An Effective Scheduler for IP Routers

Goncalo Quadros, Antonio Alves, Edmundo Monteiro, Fernando Boavida
CISUC – Centro de Informática e Sistemas da Universidade de Coimbra
Departamento de Engenharia Informática
{quadros, aalves, edmundo, boavida}@dei.uc.pt

Abstract

At the Communications and Telematics Laboratory of the University of Coimbra is being developed a router prototype with the aim to provide QoS to different traffic classes. One of the most important mechanisms of this router is the IP packet scheduler.
It is well known that the common scheduling discipline of current routers (first come first serve) turns them useless when QoS is needed - a different type of scheduler must be used. Our first idea to overcome this problem was to use a simple, open, and available scheduler, easy to adapt to the system we wanted to implement. We thought of the WFQ discipline, and, as we are using a testbed of Intel machines running FreeBSD OS, we admitted that the WFQ/ALTQ implementation would be an interesting choice.
Nevertheless, a broad set of tests carried out at our laboratory proved the contrary. Most important, these tests guided us to a deep knowledge about the problems, and causes, that can weaken the effectiveness of IP schedulers. Given the importance of that surplus information, we decided to implement our own scheduler. The idea was to take advantage of a most pragmatic view of scheduling activities to construct a scheduler with the best possible characteristics, but also very simple, thus, able to reach very good performance levels. This paper presents the scheduler that resulted from our attempts.
The proposed scheduler was subject to a set of tests that proved its ability to effectively differentiate traffic classes. The results of these tests are also presented and analyzed.

1. Introduction

This paper presents a new IP packet scheduler implemented at the Communications and Telematics Laboratory of the University of Coimbra (LCT-UC). This work was conducted to fulfill the requirements of a broader project which main goal is to develop a new service model for IP networks. This model should be able to provide different QoS to the various traffic classes, using essentially the same technologies as the ones currently used in the Internet.

A fundamental step for the development of the intended service model was the selection of an alternative for the common – but useless for this purpose – FIFO discipline used in routers.

As we are using Intel/FreeBSD platforms at LCT-UC, we decided to use the ALTQ technology [Cho98, Cho99] as an alternative for the IP queuing system. Given the architectural aspects of the model we are working on, we admitted that the WFQ discipline would be an interesting choice. Summing up, we decided to use the ALTQ implementation of WFQ in our project, and so we submitted it to tests. The results stood behind what we expected [Quadros99a], so this option was abandoned.

The lessons learned with the performed tests convinced us that the best approach to the development of the desired model was to construct our own scheduler. This paper presents this work. Section 2 details the proposed scheduler, its architecture and main operational characteristics. Section 3 presents the tests made to evaluate the scheduler behavior, whose results are discussed in section 4. Section 5 concludes this paper and positions it in the global LCT on-going project for the implementation of a new service model.

2. The Scheduler

2.1 Operational Principles

The basic idea behind the model is to differentiate the QoS given to classes[1] controlling the transit delay and the loss level provided to each one. As we want to avoid complexity, traffic specifications and explicit resource reservations are uninteresting. As in the current Internet paradigm, our model strategy is to forward as well as possible the data that reaches the IP level of routers and switches – but now considering the different nature of the handled data. In other words, in our model the statement "as well as possible" means different things according to the considered traffic class.

[1] The traffic is organized in different classes according to the types of applications.

Through transit delay we prioritize traffic, giving a relatively better treatment to some traffic at the expense of the other. Through losses, we can grossly control the assignment of communication resources to classes. Each class will have varying bandwidth requirements. For a given class, no loss means that the resources assigned to it are enough for its needs. Conversely, high loss levels mean those resources are scarce.

Nevertheless, we will not use the absolute measures of delay and losses to control the systems but, rather, relative measures. To understand the reason for this one can consider, for instance, that a delay of 50 ms for one particular application may be catastrophic while for others it can be meaningless. Thus, the taken approach uses a quantification of the impact on applications of the degradation of such QoS characteristics, taking into account their known normal values [Quadros99b]. Traffic classes typify this impact, which means that through the traffic class the impact of a given delay or loss degradation can be determined. We use a QoS metric developed at LCT-UC for the purpose of the quantification of the impact of QoS degradation on applications [Quadros98].

2.2 Why not use WFQ?

In short, in our model we redistribute communication resources in such a way that the impact of losses and delay, as measured by the referred metric, will be the same for all classes. One fundamental component to achieve this is the scheduler. Initially we thought that the ALTQ implementation of the WFQ discipline would be a good choice: it was simple, available, and seemed to fit very well in the model we wanted to develop. The tests, nevertheless, revealed important WFQ/ALTQ flaws [Quadros99a] and so we abandoned this idea.

It is worthwhile to mention the problem we found with our WFQ/ALTQ tests. Despite the load, most of the times the scheduler only finds packets in one of the queues (that is, packets belonging to a single class). Due to its work-conserving nature, the scheduler injects packets in the network at the maximum possible rate and thus – as it usually finds packets in only one queue – these packets will be processed without taking into account the class weight (there are no packets waiting in other classes). Therefore, despite eventual different weights assigned to each class, the treatment given to their packets is essentially the same.

For short, the WFQ discipline can only work if the correspondent scheduler finds packets in more than one queue. The system dynamics turns this situation rare – even with very favorable load conditions. Only by significantly raising the maximum output queue lengths (and so augmenting the probability of simultaneously having packets in more than one queue) the scheduler can reveal some QoS differentiation characteristics. In this case, nevertheless, we are also raising the transit delay beyond reasonable limits.

2.3 Scheduler Architecture

Therefore, we soon understood that WFQ disciplines can, in practice, be unable to differentiate traffic classes. The problem has to do with the work-conserving nature of the discipline. Thus, besides the advantages of this type of disciplines there are some important disadvantages. We also understood, like other researchers [Liebeherr99], that it makes sense to develop a scheduler able to pick the best part of the work-conserving and non-work-conserving worlds. Namely, (1) the simplicity and good level of communications resource utilization and (2) the capacity to reserve resources or to maintain some part of them available for high priority traffic.

We worked out this idea and developed the scheduler logically presented in the following figure. The scheduler behaves as a work-conserving one when processing packets from the highest priority queue, but as a non-work-conserving one when processing packets from the remainder queues.

To better understand our idea, we will first discuss the module 1 of the scheduler (see figure 1) and then module 2.

Module 1 corresponds to a work-conserving rate based scheduler. The scheduler visits the queues using a round robin discipline. Each queue has a DEQUEUE_TIME associated with it. When the visit happens after the queue's DEQUE_TIME, or when there is only a single queue with packets, the packet in the head of the queue is processed. When none of these two conditions occur, the scheduler goes to the next queue (until it finds one packet to process). Therefore, the scheduler processes packets at full speed whenever there is no competition for the transmitter, or when there are no packets in more than one queue.

X_DELAY is also an important queue's characteristic used to update the correspondent DEQUEUE_TIME. X_DELAY is the time interval (in µs) that should elapse between the processing of consecutive packets of the considered class. Through it, it is possible to effectively control the performance given to each class.

Module 2 makes the difference in our proposal. It guarantees that:

- For the queue with more stringent requirements (related with transit delay) the scheduler processes packets at full speed; it has a work-conserving behavior.

- For all the other queues, the scheduler processes packets at a fraction of the full speed, thus reserving some capacity for processing traffic that is more important; it has a non-work-conserving behavior.

The following points concisely present the scheduler operation:

- The scheduler visits each queue (the same is to say, each class) in a round robin fashion;

- In each visit, it compares the current TIME with the queue's DEQUEUE_TIME. If the former is greater than the latter, a packet is processed;

- For the most important class at a given moment (considering transit delay) X_DELAY is always zero; so the scheduler behaves as a work conserving one;

- For the remaining classes, the X_DELAY will be greater than zero and it will reflect the relative importance of the class; the scheduler behaves as a non-work-conserving one.

Therefore, even when the system dynamics turns rare the situations where there are packets in more than one queue, packets belonging to a class different from the most important one cannot monopolize the transmitter. Thus, even when that happens, the most important class will receive a better treatment than the next most important one, which will be better treated than the next, and so on.

As said before, we constructed our scheduler systematically integrating the lessons learned from the tests carried out on the WFQ/ALTQ scheduler. Its diagram, presented in figure 1, reflects this. In fact, this diagram could have been optimized, through the elimination of redundant paths (for instance, connecting point A directly to point B). We decided to present the scheduler this way to highlight its characteristics and the main decisions made during its implementation.

The logical level software calls the scheduler whenever a packet arrives or leaves the router. It exits returning one packet (there is only one exception, mentioned below, when the scheduler returns no packets). Before exiting, it updates its round robin pointer (RR) to the next queue in the ring. The following paragraphs discuss some decisions taken during the scheduler implementation (referencing some points marked in figure 1).

In position 1, the scheduler searches for the single queue with packets. As it has found a queue without packets with a past DEQUEUE_TIME, it updates this value (adding X_DELAY to the current time). The same happens in position 2. In this case there is more than one queue with packets, and the DEQUEUE_TIME of empty queues is also updated.

In position 3, the scheduler loops searching for a queue with packets. As can be seen there are no DEQUEUE_TIME updates in the loop. In fact, this is unnecessary because we assume that the most important queue has a X_DELAY equal to zero, which avoids an infinite loop.

In position 4 there is only one queue with packets, found before its DEQUEUE_TIME is past. This is the single case where the scheduler exits without returning a packet. In this particular situation we trigger a timeout. Without it, packets would be locked in the queue if there was no activity at the input and output interfaces. This is a reasonable approach because we assume that it should not happen often.

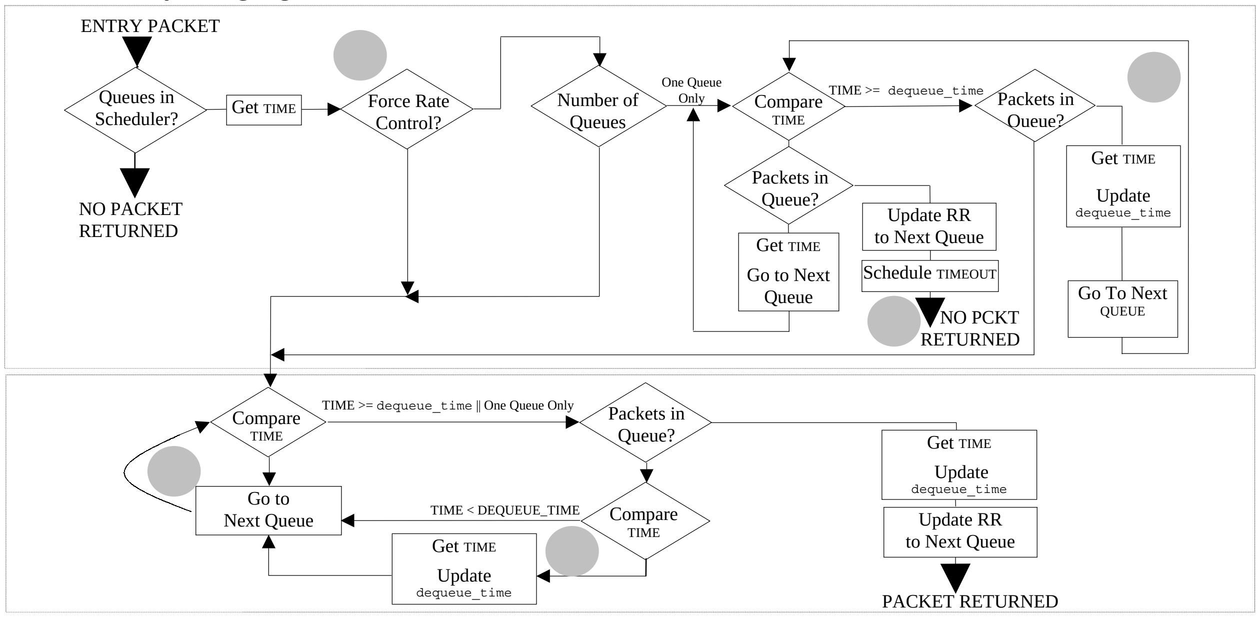

Figure 1 – Scheduler logical presentation

Lastly, we have introduced a switch to activate or deactivate scheduler module 2 (position 5) during the tests. Through it, we intent to show its usefulness.

In short, the scheduler presented here is conceptually very simple and provides effective means to control the performance given to different traffic classes. We intent to use it in the new IP service model we are developing at LCT-UC. With this scheduler, we will dynamically control the QoS (transit delay) for each class. The idea is to continuously measure the impact of transit delay on applications and, according to this impact, to dynamically modify the treatment given to the traffic - the same it to say, the X_DELAY associated with each class. The highest priority class, in a given time window, will have a null X_DELAY. The other ones will have an X_DELAY greater than zero, which will increase as the needs of the class decrease, in the considered time window. We will present soon, with more detail, this approach.

3. Test Environment

3.1 Goals

The main goals of the tests carried out on our scheduler were twofold:

- to evaluate its real capacity to differentiate traffic;

- to evaluate how well, and how easy, its possible to control the treatment given to different traffic.

We compared these results with the ones obtained for the WFQ/ALTQ, which was our first (and abandoned) scheduler choice for our model. The idea was to evaluate if, despite its simplicity, our scheduler could solve the flaws we identified in the WFQ/ALTQ case. Also because of this, the tests carried out on our scheduler followed the general principles established for the WFQ/ALTQ tests [Quadros99a].

The next section presents the testbed. Afterwards, we introduce the main test tools used.

3.2 The Testbed

Figure 2 presents the testbed used to perform the tests. 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 generate traffic directed towards HOST_D through HOST_R. Each host only generates traffic for a given class.

To perform broader tests than the ones carried out on WFQ/ALTQ scheduler, namely involving one more class, we have installed another generator host in the testbed.

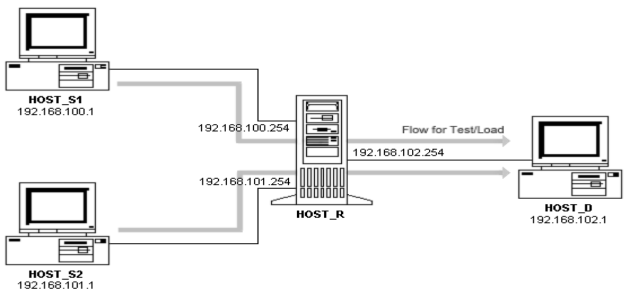

Figure 2 – The testbed

Moreover, to enable loads as high as possible, long packets (1400 bytes) constituted all the generated flows.

We used two tools to perform our tests:

- *Nttcp* [NTTCP]; to generate data flows able to compete for all the resources available in the communication system.

- *QoStat* [Alves99]; implemented in our laboratory and used to monitor the kernel, namely the operation of the installed scheduler. Through it, we can dynamically change the most important operational parameters of the scheduler. We can also obtain, in real time, some of the most important figures related to QoS provision. *QoStat* produced all the graphics presented in the present paper.

Figure 3 presents a logical scheme of the tested system. Notice that the dropper mechanism is a simple tail-drop (we are now working on a different approach for this mechanism, which will be incorporate in our work in the near future).

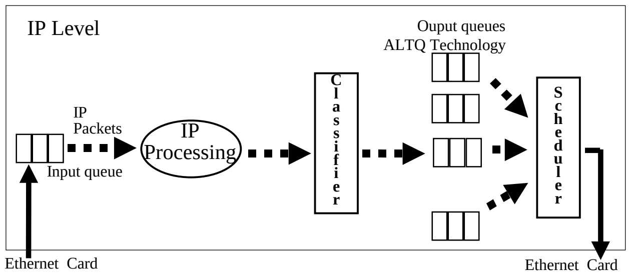

Figure 3 – Prototype logical architecture

4. Tests Results and their Analysis

We exclusively used UDP traffic and a simple tail-drop mechanism for the tests presented here. The general test strategy was the following:

- to use two different classes, composed of traffic generated by two independent hosts;

- to set the X_DELAY associated with class 1[2] to zero,

- to vary the X_DELAY associated with class 3.

[2] Whose queue is named q1 in the figures.

In the first set of tests, we used the default ALTQ configuration, which means maximum queue length of 50 packets. The X_DELAY values chosen for class 3 were successively 0, 10, 15, 50, 200. We used the *qostat* tool to dynamically change those values at the end of each 25 seconds time interval[3].

The values shown in the graphs are:

- the average transit delay of packets at IP level, measured over 1-second time intervals. A packet transit delay is the time that elapses between its enqueuing at IP input queue, and its dequeuing from the IP output queue (see figure 3);

- the number of packets sent by the output interface per 1-second interval;

- the number of dropped packets per 1-second interval;

- the instantaneous IP output queue length sampled every second.

As can be seen in figure 4, it is evident that for the referred conditions our scheduler is able to differentiate traffic. This contrasts with the behavior of the WFQ/ALTQ scheduler, which, for maximum queue lengths of 50 packets, does not have this capacity - as shown in figure 5 [Quadros99a]. This last figure presents the result of a test carried out on the WFQ scheduler implementation of the ALTQ project, using the same load conditions as before. In that test we assigned the weight 100 to one of the classes and successively the weight 30, 70, 100, 500 and 1000 to the other.

Our scheduler is able to benefit one class at the expense of the other. Taking as reference the situation where both classes are equally important, it is evident that for the class with growing X_DELAY the average transit delay and the number of packets sent per unit of time becomes worst. Conversely, for the other class these figures become better.

Figure 6 shows the importance of the scheduler module 2. The test presented in this figure is the same as the previously discussed one, but now we divided each 25 seconds time interval with fixed X_DELAY in two. On the first part we activated module 2, on the second one we deactivated it. It is clear that, without the action of this module the scheduler is not able to differentiate traffic.

Therefore, it is useful to guarantee that when there are packets on only one queue, other than the most important one, they can not use all the communication resources available. Notice that this is not a pure non-work-conserving behavior. In fact, the most important of the active classes in the system can always use all the available communication resources.

Until now we have been presenting tests carried out using maximum output queue lengths of 50 packets. This is the default value for FreeBSD systems and ALTQ technology. We referred that the WFQ/ALTQ does not behave as expected for such a maximum length (it needs bigger queues). Conversely, we showed here that our scheduler does behave well with this maximum length.

Nevertheless, to admit 50 packets in each queue is not reasonable. This is obvious if we recall that in current routers the space available on the single output queue is more or less the same value. So 50 packets should be the maximum space available on all the output queues, not in a single one. The importance of this is not related to memory scarcity or price, but to the maximum transit delay routers might introduce. Thus, independently of the final value to define for the maximum space that should exist in output queues on these new routers, 50 packets per class seems too high. In this case, in a system with only 4 classes, we could have 200 queued packets waiting for their opportunity to be processed, which is too much.

Given this, it does not suffice to demonstrate that the scheduler is able to work well with maximum output queues of 50 packets. It must behave correctly with shorter queues as well. Therefore, we repeated the tests with maximum output queue lengths of only 20 packets. Again, we fixed the X_DELAY value associated with class 1 to zero. The one associated with class 3 was set initially to zero, and afterwards successively to 3000, 5000, 10000 and 20000. The results presented in figure 7 prove that, despite the shorter queues, the scheduler still exhibits a quite good capacity to give different treatment to different classes.

Given the good results obtained, we decided to test the scheduler with more flows. So we joined a new host to the testbed – to increase the independence of the generated traffic – and we made some more tests.

Figure 8 presents a test carried out with a maximum output queue length of 50 packets. Each host generated a 45 Mbps flow, composed of 1400 bytes packets, associated to one exclusive class. Initially, we assigned a null X_DELAY to all the traffic classes. This value never changed for class 3. At the 25th second we changed the X_DELAY of classes 1 and 2 to 7500 and 10000 respectively, with the scheduler module 2 disabled. At the 50th second, we enabled the module, maintaining the X_DELAY values. At the 75th second we exchanged class 1 and 2 X_DELAYs, maintaining the module 2 enabled. At the 100th second we disabled module 2.

It is clear that the scheduler is able to give different performance levels to the flows, for all the measured QoS characteristics– transit delay, drops and losses. Having as reference the starting situation, when all classes have a null X_DELAY, we can see that its possible to give a better treatment to some classes at the expense of the others. Thus, the redistribution of resources produced by changing X_DELAY is effective – it has a consistent

[3] We didn't use class 2 in the tests. As we are only using traffic of classes 1 and 3 the measured values of class 0 are always null.

reflection on transit delay, losses and packets sent for each class.

Lastly, it is clear that for this particular situation the operation of module 2 is not relevant. In fact, no substantial difference comes out from figure 8. Comparing, for instance, the intervals [25s 50s] and [50s 75s] we can, nevertheless, say that a more stable situation is expectable with the operation of the module.

The tests presented in figure 9 are the repetition of the ones just referred with a single difference: we set the maximum length of the output queues to 20 packets. Despite the shorter queues, the scheduler still reveals capacity to treat differently the classes of traffic.

However, the differences introduced by module 2 operation are now evident. In fact, without the module, we can see that for the number of dropped or sent packets different classes are equally treated. For the average transit delay module 2 is also useful. Without it, the scheduler introduces additional delay on classes 1 and 2 traffic, without any positive impact on class 3 delay. On the other hand, with the module, the most important class gets an improvement on transit delay, which nevertheless is very small and corresponds to a strong penalization of the other classes.

5. Conclusion and Future Work

At the Communications and Telematics Laboratory of the University of Coimbra we have been developing a new scheduler for IP routers. Our main motivation was to solve some flaws we found when we tested the WFQ implementation of the ALTQ project.

Those tests revealed some weaknesses in terms of the WFQ/ALTQ capacity to differentiate the performance given to different traffic classes. From the lessons learned during their execution, we were able to conceive and develop a new scheduler, with a very simple architecture but, nevertheless, able to differentiate traffic in a way we could control.

This paper presents the developed scheduler and some tests that show its effectiveness. It reveals a good behavior even under those situations where we found weaknesses in the WFQ/ALTQ operation – for instance, when there is much less space in output queues.

Our idea is to use this scheduler with the QoS metric conceived in LCT-UC. By integrating the scheduler and the metric, we expect to construct a router with an effective traffic differentiation capacity. Our intention is to continuously measure the QoS provided to each class, and to dynamically redistribute the communication resources according to the measured quality of service [Quadros99b]. In the near future, we are going to use a different dropping mechanism with our scheduler that we are now implementing at LCT-UC. With it, we expect to enhance the characteristics of the proposed scheduler, namely its ability to deal with TCP traffic.

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.

References

[Alves99] A Alves, G Quadros, E Monteiro, F Boavida, *QoStat – A Tool for the Evaluation of QoS Capable FreeBSD Routers*, Technical Report, CISUC, July 99.

[Cho98] K. 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. www.csl.sony.co.jp/person/kjc/kjc/papers/usenix98

[Cho99] Kenjiro Cho, Managing Traffic with ALTQ, in *Proceedings of USENIX 1999 Annual Technical Conference: FREENIX Track*, Monterey CA, June 1999. www.usenix.org/events/usenix99/technical_freenix.html

[Liebeherr99] Jorg Liebeherr, Erhan Yilmaz, Workconserving vs. Non-Workconserving Packet Scheduling: An Issue Revisited, in *Proceedings of IWQoS'99*, London, May 31-June 4, 1999.

[NTTCP] New TTCP Program, www.leo.org/~bartel/nttcp/

[Quadros98] G Quadros, E Monteiro, F Boavida, "A QoS Metric for Packet Networks", Proceedings of SPIE International Symposium on Voice, Video, and Data Communications, Conference 3529A, Hynes Convention Center, Boston, Massachusetts, USA, 1-5 November 98.

[Quadros99a] G Quadros, A Alves, E Monteiro, Fernando Boavida, *Analysis of the WFQ Implementation of the ALTQ Project*, Technical Report, CISUC, July 99.

[Quadros99b] G Quadros, A Alves, E Monteiro, F Boavida, "Approach to the Dynamic Forwarding of Packets in a Differentiated Service Based Router", Proceedings of SPIES's symposium on Voice, Video, and Data Communications conference on QoS Issues Related to Internet II, Boston, MA, USA, September 19-22, 99.

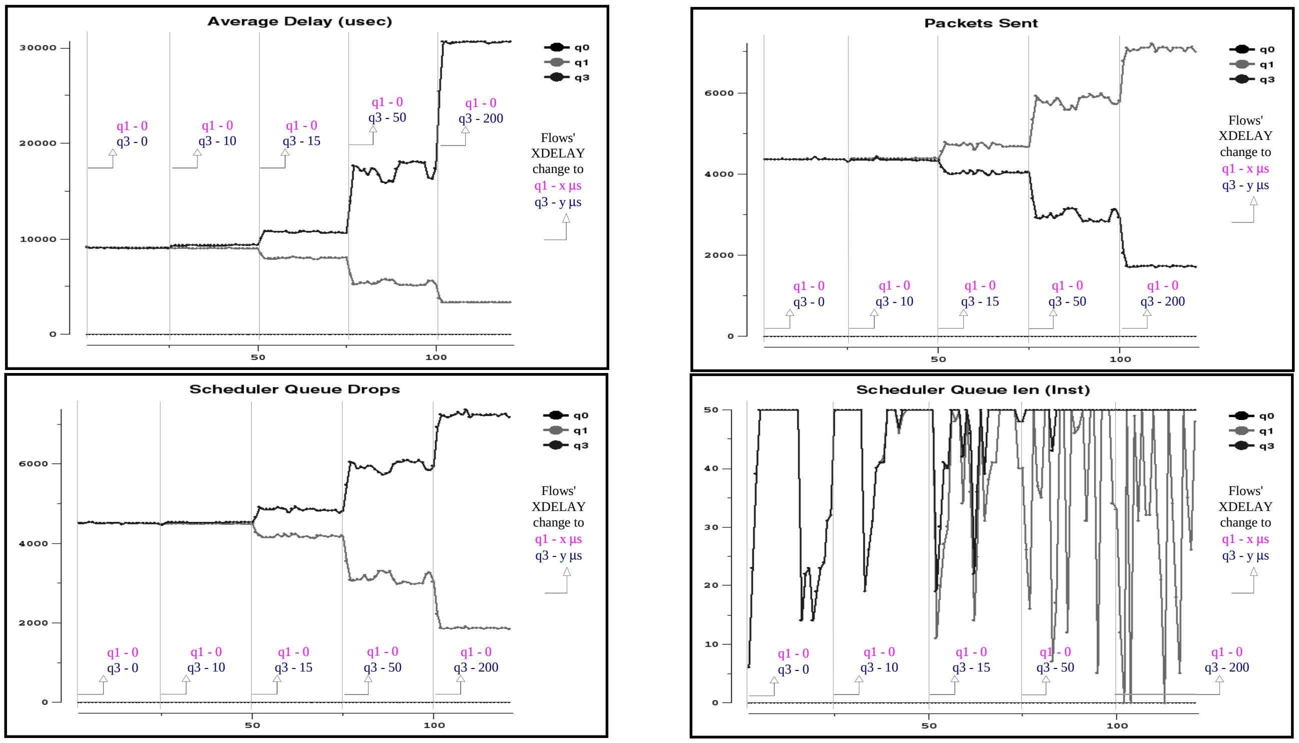

Figure 4 – Tests made to the proposed scheduler with maximum output queue lengths equal to 50 pcks. Variation of the following values with X_DELAY: ↖) Average IP transit delay (over 1sec. time intervals); ↗) Number of pcks sent per sec.; ↙) Number of dropped pcks per second; ↘) Instantaneous IP output queue length (sampled every sec.).

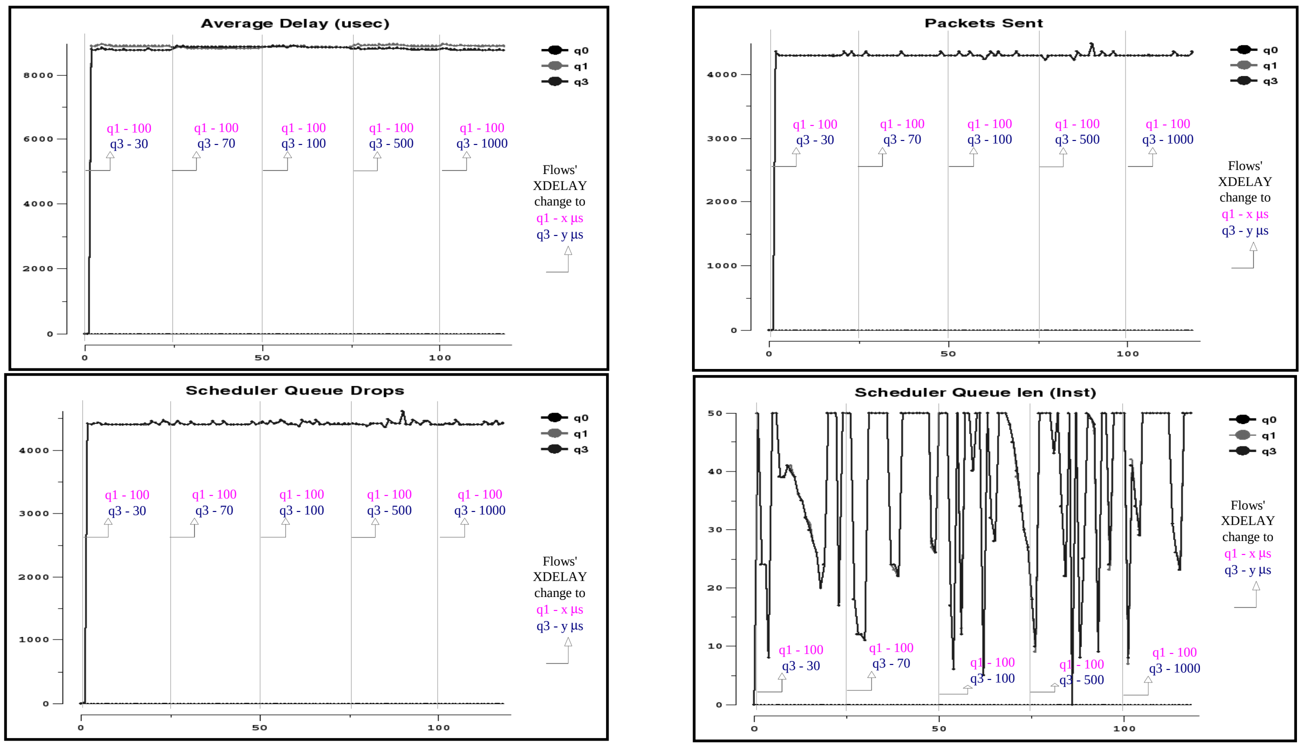

Figure 5 – WFQ/ALTQ tests: variation of the following values with class weight: ↖) Average IP transit delay (over 1 second intervals); ↗) Number of packets sent per second; ↙) Number of dropped packets per second; ↘) Instantaneous IP output queue length (sampled every second)

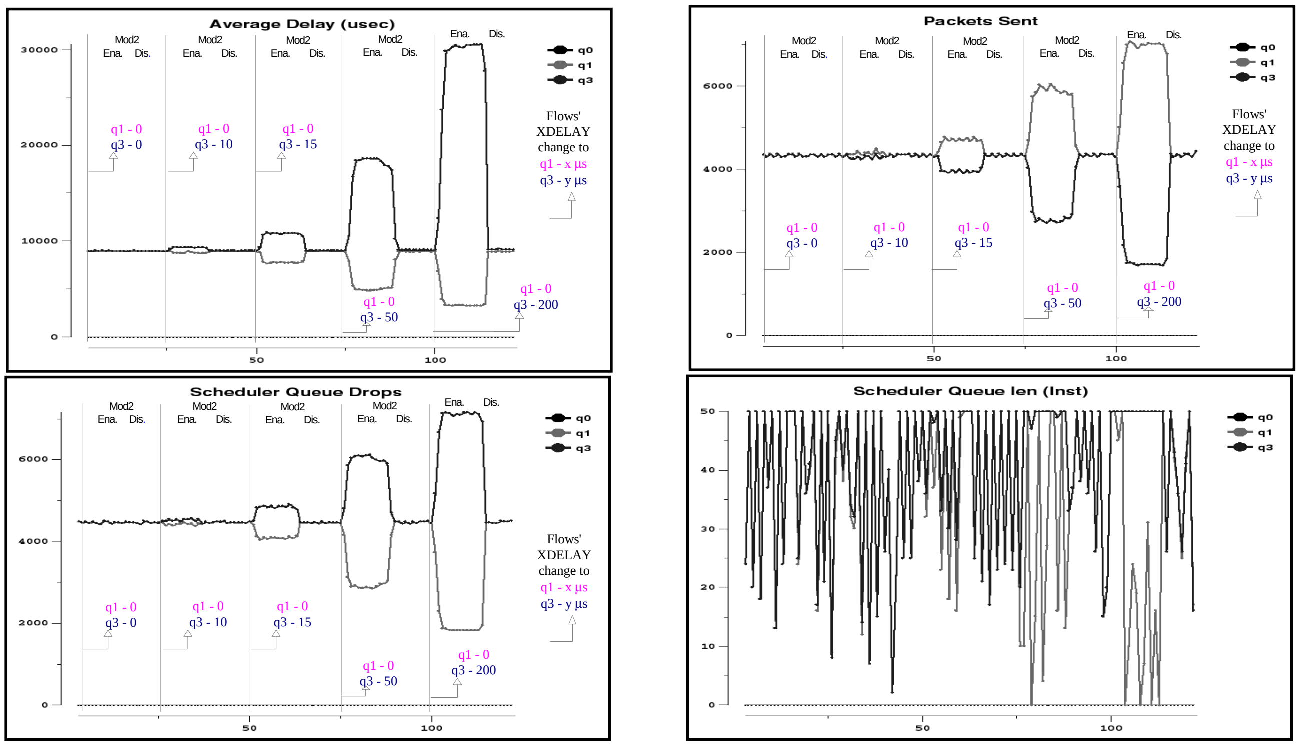

Figure 6 – Tests made to the scheduler with max output queue length equal to 50 pckts, activating and deactivating its module 2. Variation of the following values with X_DELAY: ↖) Average IP transit delay (over 1-sec intervals); ↗) Number of pckts sent per sec; ↙) Number of dropped pckts per sec; ↘) Instantaneous IP output queue length (sampled every sec).

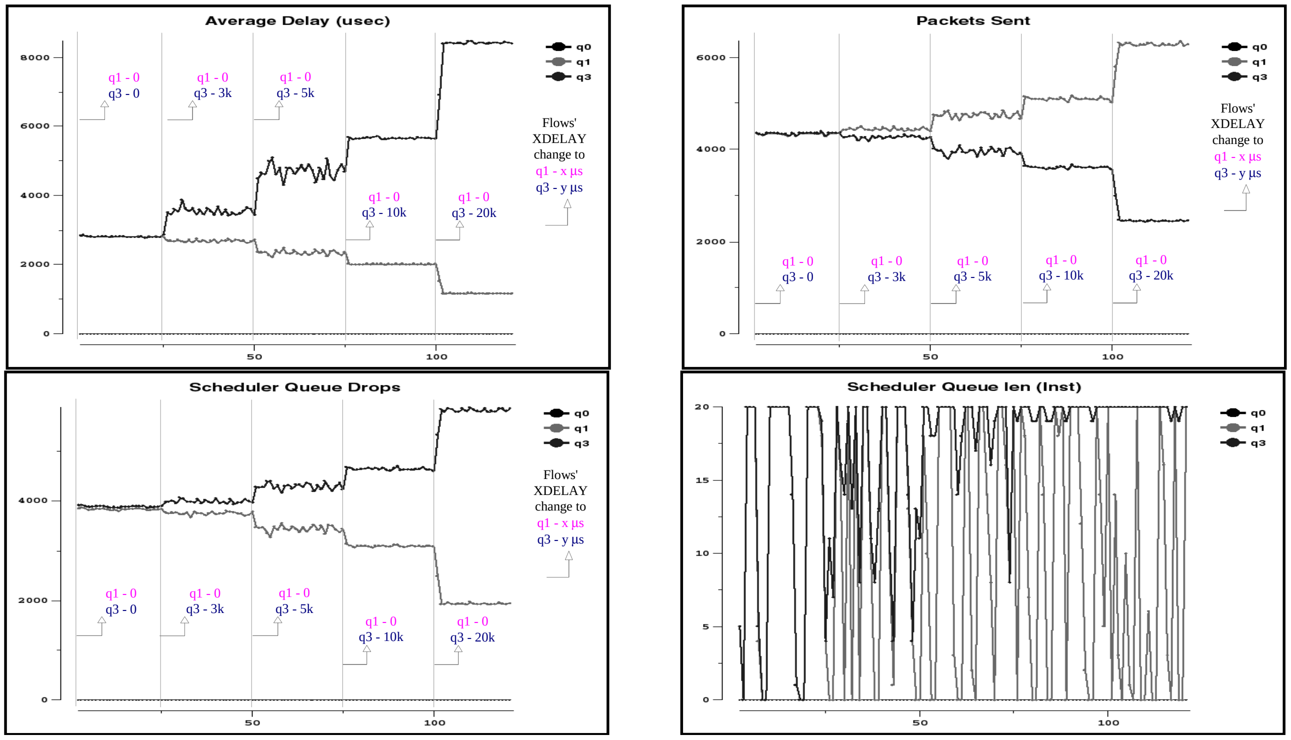

Figure 7 – Tests made to the scheduler with maximum output queue length equal to 20 pcks. Variation of the following values with X_DELAY: ↖) Average IP transit delay (over 1 second intervals); ↗) Number of packets sent per second; ↙) Number of dropped packets per second; ↘) Instantaneous IP output queue length (sampled every second).

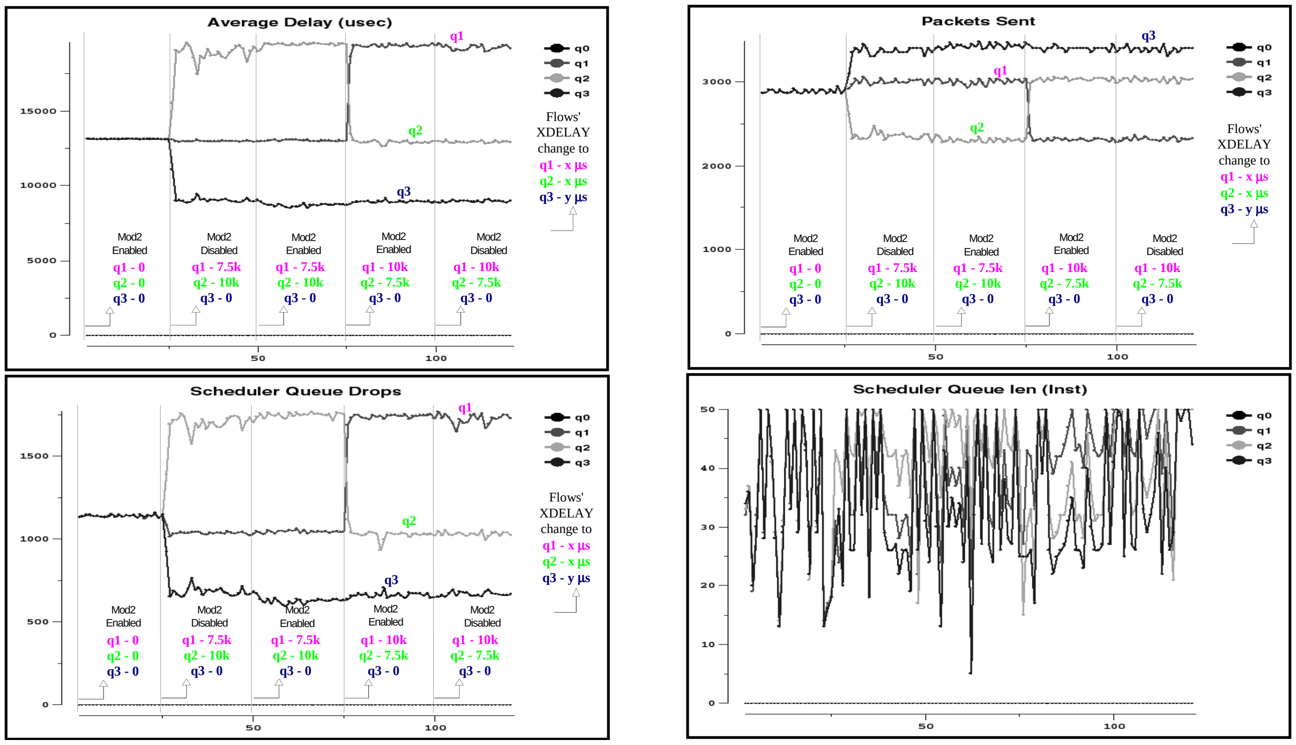

Figure 8 – Tests made to the scheduler with maximum output queue length equal to 50 pcks. Variation of the following values with X_DELAY: ↖) Average IP transit delay (over 1 second intervals); ↗) Number of packets sent per second; ↙) Number of dropped packets per second; ↘) Instantaneous IP output queue length (sampled every second).

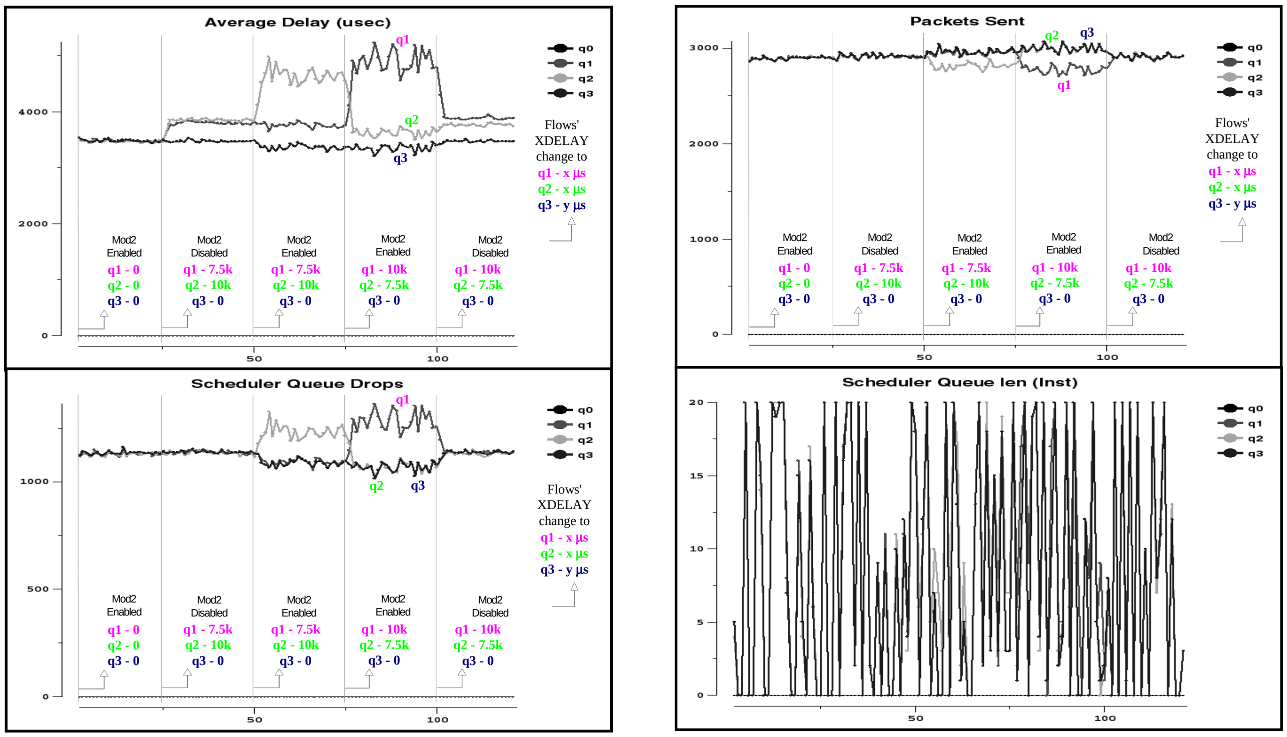

Figure 9 – Tests made to the scheduler with maximum output queue length equal to 20 pcks. Variation of the following values with X_DELAY: ↖) Average IP transit delay (over 1 second intervals); ↗) Number of packets sent per second; ↙) Number of dropped packets per second; ↘) Instantaneous IP output queue length (sampled every second).