destination addresses, and the rules present in a table configured by the user. As soon as a packet arrives at the Enqueue module its DS field is checked to verify if the packet is already marked in our domain. If yes, the packet is not compared with the user classification rules, being assigned to the corresponding class based on a codepoint in the DS field. In the other case, a search is made through the table of rules until whether a match is done or no match at all is found. In this later case the packet is sent to the default class. The packet is then marked or remarked if necessary with the corresponding codepoint.

Right now, the classifier is very simple, as it is not a current priority in the prototype. At this time, packets that leave this node are only compatible with diffserv compliant nodes and can cause strange behaviors if received by a non-compliant node.

Dropper: this is work in progress. In the current implementation we adopted a traditional drop-tail scheme. This mechanism simply drops the packet when the corresponding queue is full.

Module 4 – Output

This module includes the very important packet scheduling mechanism, as well as other sub-modules. It is here that the decision for the next packet to be sent through the output interface is made.

A good scheduler shall be able to efficiently respond to the various inputs it receives from the other modules in the queuing discipline, and to correctly output the right packet. The sharing of the output link shall be as precise as possible, taking into consideration the importance of the class of the packet waiting to be sent through the network. However, this is not a simple and trivial task to implement.

The Scheduler: our algorithm tries to do the best regarding fairness and protection, and takes into consideration the dynamics of the underlying system. In fact, we dedicated a great deal of time evaluating the dynamics of the network code in the FreeBSD kernel, due to its influence in the behavior of the scheduler. Although this is work in progress, we already got very positive and interesting results with the current implementation.

Basically, the algorithm is a class-based, temporal round robin. There is a queue for each class and each class has its priority based on the higher or lower importance of the packet average delay in the forwarding path of the node (which will vary along the time).

The arriving packets are classified into the appropriate class, as described in module 3, and the scheduler outputs the packet from the current class being serviced if it is already the time to do so. Otherwise the scheduler skips into the next class. The time that each class wait to be serviced depends on its degradation slope and on the current CI of the *reference* class (i. e., the highest priority class). If we take into consideration that the reference class is always serviced in each round by the scheduler, all the other classes are more or less delayed in accordance to it. The goal is the equality of all the classes' CIs. The lowest priority class, also known as the *default* class, is only serviced when there are no other packets waiting to be serviced on the scheduler.

CI to packet scheduling mapping: the mapping of the current class' CI into the time each class waits to be serviced on the scheduler is another important part of our implementation. The current implementation uses a fairly simple algorithm

[4] The normal priority level for network protocol processing. We avoid code zones executed at higher priority levels (for instance *splimp*)because that would result in a higher probability of the system miss the execution of some important task.

based on intervals. Although simple, this scheme has already presented acceptable results. However, there is a lot of improvement to be done in this area, and this definitely constitutes a matter of further study.

Packet Average Delay: This is the end of the line; the packet is being sent to the network device driver. Based on the time stamping in module 1, the average packet delay for each class, which will be used later by module 2, is calculated. This sub-module is also responsible for scheduling the next run of module 2.

Statistics

Although not quite an independent module, the statistics play an important role in the whole prototype architecture. We biased the statistics collection towards queuing behavior, as it helps us tuning the prototype, although it also gives other high level data that is normally useful to the network administrator. Information such as how many queues are available at a given time in the scheduler, or the average packet delay, or even the bytes sent by each class, is made available through a character device driver to the graphical application QoStat []. This user level application displays a great deal of information from the kernel in several graphics' formats. It also allows the configuration of several parameters in various modules and the immediate graphical representation of the system reaction. Finally, this application can be used as a learning tool for the study of different types of queuing disciplines [Alves 99].

4. TESTS AND RESULTS

The development of our prototype includes an extensive phase of experiments. Figure 4 schematizes the testbed we are using in these experiments. It consists of a small isolated network with 4 Intel Pentium PC machines configured with a Celeron 333Mhz CPU, 32 MB ram and Intel EtherExpress Pro100B network cards. The prototype PC Router (named HOST_R in the figure) runs FreeBSD 2.2.6, patched with ALTQ version 1.0.1, with 64MB ram. HOST_S1 and HOST_S2 are used to generate traffic directed towards HOST_D, through HOST_R.

In this section we present the first stage of the tests we have already made with the prototype to evaluate its performance and its real capacity to differentiate the performance given to different traffic classes.

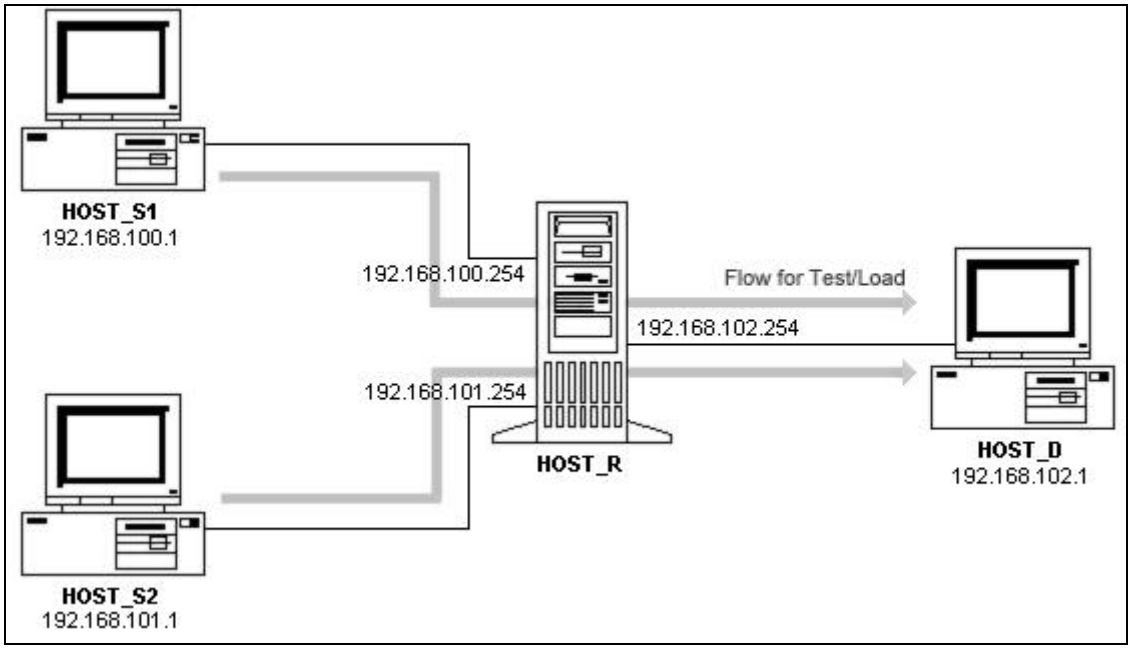

Figure 4 – The testbed

We used *netperf* (Network Performance Benchmark) [NETPERF] and *netPIPE* (Network Protocol Independent Performance Evaluator) [NETPIPE] tools in our tests.

Netperf was used to generate UDP traffic at the maximum throughput allowed by the network. NetPIPE was used to generate a probe flow of data (which also generates load). The use of NetPIPE in conjunction with the TCP protocol allowed to clearly checking the higher or lower availability of resources, or the network performance in fact attainable. This tool is based on the concepts used in the HINT computer performance metric []. The idea here is to evaluate the performance of the network using different size communication transfers. NetPIPE allows this by increasing the transfer block size from a single byte until the transmission time exceeds one second. It uses “ping-pong” transfers to make more rigorous evaluations.

The results of the experiments we made were represented with the *network signature graph* produced by netPIPE. This graph synthesizes important information about the network performance, such as the network latency (the time of the first data point on the graph) and the maximum attainable throughput. It depicts the transfer speed versus the elapsed time, so it actually is a *network acceleration graph* [NETPIPE]. The first points of the graph correspond to the lowest latencies measured, the same is to say, to the shortest message blocks. The last ones correspond to the largest message blocks.

For evaluating the performance overhead introduced by our prototype, we traced the network signature with netPIPE when using at HOST_R: (1) a normal FreeBSD kernel, and (2) a FreeBSD kernel with our prototype installed. As referred before, NetPIPE was executed in HOST_S1, and the data directed towards HOST_D through the router. The results of this test are shown in figure 5. As can be seen, the two curves are almost coincidental, which is a strong evidence of the very low overhead introduced by our prototype.

The second set of tests was made to evaluate the prototype capacity to differentiate traffic. We used Netperf in one of the source hosts to generate load, and we ran netPIPE in the other source host. Each flow of data was assigned to a different class. Then, we configured our prototype in a way that the traffic generated by netPIPE had a high sensitivity to delay degradation (degradation slope equal to 80 degrees), and the traffic generated by the netperf tool had a varying sensitivity to delay degradation (degradation slope equal to 60, 45, 30 and 15 degrees). We knew that despite the ping-pong transfer method used by netPIPE, and the availability of our scheduler in only one way of the data path, the signature graph should be able to clearly show the differences in case of a real installed capacity on router to differentiate traffic.

The signature graphic presented in figure 6 shows, in fact, those differences. By settling a lower sensitivity to transit delay degradation in the UDP traffic we suspected that the correspondent applications would not suffer a significant impact with increasing delays. This means that, in this case, the system would give a better relative treatment, or quality of service, to the TCP traffic. In fact that is clearly shown in the signature graph; we changed sensitivities manipulating the degradation slope that characterizes one of the flows (UDP traffic), and we consistently obtained worst or better QoS for the other flow (TCP traffic).

The differences can be seen in the entire graph but are more evident when the load is higher (right part of the graph). So our prototype is undoubtedly able to differentiate traffic even with an uneasy traffic mix (UDP and TCP).

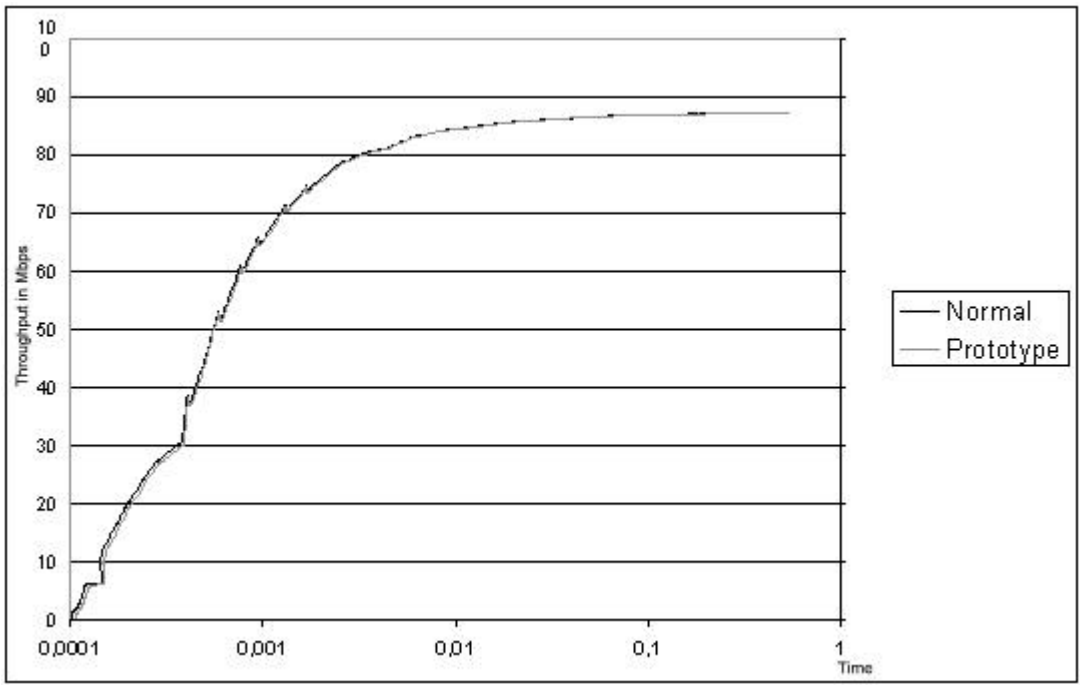

Figure 5 – Netpipe signature graph with a normal kernel, and with a kernel with the prototype installed

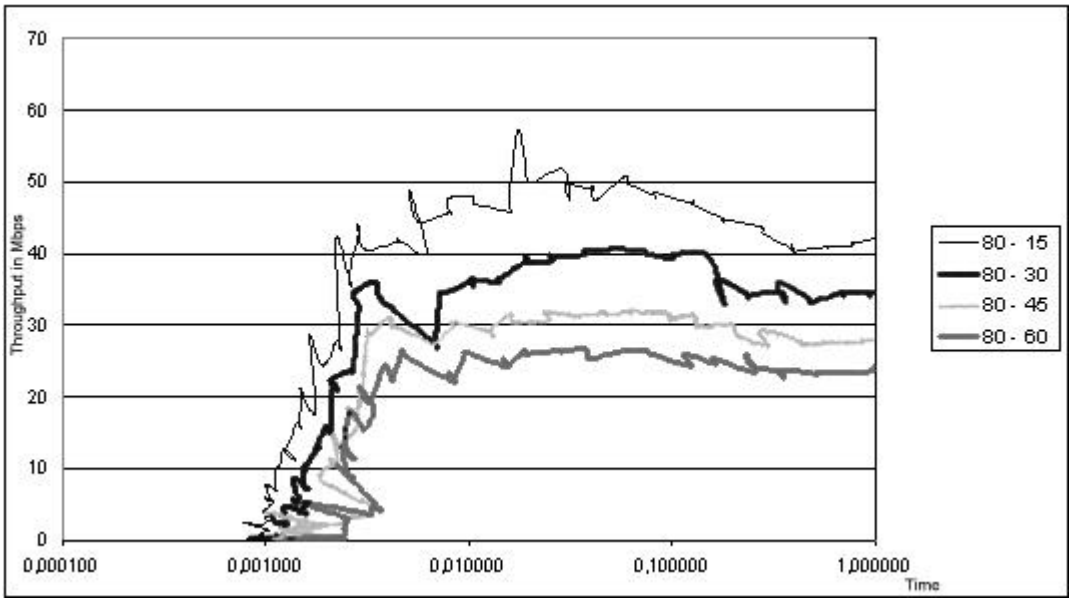

Figure 6 – NetPipe signature graphs using the prototype and a degradation threshold equal to 80 for TCP traffic and different degradation thresholds for UDP traffic (15, 30, 45 and 60).

5. CONCLUSION AND FUTURE WORK

The LCT-UC group is developing a new model of service for the Internet, based on the Differentiated Service approach. This model is characterized by its extremely simplicity and its capacity to dynamically scheduling packets according to the

reaction of applications to the continuos changes in QoS network parameters. For this purpose, the model uses a metric developed at our group that monitors quality of service in packet switched networks.

The main idea of this technique is simple: the traffic generated by different applications is divided into several classes. Each class has a certain priority that is computed *dynamically* based on its delay and loss congestion indexes (CI) at a given time. I.e., the priority of each class changes according to its sensibility to QoS changes in the network. The goal of the entire system is to fairly share the system resources by equalizing the delay and the loss indexes for all the classes.

In this paper, we referred the current state of development of this model and described the principal modules that constitute our prototype, as well as the main challenges we faced when designing and implementing it. More particularly, we focused on two crucial router mechanisms, the queuing system and the related packet scheduler, and showed how these mechanisms used the CI indexes to reach their desired behavior.

Finally, we described a phase of experiments we have done with the prototype, and presented some conclusions about the results obtained. In a preliminary interpretation of the results, it seems that our prototype is undoubtedly able to differentiate traffic, even in the case of uneasy traffic mix.

6. REFERENCES

[Quadros98] Gonçalo Quadros, Edmundo Monteiro, Fernando Boavida, "A QoS Metric for Packets Networks", in *Proceedings of SPIES's symposium on Voice, Video, and Data Communications conference on Quality of Service Issues Related to Internet*, Boston, MA, USA, November 2-5, 1998.

[Quadros99] Gonçalo Quadros, António Alves, *Relatório de Testes – Encaminhador WFQ, implementação ALTQ-FreeBSD*, Relatório Técnico, CISUC, Julho 99 (portuguese).

[Alves99], António Alves, Gonçalo Quadros, *Qostat – Uma Ferramenta de Monitorização de QoS para a Camada IP de um Sistema FreeBSD*, Relatorio Técnico, CISUC, Agosto 99, (portuguese).

[NETPERF] Hewlett-Packard Company, *Netperf: A Network Performance Benchmark*, Fevereiro 96;
www.netperf.org

[Cho98] Kenjiro Cho, A Framework for Alternate Queueing: Towards Traffic Management by PC Based Routers, in *Proceedings of USENIX 1998 Annual Technical Conference*, New Orleans LA, June 1998.
http://www.csl.sony.co.jp/person/kjc/papers.html

[DIFFSERV] DDDD

[FreeBSD] DDDD

[NETPIPE] Q.O. Snell, A.Mikler, and J.L. Gustafson, NetPIPE: A Network Protocol Independent Performance Evaluator, IASTED International Conference on Intelligent Information Management and Systems, June 1996.
www.scl.ameslab.gov/Personnel/quinn.html/

[Cho99] Kenjiro Cho, Managing Traffic with ALTQ, in USENIX 1999 Annual Technical Conference: FREENIX Track, Monterey CA, June 1999.
www.csl.sony.co.jp/person/kjc/papers.html

[Wright96] Gary Wright, W. Richard Stevens, *TCP/IP Illustrated, Volume 2, The Implementation*, Addison-Wesley Professional Computing Series, March 96.