QoStat - A tool for the evaluation of QoS capable routers

Antonio Alves, Goncalo Quadros, 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
Tel: +351-239 790000; Fax: +351-239 701266
{quadros, aalves, edmundo, boavida}@dei.uc.pt

ABSTRACT

At the Laboratory of Communications and Telematics of the University of Coimbra we have been working on the IP QoS field. Our work includes researching, testing and implementing mechanisms that can be used on current IP networks in order to make them able to provide different performance levels to traffic with different QoS needs.
Our work is being supported by a testbed of INTEL hosts running the FreeBSD operating system patched with ALTQ software. This is a common platform in this research field. We chose it, in part, for that reason - there is much work produced for this environment in which we are interested - and also because it is fully open, allowing us to freely test new approaches for QoS provision.
Despite the amount of work being done in this area, soon we realized the lack of a tool to monitor with detail the operation of the IP layer on routers, in particular, the mechanisms related to QoS provision. It was evident we needed such a tool, so we decided to implement one - which we named QoStat.
This paper presents QoStat a GUI based tool, which gives us the possibility to visualize in real time the most important values related to QoS provision. With it we can also change on the fly the most important operational parameters of the mechanisms associated with QoS provision. In this way, it is possible to understand with depth the behavior of those systems - for instance, to study cause-effect relations between the values of the referred parameters and the QoS in fact provided to the different IP traffic flows or classes.

Keywords: QoS capable routers, QoS Monitoring Tool

1. INTRODUCTION

During our work at LCT-UC[1] we carried out some tests on FreeBSD Unix Routers [7], with QoS capable schedulers and other mechanisms related with QoS provision. Soon, it became obvious that it was needed a tool capable of providing a better understanding about the behavior of those mechanisms. In fact, in many situations the observed behavior was not as we expected. This required a detailed analysis, which could only be accomplished with a specialized tool.
That tool should provide a way to rigorously measure some characteristics directly related to the operation of the mechanisms used, such as the average transit delay of packets, the number of packets processed and dropped per unit of time, the length of the different queues, etc. Conversely, to understand the influence of the operational parameters of such mechanisms on the monitored traffic, we needed a way to dynamically alter their values and to visualize in real-time the changes of the measured values.
As there is no such a tool available we decided to implement one. For short, our main goal was to develop a monitor that was able to:
- Graphically present, in real time, IP traffic measures as the ones referred above[2]. The purpose was to enable some flexibility on the definition of the measuring time scale and type of statistical treatment (allowing, for instance, average and maximum values of the evaluated parameters).
- Change on the fly some fundamental parameters related to kernel operation, such as, scheduler queue lengths or queue priorities.

An additional goal was to implement a tool as general as possible, i.e. applicable to the monitorization of a wide set of QoS related mechanisms. Nevertheless, the mechanisms we want to monitor are normally implemented in system space, and their functionality substantially diverges from one to another. Given that, we knew from the very beginning that we could not construct a fully generic tool, directly usable with all the mechanisms we might monitor. We assume that, depending on

[1] Laboratory of Communications and Telematics of the University of Coimbra.
[2] A printing option was also included.

the mechanism to monitor, some code customization must be done. Nevertheless, we claim that this task is simple to perform.
This paper presents the result of our work - a tool named QoStat constructed to be used in FreeBSD systems. In the next section we present its architecture and the technologies it uses. In section 3 we detail its functionality, namely the measures we can get, the parameters we can modify, the types of graphics, etc. Section 4 presents some examples that express the usefulness of QoStat. Finally, section 5 concludes our paper.

2. ARCHITECTURE AND TECHNOLOGIES USED

Two fundamental components compose the QoStat tool. One runs in user space, and is responsible for the visualization and interaction with the user. The other runs in system space, and is responsible for the measurement tasks. Figure 1 generally presents the QoStat architecture. The modules that were specifically developed for this tool are marked in gray.

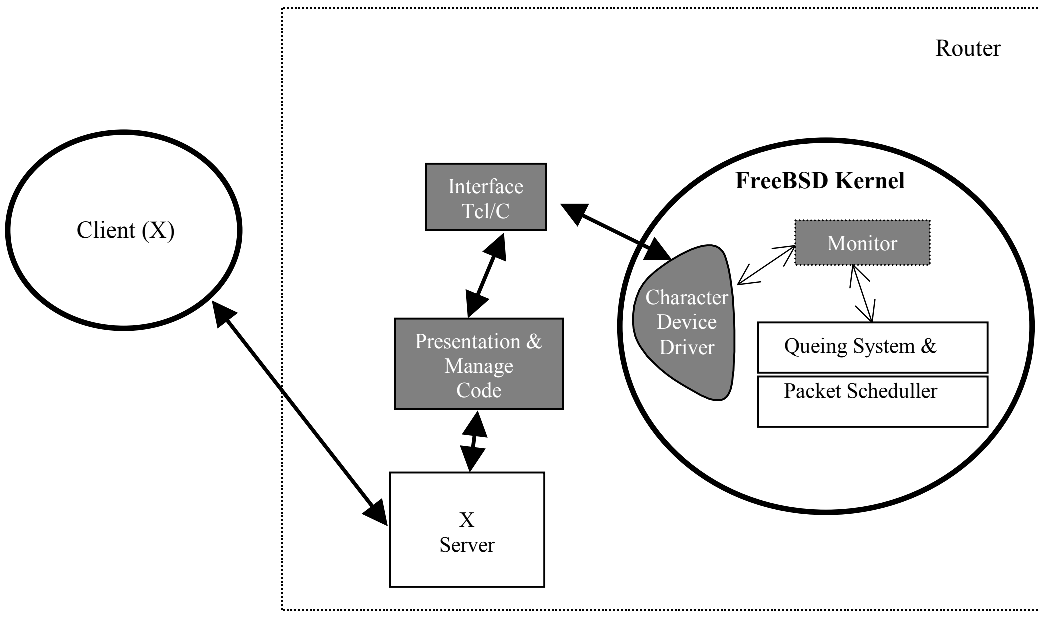


Figure 1. QoStat architecture

The *presentation & manager* module is the nuclear part of the component running in user space. It implements the interface with the user and is responsible for all the tasks related to the presentation of measurements. This module was developed using Tcl/Tk technology, given its powerful capabilities and flexibility to design graphical interfaces, and its nice portability characteristics.
The *monitor* module is the major component executing in system space. In fact, the monitor is not a real module. It is essentially composed of a set of data structures, manipulated by the different kernel mechanisms related to QoS provision (such as the scheduler), and susceptible to be read by the routines of a character device driver installed for this purpose.
Thus, this device driver, together with the *Interface Tcl/C* module, interfaces the user space component of the tool with the system space component. In other words, the *Presentation & Manager* module uses device driver commands, through the Tcl/C interface, to get the values it needs to construct the charts. On the other side, it uses X technology to allow the tool to be used from any X terminal available in the network.
Despite all the good characteristics we claim for QoStat, there is one which is not so good (but is unavoidable): the tool is tightly connected to the kernel of the monitored systems. Therefore, when we change, for instance, the type of the scheduler, we also need to change the code of the tool. This is not only true for the code running in system space. In fact, the code that is related to the user interface must also reflect the different operational parameters of each mechanism.
For instance, in order to perform meaningful tests when using a WFQ scheduler, the user should be able to change the weights associated with each class. Nevertheless, when using the scheduler implemented in our laboratory, the user should

be able to change the sensitivity to QoS degradation of each class. Concluding, the user space code must also be adapted according to the mechanisms used to provide QoS in the monitored system.
Considering that, one of our main concerns was to develop the tool in such a way that the adaptations or modifications needed when changing the mechanisms to monitor would be quite easy to do.
Figure 2 presents the IP layer of a generic system with capacity to provide QoS to traffic classes or flows of data. We will refer to this diagram again when presenting the measures provided by QoStat. For now, we only want to make some considerations about packets time stamping. For monitoring purposes, each packet must be time stamped when it arrives at the IP input queue. We decided to use the 64-bit time stamp counter included in Intel Pentium processors as the source clock. The major problem was where to store its value, for each packet.
We first thought of using a field of the FreeeBSD *mbuf* structure. However, this structure does not contain any extra field, and thus can not carry the needed information. Another option was to transport the information in the data area of the *mbuf*. As that would not be transparent for the rest of the code that uses *mbuf* structures we abandoned also that solution. Finally, we found a very simple scheme that just needed a few patches to the original BSD *mbuf* handling code. Starting with the original size of 128 bytes for each *mbuf*, we increased it to 256 bytes. In 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 and we are using only a few Kbyte more.

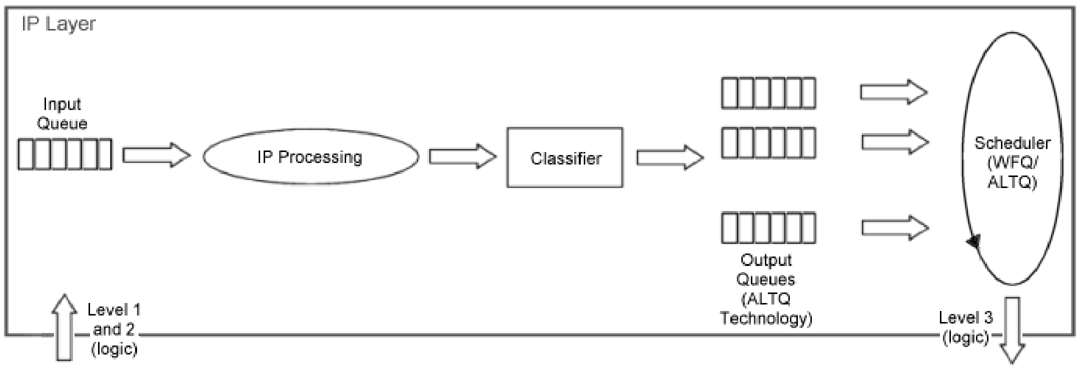


Figure 2. General architecture of a class differentiation system at IP layer.

The actual stamping code represents just a few lines of code included in the IF_ENQUEUE() macro.
In the next section we will present the basic version of the QoStat tool - to be used with the WFQ scheduler on FreeBSD v 2.2.8 operating system patched with ALTQ v 1.2 [3]. This is a widely available, and known, platform, used on testbeds for IP QoS experimentation. That was the reason why we chose this configuration for basic distribution. We show also that such version can be used as a starting point for developing its ability to monitor different systems, using other QoS related mechanisms.

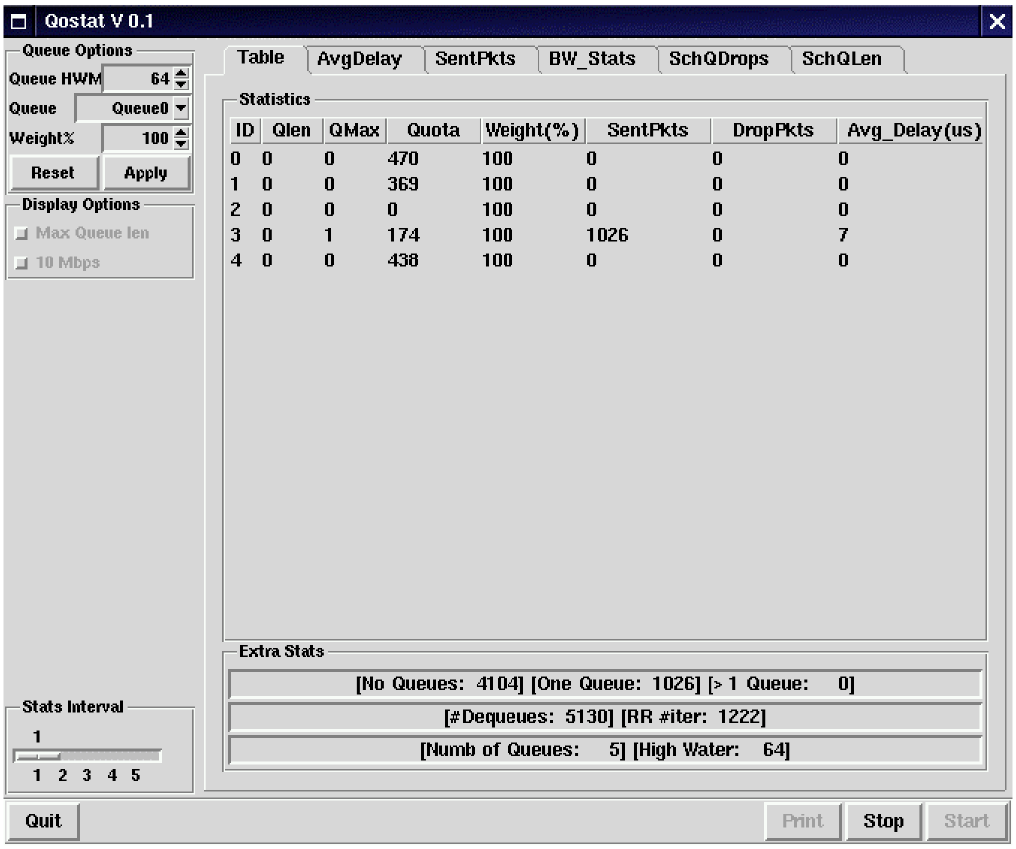
3. FUNCTIONALITY

3.1 Basic tool - WFQ/ALTQ monitoring

Figure 3 presents the basic structure of the QoStat GUI. The main screen - which is the one with the table *tab* activated, as presented in the figure - is divided into several rectangles. They organize the different types of information used on the interface and have the following names: *Statistics*, *Extra Stats*, *Stats Interval*, *Display Options* and *Queue Options*. The first two are only informative. The rest of them contain the means to change the operation of the tool, or some important parameters of the system being monitored.
In addition, there are different *tabs* that can be activated. Through them, some of the monitored values showed in the statistics rectangle can also be displayed graphically. In this way it is possible to follow, in real time and per queue, the average packet delay at the IP layer, the number of processed packets, or the number of dropped packets at output queues. It

is also possible to follow the length of each output queue (sampled value or maximum value on the *stat period* of time[3]), and the bandwidth share among all classes. Examples of such graphs are presented later in this paper.
Finally, there are some buttons in the bottom part of the screen. They allow to start or to stop the monitorization process, to print the results shown on the screen, or to quit the QoStat tool.
In order to explain the whole QoStat functionality, we will discuss with more detail the main screen rectangles mentioned above, starting with the *statistics* rectangle. This component shows the values of almost all the system characteristics that can be monitored. Consider that the *measurement interval* (MI) is the period of time defined through the *stats interval* button located in the bottom left zone of the screen; for each class - the same is to say, for each ALTQ queue - it is possible to watch:
- The QoStat queue identification - an internal id for each queue.
- The instantaneous queue length - sampled at the MI rate.
- The maximum length of the queue in each measurement interval.
- The last measured queue quota[4], for each measurement interval.
- The queue weight.
- The number of processed packets per MI.
- The number of dropped packets per MI.
- The average packet transit delay, for each measurement interval, at IP level[5].

Due to some difficulties in understanding the behavior of the WFQ/ALTQ scheduler we introduced in this version the *Extra Stats Rectangle*. It shows some additional values specific to the operation of this scheduler. More exactly, again for each



Queue HWM 64
Queue Queue0
Weight% 100

ID	Qlen	QMax	Quota	Weight(%)	SentPkts	DropPkts	Avg_Delay(us)
0	0	0	470	100	0	0	0
1	0	0	369	100	0	0	0
2	0	0	0	100	0	0	0
3	0	1	174	100	1026	0	7
4	0	0	438	100	0	0	0

```
[No Queues:  4104] [One Queue:  1026] [> 1 Queue:    0]
[#Dequeues:  5130] [RR #iter:  1222]
[Numb of Queues:    5] [High Water:   64]
```

Stats Interval 1

Figure 3. The QoStat main screen

[3] The *stat period* of time is explained below.
[4] Bytes processed in each scheduler visit to a queue with weight equal to 100.
[5] The Packet transit delay is the period of time since packet enqueuing into the input queue and its dequeing from the IP output queue (see figure 2).

measurement interval, the number of times the scheduler finds no packets on the queues, the number of times it finds packets in only one queue, and the number of times it finds packets in more than one queue. It also shows the number of times the 'dequeue' function is called, the number of iterations of the scheduler round robin and the configured queue size limit in Kbyte (High Water).
The left part of the screen vertically concentrates the rectangles that provide the users with the means to change some parameters related with the operation of the monitored system, or the tool itself. The *stats interval button* was referred before. It determines the period of time named 'measurement interval', which can be 1, 2, 3, 4 or 5 seconds. MI is used for statistic calculations (maximums and averages), occurrences count, or as the sample rate of instantaneous values.
One of the buttons of the *display options rectangle* allows the selection of the maximum network bandwidth to be used (10 or 100Mbps). Without it certain graphs will necessary use a very uninteresting y-axe scale. The other button is useful when the *SchQLen tab* is selected. Through it, it is possible to choose whether to show the sampled instantaneous queue length or the maximum queue length achieved during each measurement interval.
Lastly, through the *queue options rectangle* one can determine the value of the *high water mark* (the maximum number of bytes that can be simultaneously in all IP output queues) and each class weight. These values can be freely changed, and applied simultaneously only when the user wants so. It is also possible to reset them to a default reference.
Figures 4 and 5 present examples of the graphs constructed by QoStat - namely the sent packets per MI and the bandwidth share among classes, respectively. These graphs are constructed in real-time, refreshed at the MI rate. They represent the two types of graphs available in the tool. The first is a line graph - after each MI a new point is added to the graph. The second is a column graph, the bars present values measured in the last MI. The majority of the graphs are of the first type.

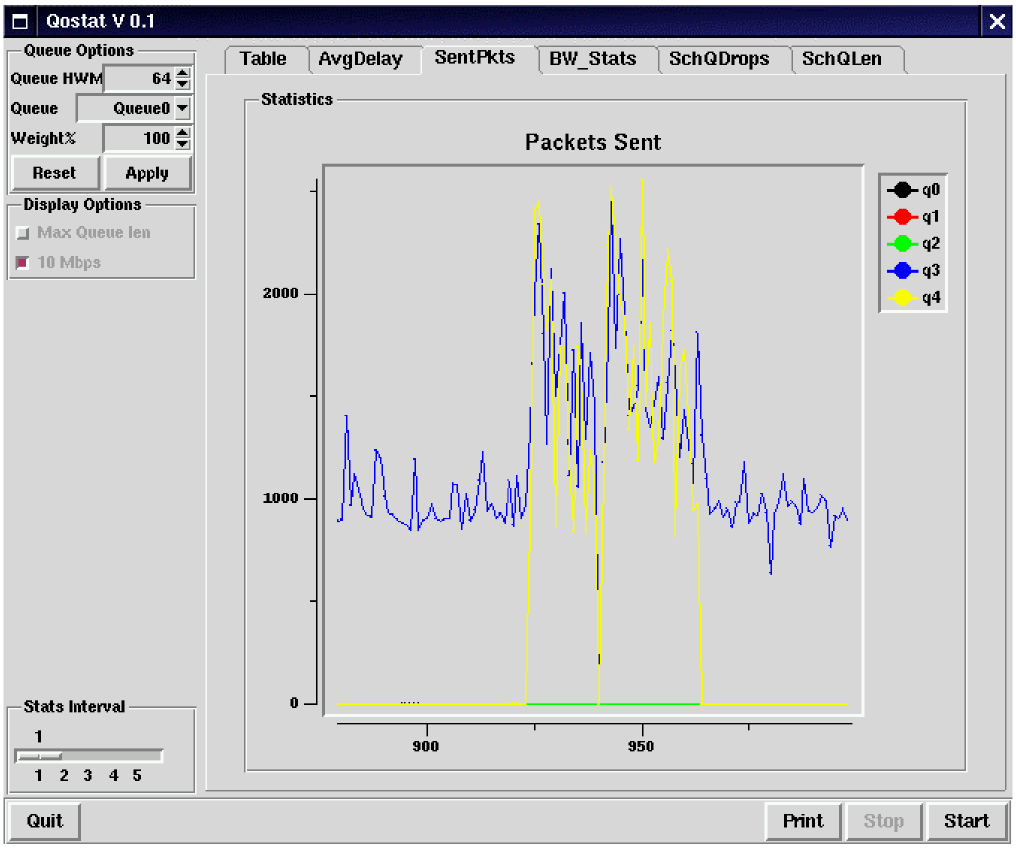


Figure 4. A QoStat screen - number of packets sent per queue

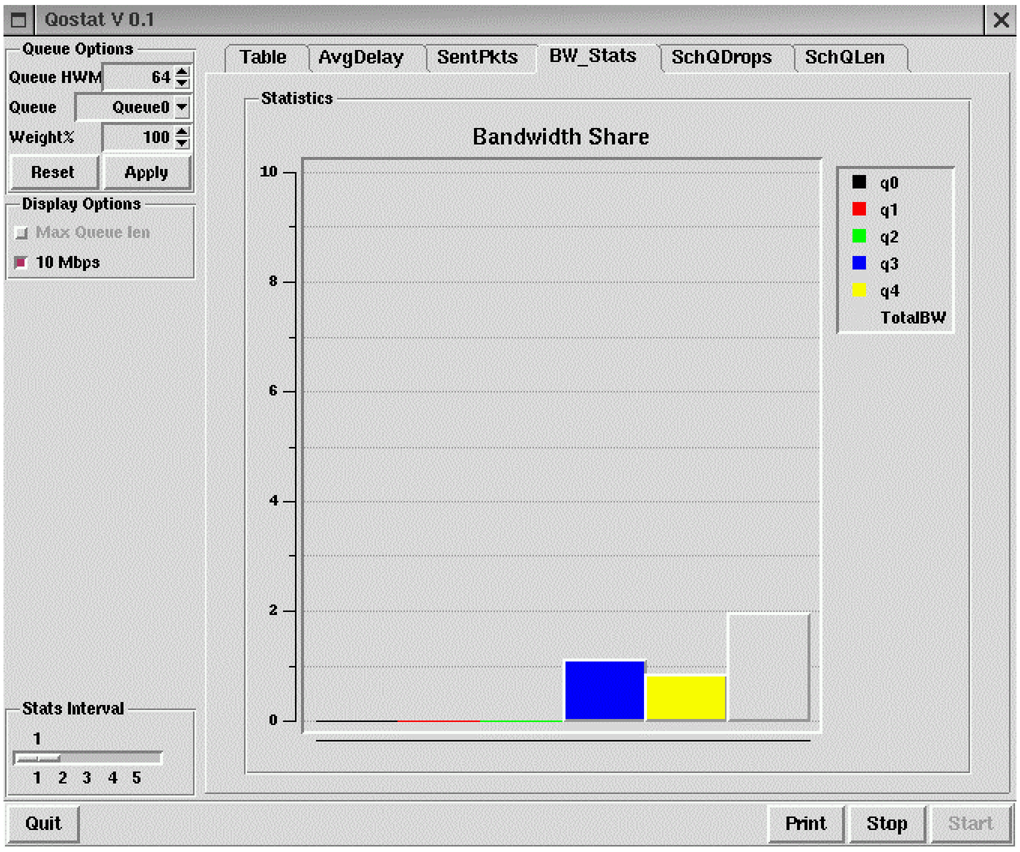


Figure 5. A QoStat screen - Bandwidth Share

3.2 Enhanced Tool - LCT UC prototype monitoring

We extended the features of the tool described above to support the development of a QoS capable router prototype, and further test some ideas on IP scheduling that emerged in LCT-UC. The changes were made in a relatively short time, without too many difficulties, corroborating the easiness of installing new features on QoStat or changing existing ones. Figure 6 presents the main screen of the QoStat v0.1.2.
As can be seen, in this version of QoStat there is additional information in the *statistics rectangle* of the table tab. This information is related to the operation of the scheduler developed in our lab [5], namely:
- The degradation slope of the class, concerning the delay parameters - which determines its sensitivity to delay degradation;
- The reference delay - the maximum possible delay (in us) without impact on applications;
- The Xdelay - the delay each packet must wait on its queue until it becomes eligible for scheduling (this is an important parameter of our scheduler; because it is through it that we dynamically control the treatment given to each class);
- The IC, or congestion index - the value of the last calculated index (notice that the main goal of the scheduler is to maintain equal congestion indexes for all classes; this is continuously achieved changing the XDelay of the classes).

As can be seen, new tabs have been added to show graphically the evolution of the congestion indexes, and also, in an integrated manner, the evolution of all the other parameters independently shown on the other tabs.
Nevertheless, the majority of the changes appear in the vertical left zone of the screen. In the *Display Options* rectangle there are now buttons for choosing the unit of time values (clock cycles or microseconds) and the type of y-axe scale (normal or logarithmic). There is also a button to define if the congestion indexes should be calculated in system-space or user-space (which can be used to test the performance of both approaches).

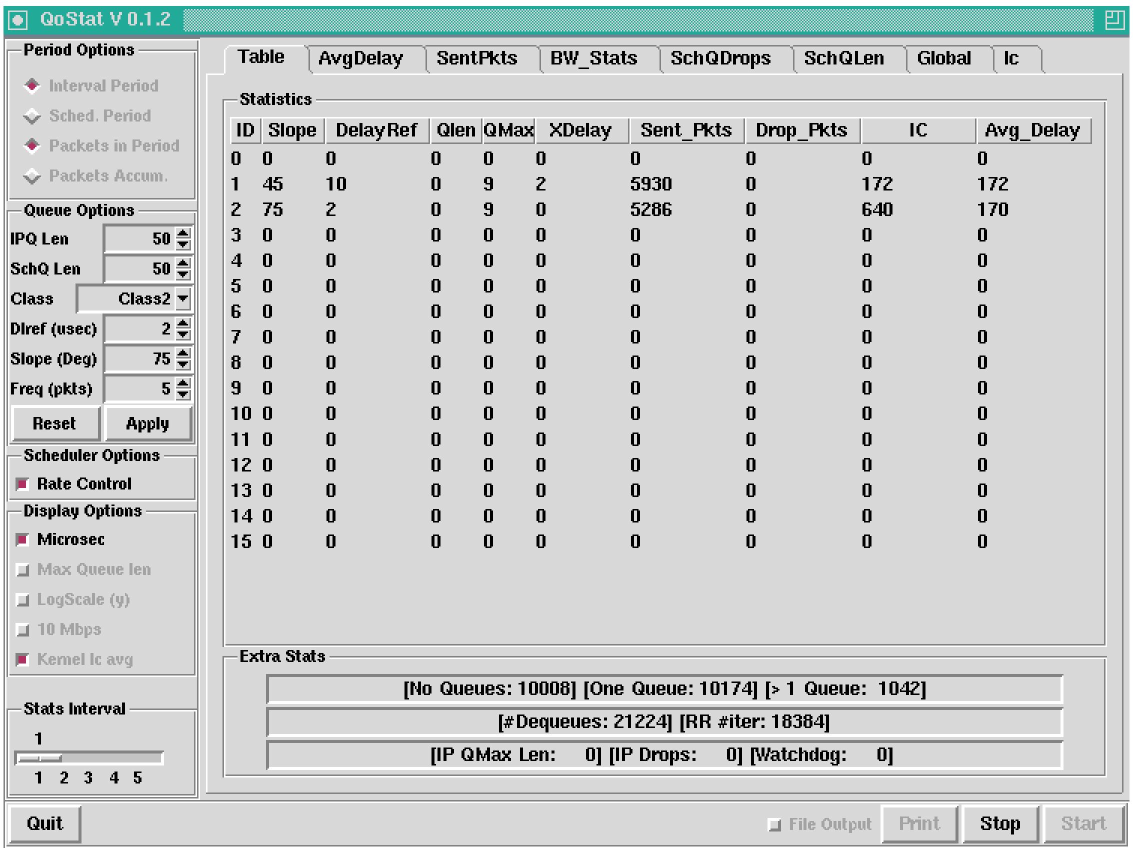

QoStat V 0.1.2

Table | AvgDelay | SentPkts | BW_Stats | SchQDrops | SchQLen | Global | Ic

Period Options
- [✓ Interval Period]
- Sched. Period
- [✓ Packets in Period]
- Packets Accum.

Queue Options
- IPQ Len: 50
- SchQ Len: 50
- Class: Class2
- Dlref (usec): 2
- Slope (Deg): 75
- Freq (pkts): 5

Reset | Apply

Scheduler Options
- [✓ Rate Control]

Display Options
- [✓ Microsec]
- Max Queue len
- LogScale (y)
- 10 Mbps
- [✓ Kernel lc avg]

Stats Interval: 1 (1 2 3 4 5)

Statistics

ID	Slope	DelayRef	Qlen	QMax	XDelay	Sent_Pkts	Drop_Pkts	IC	Avg_Delay
0	0	0	0	0	0	0	0	0	0
1	45	10	0	9	2	5930	0	172	172
2	75	2	0	9	0	5286	0	640	170
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0

Extra Stats

```
[No Queues: 10008] [One Queue: 10174] [> 1 Queue:  1042]
[#Dequeues: 21224] [RR #iter: 18384]
[IP QMax Len:     0] [IP Drops:     0] [Watchdog:     0]
```

Quit | File Output | Print | Stop | Start

Figure 6. Main QoStat screen with more functionality

In the *Queue Options* rectangle of this version it is possible to determine the degradation slope, and the delay of reference, for each queue. It is also possible to change the length of the IP input queue and, independently, of the IP output queues. Thus, this rectangle provides the means to a deep evaluation of the influence of these values on the scheduler behavior, or in the way traffic is in fact treated.
Lastly, we installed a *Period Options* rectangle in this QoStat version. It contains buttons to determine how the *Measurement Interval* is used in Average Delay calculations. It is possible to select only one of the *Interval Period* or *Sched Period* buttons at a time. In the first case, the average delay value is calculated averaging all the packets that were processed during MI. In the second case, the calculation is performed using only the last five packets processed in the interval. Basically, through those two buttons, we can change the time scale used to measure packet delay.
Through the other two buttons of the same rectangle the user can choose the nature of the presented values. They can reflect the activity in each *Measurement Interval*, or the accumulation of that activity since the beginning of the monitorization process.

4. SOME EXAMPLES OF QOSTAT UTILIZATION

The most important properties of our tool are the following:
- Capacity to monitor, in real time, QoS aware systems,
- Capacity to change on the fly some operational parameters related to QoS provision of the system being monitored.

These allow a better understanding of the behavior of the monitored system and a deeper knowledge about the influence of some of the system operational parameters on its behavior. Next, we present examples of some experiments that were positively impacted by the use of QoStat.
As told before, we are working on a new IP service model, which explores the IETF Differentiated Services approach. The idea is to provide QoS to classes without a substantial displacement from the technologies currently used on Internet. The first step is to conceive and implement a prototype of a QoS capable router with the right characteristics for our model.

In the scope of this project, we made several experiments with some IP scheduler mechanisms. We started with the WFQ/ALTQ scheduler. Because we wanted to develop a new dropper mechanism, in the first set of those tests we decided to deactivate the dropper mechanism distributed in the original WFQ/ALTQ implementation. Instead, we used a simple tail drop mechanism. We were surprised by the behavior of the system using that scheduler. QoStat was fundamental to understand the weakness of WFQ/ALTQ - namely its inability to differentiate traffic classes [4] - and to perceive that such weakness could in fact appear in any system that uses schedulers based on work-conserving disciplines.
In fact, we understood that the problem steams from the following situation: the dynamics of the operating system turns quite improbable that the scheduler finds packets in more than one queue simultaneously, even with very high loads. In such conditions, by definition, work-conserving disciplines cannot function. Using QoStat it was also easy to understand the influence of the maximum length of output queues on the behavior of the scheduler. Figure 7 clearly shows that, and exemplifies the usefulness of the tool.
With QoStat we could also conclude that the traffic differentiation capacity exhibited by the FreeBSD router patched with the WFQ/ALTQ technology, was due not to the influence of the WFQ discipline but as a consequence of the dropper used on that implementation [6].
Most relevant, the conception and erection of an original approach to forward packets on IP routers, to be used on the mentioned above project, was made, step by step, from the lessons learned while using the QoStat tool [5]. Currently, we are developing/testing a new dropper strategy, which is also being extensively supported by the use of QoStat.
The current QoStat distribution is based on the WFQ implementation of the ALTQ project (FreeBSD v 2.2.8 and ALTQ v 1.2). It should work with other versions of ALTQ, although some minor modifications may be necessary. It can be found in lct.dei.uc.pt and includes details and clues for the development, or adaptation, of the tool to different monitorization needs. As told above, we claim that these tasks of adaptation are straightforward, so it is quite simple to extend the use of QoStat to other QoS technologies.

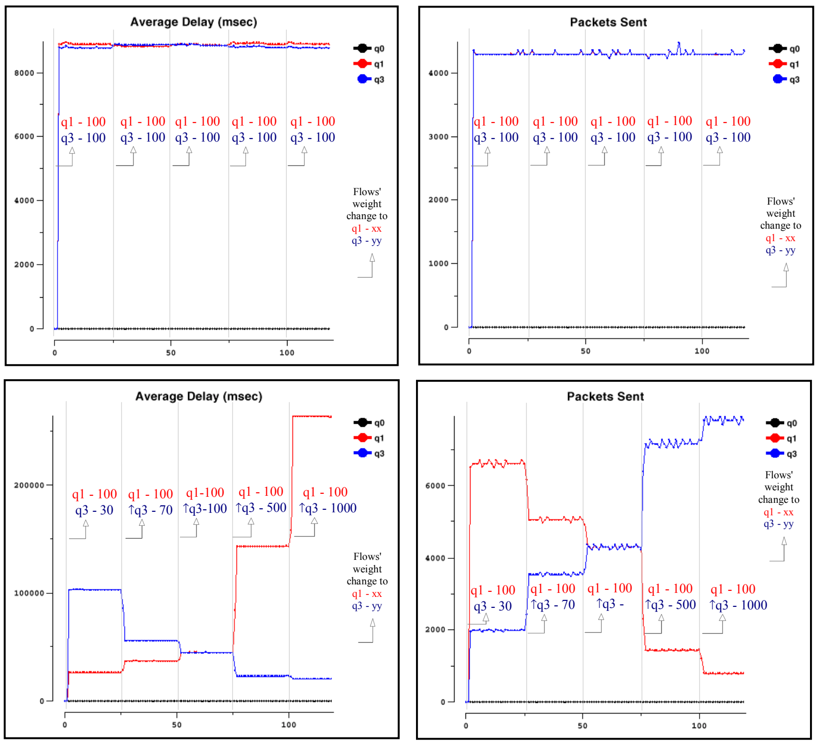


Figure 7. QoStat tests printouts (WFQ/ALTQ with simple tail-drop mechanism). Top: maximum queue length equal to 50 packets. Bottom: maximum queue length equal to 200 packets

5. CONCLUSION AND FUTURE WORK

In this paper we presented a tool developed at LCT-UC to monitor QoS capable IP FreeBSD routers.
This tool is able not only to evaluate in real-time the QoS behavior of the monitored systems, but also to change during the tests, on the fly, important parameters related to QoS provision.
Thus, with QoStat, it is possible to understand, with a good level of detail, the behavior of routers able to provide QoS. It is also possible to deeply study the exact influence on QoS of the different operational parameters of the mechanisms used on such routers. For instance the influence on QoS of the operational parameters of schedulers and droppers.
With this tool we were able to conduct several experiments, which taught us many important things related to QoS provision on communication systems and the dynamics of the underlying systems that support it- for instance, operating systems and protocol stacks. Those lessons were important in order to better understand the behavior of technologies such as WFQ/ALTQ and, most important, to develop a novel and efficient way to forward and drop IP packets on FreeBSD network elements.
We claim that QoStat is in fact very important to make research in IP QoS domain. Despite the impossibility to develop a generic tool, to be directly used with the different mechanisms used to provide QoS, we tried hard to develop a tool able to be adapted to the different possible conditions of use, without too much complexity. The basic distribution, which includes instructions for adaptation, can be found in http://lct.dei.uc.pt.

ACKNOWLEDGEMENTS

This work was partially supported by the Portuguese ministry of science and technology (MCT), under the program praxis XXI - PRAXIS/P/EEI/10168/1998 (Project QoS II - Quality of Service in Computer Communication Systems).

REFERENCES

1. 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/kjc/papers/usenix98.
2. Kenjiro Cho, Managing Traffic with ALTQ, in *Proceedings of USENIX 1999 Annual Technical Conference: FREENIX Track*, Monterey CA, June 1999. http://www.usenix.org/events/usenix99/technical_freenix.html
3. Antonio Alves, Goncalo Quadros, Edmundo Monteiro, Fernando Boavida, *QoStat – A Tool for the Evaluation of QoS Capable FreeBSD Routers*, Technical Report, CISUC, July 99.
4. Goncalo Quadros, António Alves, Edmundo Monteiro, Fernando Boavida, How Unfair can Weighted Fair Queuing be?, Proceedings of The Fifth IEEE Symposium on Computers and Communications (ISCC 2000), Antibes, France, 4-6 July 2000.
5. Goncalo Quadros, António Alves, Edmundo Monteiro, Fernando Boavida, An Effective Scheduler for IP Routers, Proceedings of The Fifth IEEE Symposium on Computers and Communications (ISCC 2000), Antibes, France, 4-6 July 2000.
6. Goncalo Quadros, António Alves, Edmundo Monteiro, Fernando Boavida, The role of packet-dropping mechanisms in QoS Differentiation, Technical Report, CISUC, October 99.
7. http://www.freebsd.org