Behavior and Performance of QoS Mechanisms on Different Router OS

David FUIN, Eric GARCIA, Hervé GUYENNET

LIFC, Université de Franche-Comté
16, route de GRAY
25030 BESANÇON CEDEX, FRANCE
tel: 33 3 81 66 20 92, fax: 33 3 81 66 64 50
{fuin,garcia,guyennet}@lifc.univ-fcomte.fr

ABSTRACT

In order to allow the CAliF QoS service to parameterize the various elements of the network according to applications needs, it is necessary to know the real behavior of all QoS mechanisms available on network elements.

In this paper we present the results of QoS mechanism performance analysis which are necessary to design coherent QoS rules. We present in particular a comparison between Cisco and Linux QoS mechanisms. We monitor, in particular, evolution of internal subqueues of routers in order to know the real behavior of QoS mechanisms. These results are used to create a consistent and efficient policy repository that is adapted to the type of application to be developed and that makes our service efficient.

KEYWORDS

Quality of Service, Control Mechanisms, Performance, Router queue.

1 Introduction

To improve the development of cooperative work applications, we designed a Multimedia platform named CAliF (Cooperative AppLIcation Framework) . CAliF supports the design of cooperative multimedia applications, while freeing the user from various issues pertaining to cooperative member communication, data consistency management and continuous media broadcasting (video, voice). This platform is developed on top of CORBA which manages the problems of heterogeneity, interoperability, portability and access to resources. It is composed of four services: a group communication management service, a shared data consistency management service, a management service for the broadcasting of continuous media and finally a quality of service (QoS) management service. The subject of this paper is to present this last service, as well as the analysis of many QoS mechanisms on Cisco routers and Linux router to parameterize this service.

The CAliF QoS service is based on a QoS request administrator and uses a policy repository to parameterize the various elements of the network according to the application behavior by using various QoS control mechanisms. The IP environment has become enriched with the publication of many QoS control techniques, but it is difficult to know their actual behavior. It is thus important to propose and to functionally analyze new solutions like IntServ or DiffServ.

In this context, we have made many tests (most curves can be seen at `http://lifc.univ-fcomte.fr/~fuin/Recherche/`) on these mechanisms in order to create coherent rules allowing our service to work effectively. Indeed, the publications on the subject have often merely described their theoretical functioning without providing any real experimentation, then, we also propose the comparison between Cisco Mechanisms and Linux ones. Thanks to tools such as SNMP and DBS we have studied the effect of QoS mechanisms on flow in term of throughput and in term of Routers internal behavior at subqueues level and CPU level.

In the first part of this paper, we present our QoS service architecture, which allows us to automate the configuration of the various network elements using the Policy-Based management principle [1]. In the second part, we study the performance of the various QoS management mechanisms to analyze their results in order to help our service users with the design of the QoS rules. In the last part, we present our future work: the use of these analyzes in order to be able to make SNMP trap signal sender in case of congestion.

2 QoS Service Presentation

We have already presented the limitations of static QoS administration, policy architecture and existing platforms for QoS Management as well as their limitations in [2].

2.1 CAliF Multimedia QoS Service

2.1.1 Target

The target of our QoS service is to offer multimedia cooperative applications an end-to-end communication system that can guarantee a given process. The CAliF Multimedia platform QoS service allows the linking of all the applications and the network. It must be capable of parameterizing the routers in order to satisfy applications but also to ask applications to reconsider their needs and adapt to a new configuration. We then obtain a more flexible management model enabling us to mix the reservation of network resources and the adaptation of application needs. Such a service allows very various admission policies, as, for instance, priority-based admission control.

2.1.2 Architecture

We therefore chose to develop a service that is capable of automating the quality of service management. We were inspired by IETF architecture for QoS Policy-based Management [1] but we also used the resource adaptation principle that allows us to consider the use of network mechanisms on active elements and application adaptation.

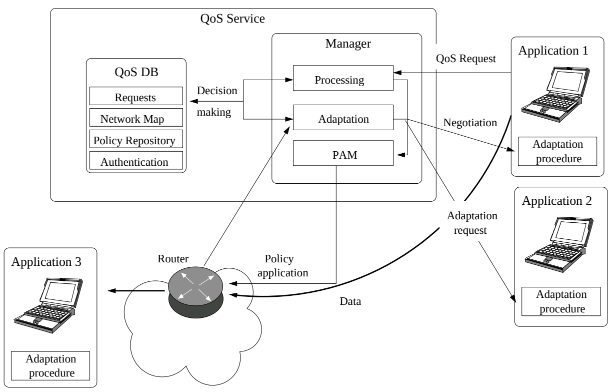

Figure 1: QoS service components

The QoS service has two main elements (Fig. 1):

- a QoS database containing information concerning the global state of the system, it is composed of four elements:
  - the request repository: it contains the list of the application quality of service requests;
  - the network map: it allows the user to know the reservations on the routers;
  - the authentication server: it contains the list of permissions and reservation quota;
  - the policy repository: it helps us to decide which policy to apply according to information in the request repository, the network map, and the authentication server.
- a manager which receives QoS requests from applications. It is composed of three modules:
  - The Policy Application Module or PAM that corresponds to the delocalisation of a PEP of the IETF architecture. It is responsible for parameterizing associated quality of service mechanisms resulting from policy decision on the routers;
  - The processing module: it receives application requests, translates them and uses information in the QoS database to send a decision to the PAM;
  - The adaptation module: it allows us to ask for the adaptation of application needs if the network resources change or if the request for a new application require fewer reservations, or in case of quality of service negotiation.

3 Quality of Service Mechanism Performance

There is a great number of QoS control mechanisms on IP networks [3]. Moreover, all routers don't use the same mechanisms. It is thus indispensable to study the actual behavior on different routers of these multiple quality of service mechanisms to be able to decide which application of a particular policy will best meet with the different application requirements.

3.1 Test description

3.1.1 Test platforms

For those tests, we use two different platforms: one in order to test Cisco mechanisms and another one to test Linux ones.

For the Cisco test platform (figure 2), we used Cisco 1605-R as routers. They are equipped with two Ethernet interfaces used to connect them to different machine (*M1* and *M2* are connected with router 1 and *M3* with router 2) and with a WAN interface (interface *Serial 0*) which allows us to connect them together.

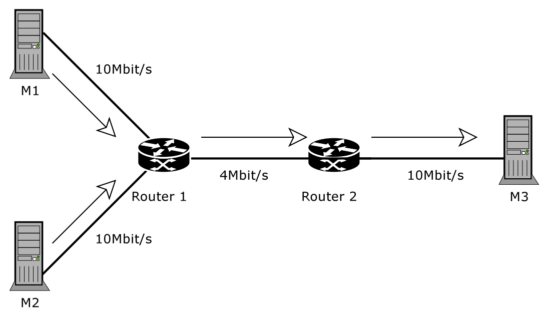

Figure 2: Test platform

For the Linux test platform (figure 2), we use two identical PCs running Debian Linux v3.0r1 (upgraded to unstable version) with a kernel 2.4.20 as routers. They are equipped with three Ethernet interfaces. We use the first two Ethernet card to connect them to different machine (*M1* and *M2* are connected with router 1 and *M3* with router 2) and the third card allows us to connect them together.

In both platform, the connections between machines and routers allow us to reach a throughput of 10 Mb/s while the connection between both routers is limited to 4 Mb/s. This makes it possible to create a congestion point between router 1 and 2. Congestion are situated at the output interface of the first met router: interface *Serial 0* of router 1 (in case of Cisco Routers) in the case of rival broadcast of data from *M1* and *M2* to *M3*. It is thus on this interface that we shall apply the quality of service mechanisms which will be in charge of processing special streams preferentially.

Our Cisco router operating system is the Cisco IOS (*Internetwork Operating System*) 12.0 IP Plus. Lower versions do not supply the means to exploit quality of service mechanisms.

3.1.2 QoS Router Architecture

In order to understand curves we obtain, it is necessary to understand a QoS router architecture.

When a packet arrives at an ingress interface (figure 3), it cross the first module 1 where packets are classified according to their characteristics (IPs, ports, protocols...). The next task is the detection of packets that are not conforms to a contract: those packets are often discarded. Others packets will be routed 2 toward an egress interface. Such packets will then enter the queues manager and packets scheduler module 3.

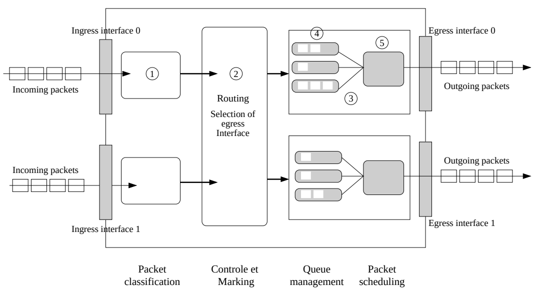

Figure 3: QoS Router Architecture

This is where the QoS will take place, indeed, packets will be put into queues according to the way they have been classified. Then, the packet scheduler determines the way each queue is dequeued toward their egress interface. For example, if packets are dequeued more often from one queue than others, packets from the first queue will have a statically higher priority than other packets.

QoS mechanism	Main queue size	Number of subqueues	Length of subqueues
FIFO	4	1	40
WFQ	2	256	64
PQ	2	4	20,40,60,80
CQ	2	16	20

Table 1: Cisco 1605-R routers queues default

So, the way packets are classified, put into queues and dequeued in a particular order makes effective a particular QoS.

There are a certain number of subqueues 4 per egress interface and one main queue 5. Packets dequeued from subqueues are placed in the hardware queue from where they are sent to the egress interface.

For example, In Cisco 1605-R routers, the number of subqueues and their size depend on the QoS mechanisms currently in use (table 1 shows Cisco 1605-R default). Some QoS mechanisms permit to change the length and the number of subqueues.

3.1.3 SNMP monitoring

In order to be able to monitor routers subqueues, we took Cisco Router Private MIB from Cisco web site. This allows us to query Cisco router for internal system values.

For example, this OID: `CISCO-QUEUE-MIB::cQStatsDepth.3.7` allows us to know the current number of messages in the seventh subqueue of `Serial0 interface` (third interface). Another example is `CISCO-QUEUE-MIB::cQStatsDiscards.1.2` allowing us to know the number of discarded messages of the FIFO subqueues (the unique subqueue is number 2 if the mechanism attached to the interface is FIFO) of the first interface: `Eth0`. This allows us to monitor the real internal behavior of QoS mechanisms.

Moreover, SNMP also allows us to monitor the CPU usage of routers in order to be able to recognize mechanisms that require a high CPU usage.

3.1.4 Experiment description

To test these mechanisms, we create traffic while modifying our router configurations. We use the DBS [4] (*Distributed Benchmark System*) software which allows us to create various TCP or UDP traffic and to introduce any sorts of delays between the packet broadcasts.

Thanks to DBS, we measure throughputs of flows (at receiver) we generate. In order to analyze the real behavior of QoS mechanisms we also monitor internal subqueues of routers. Indeed, thanks to SNMP we monitor the number of messages in each subqueue and the number of messages that have been discarded in each subqueue.

The curves we present in this section represent:

- the throughputs (ordinate) according to time (abscissa) for various configurations of quality of service control mechanisms on the routers.
- the number of messages in used subqueues (ordinate) according to time (abscissa) for the same configuration.
- the total number of discarded messages in used subqueues (ordinate) according to time (abscissa) for the same configuration.
- the number of discarded messages in used subqueues (ordinate) according to time (abscissa) for the same configuration.

The created traffic corresponds to the rival broadcast of a TCP stream and another TCP stream or a TCP stream and a MPEG stream or a MPEG stream and TCP stream from both machines situated upstream to router 1 (M1 and M2) to the machine situated downstream to router 2 (M3).

We have tested many quality of service control mechanisms [5, 6] (PQ, CQ, WFQ, PRIO, SFQ, CBQ...) on IP routers. These mechanisms were tested with various parameter settings and on different router OS (Cisco IOS and Linux). We have done a great number of tests. All these results can be consulted at the following address: `http://lifc.univ-fcomte.fr/~fuin/Recherche/`

3.2 Performance

We present here some of the mains tests which allow us to verify that it is possible to ensure a certain quality of service on IP networks thanks to Cisco routers and Linux routers. We notice that **the real behavior of mechanisms on routers is not exactly what it should be in theory**.

3.2.1 Strong priority with PQ and PRIO

For these tests, we choose to experiment strong priority algorithms. Cisco Implementation is called Priority Queueing (PQ) whereas Linux one is called PRIO.

Such mechanisms allow us to give different priorities to flows in order to improve delay and bandwidth of some flows to the detriment of others. Flows are placed in queues (according to policy). Whenever a packet needs to be dequeued, the first queue (the one with the highest priority) is tried first. Lower queues are only used if all higher queues are empty.

Cisco implementation uses four queues while Linux one uses as much as needed. In all tests experimenting strong priority algorithms, we use two flows (one with a higher priority than the other).

We saw that these implementations are good and that the priority is always satisfied. This allows to prioritize some flows (for example interactive flows from application such as telnet) in order to prevent high-bandwidth applications (such as ftp) to decrease others critical applications.

However, this queueing discipline can completely block some traffic in the case where higher priority flows consume all bandwidth.

3.2.2 Fair queueing with WFQ and SFQ

For these tests, we choose to experiment fair-queueing algorithms. Cisco implementation is called Weighted Fair Queueing (WFQ) whereas Linux one is called Stochastic Fairness Queueing (SFQ). On Cisco routers, if WFQ is used without signaling mechanisms (such as RSVP), WFQ subqueues have no weight. In this case, WFQ has the same behavior as a fair queueing algorithm.

Such mechanisms allow us to share equitably the bandwidth between flows. Each flow is placed in a different queue. Traffic is then sent in a round robin way in order to give to each flow an equal chance to send data and disallow any single conversation from consuming all bandwidth.

Cisco implementation uses up to 4096 subqueues while Linux uses up to 1024 subqueues. In both implementation, information to divide traffic are based on source IP, destination IP, source port, destination port, protocol, DSCP field... However, algorithms to divide traffic and to dequeue packets are quite different.

SFQ is designed for high-speed routers (routers that must be able to handle a very high number of packets per seconds), indeed, it consumes less CPU and memory than WFQ [7]. One of the major differences is that SFQ uses a simple hash function to assign flows to subqueues while WFQ use a more complex assignment function. The result is that SFQ is faster but not as fair as WFQ.

We saw that experiments with WFQ as well as SFQ give a good distribution of bandwidth even with a high bandwidth consuming MPEG flow. Moreover, we can see on SFQ curve the periodically change of the Linux SFQ assignment function.

If two flows must be put into the same subqueue (for example because the hash function assign two different flows to the same subqueues or there are more flows than subqueues...), these flows will share as much bandwidth than other single flow. The Linux SFQ assignment function changes periodically in order to prevent that the same flows are always being handicapped.

3.2.3 Bandwidth Reservation with RSVP, HTB and CBQ

For these tests, we choose to experiment bandwidth reservation algorithms. On Cisco routers we use RSVP (ReSource reSerVation Protocol) that must be associated to one of these congestion managers: RED (Random Early Detection) or WFQ. On Linux routers, we can choose CBQ (Class Based Queueing) or more recently HTB (Hierarchical Token Bucket).

These mechanisms permit the reservation of bandwidth and allocation of flows to these reservations. RSVP/RED, RSVP/WFQ and HTB have a similar behavior: packets from flows are dropped according to their reservation (the less reserved bandwidth they have, the more they are dropped). CBQ has another behavior and is far more complex to setup. From [6] we can read: "It should work like this. If you try to shape a 10mbit/s connection to 1mbit/s, the link should be idle 90% of the time. If it isn't, we need to throttle so that it IS idle 90% of the time."

We saw that experiments with HTB are very good: reservations are well respected. Experiment with RSVP/RED, RSVP/WFQ and CBQ are quite good although flows can exceed a bit their reservations. **CBQ parameters have to be adjusted in order to have the wished behavior**. This is due to the fact that two different 10mbit/s network card have not always the same real performance. So the parameters may need to be adjusted.

3.2.4 Statistics Priority with CQ, DiffServ and GRED

For these tests, we choose to experiment statistics priority or weak priority (in opposition to strong priority seen in section 3.2.1). On Cisco we use CQ (Custom Queueing), DiffServ/RED or DiffServ/WFQ while on Linux we use GRED (Generalized Random Early Detection).

These mechanisms allow us to give some packets a better probability to cross routers than other packets. DiffServ/RED, DiffServ/WFQ and GRED have a similar behavior: in case of congestion, packets are dropped according to their priority. The lower the priority is, the greater the probability of destroying packets is. CQ has another behavior: when a subqueue is dequeued, the number of dequeued packets is determined according to the `byte-count` of the queue. The `byte-count` of each subqueue can be set according to the wished QoS. CQ uses 4 queues, DiffServ use 8 preconfigured priorities and GRED uses up to 16 queues.

We saw that the 8 preconfigured priority DiffServ/-RED and DiffServ/WFQ permit to easily have a good distribution of flows with a well-respected priority. CQ allows to choose the priority of each subqueue but there are only 4 subqueues, moreover, the `byte-count`, that permit to choose the number of byte to transmit, is round upper packets (in order to transmit a whole packet). This means that the result strongly depends of the packet size. We also saw that in order to give a high priority to a flow thanks to GRED, the difference between the parameters must be important (such as 0.001 vs 0.5).

3.3 Traffic Shaping with GTS, TBF and HTB

For these tests, we choose to experiment traffic shaping. On Cisco we use GTS (Generic Traffic Shaping) while on Linux we use TBF (Token Bucket Filter) and HTB seen in section 3.2.3 about bandwidth reservation.

These mechanisms allow us to limit the bandwidth usage of some flows. TBF permits only to limit the total bandwidth usage while GTS and HTB allow us to set the limit for a particular flow or group of flows. All these mechanisms use a Token Bucket algorithm: packets are dequeued at the same rate than token arrives in the bucket. Setting the frequency of incoming token set the rate of packets. TBF uses only one token bucket algorithm for all flows while GTS and HTB use one token bucket per bandwidth limited group of flows.

On the one hand, we saw that **TBF applies a lower limitation than the parameters set**: for example, in figure 4, we set a 4Mbit/s bandwidth limitation and we obtain a 3.5Mbit/s bandwidth limit. So, these parameters must be adjusted. On the other hand, we saw that both GTS and HTB limitations are quite precise, so, it is easy to set a bandwidth limitation for example in order to limit the bandwidth usage of UDP flows.

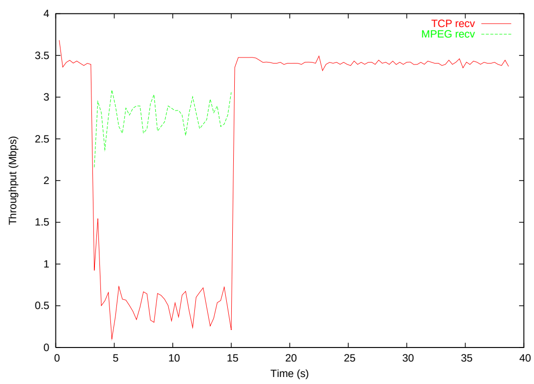

Figure 4: TBF TCPvsMPEG Throughput.

3.4 SNMP monitoring

As an example, if we compare WFQ (figure 6) to FIFO (figure 5) with two TCP flows (WFQ and FIFO with Cisco default parameters), we can see interesting results: in both case, the bandwidth is shared equitably (the probability of dropping packets from the first flow is the same than from the second flow). But in the case of WFQ, bandwidth of both flows are quite stable.

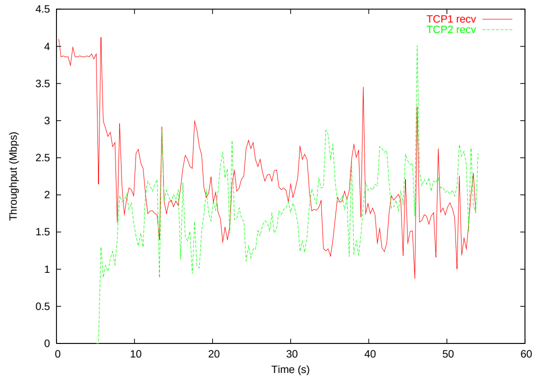

Figure 5: FIFO TCPvsTCP Throughput.

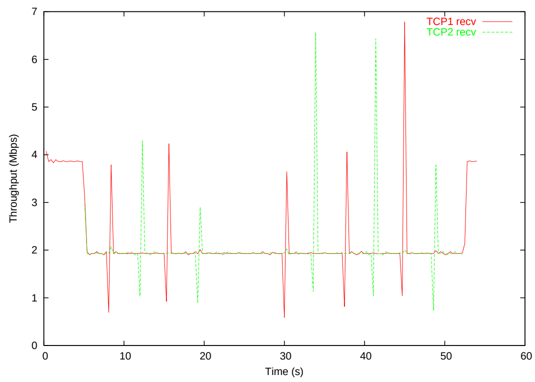

Figure 6: WFQ16 TCPvsTCP Throughput.

From SNMP monitoring, if we compare the number of packets lost, we can see that WFQ needs to drop less packets (13 for the first flow and 11 for the second in our tests figure 8) than FIFO (45 for the sum of both flows figure 7) to share the bandwidth between the two flows.

This is due in particular to the number of messages the subqueues can handle: indeed, in the case of FIFO, the flows share a unique subqueue that contain about 35 messages (figure 9) during the congestion, in the case of WFQ, each flow has is subqueue that contain about 30 messages (figure 10) during the congestion. We can increase the size of FIFO subqueue but this will increase delay transmission too.

We can conclude that with TCP flows, WFQ is not only fairer than FIFO, it also permits to discard fewer packets (so fewer packets are to be sent again).

On an other experiment about WFQ, we can see after congestion caused by a MPEG flow and a TCP flow (with a throughput limited to 2.6Mbit/s) that when the MPEG flow is over, the TCP flows use all bandwidth

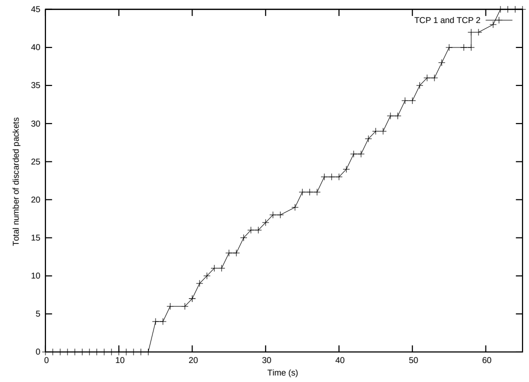

Figure 7: FIFO TCPvsTCP Total number of discarded packets.

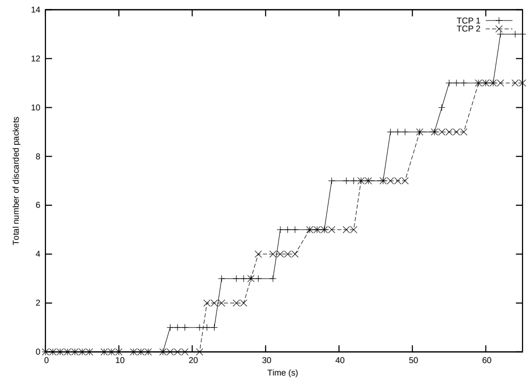

Figure 8: WFQ16 TCPvsTCP Total number of discarded packets.

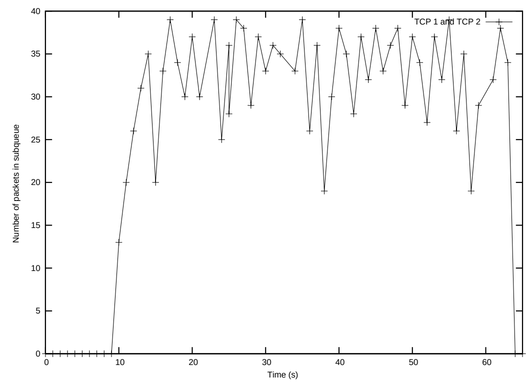

Figure 9: FIFO TCPvsTCP Number of packets in subqueue.

available while his subqueues are emptying (figure 11 and 12).

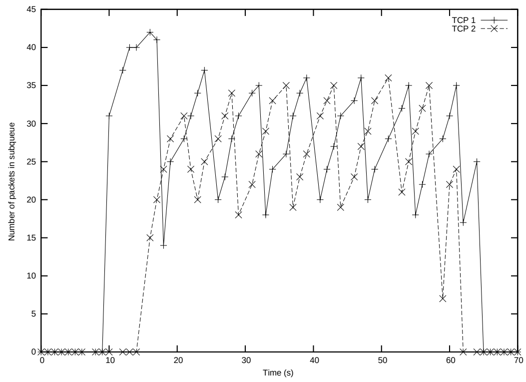

Figure 10: WFQ16 TCPvsTCP Number of packets in subqueues.

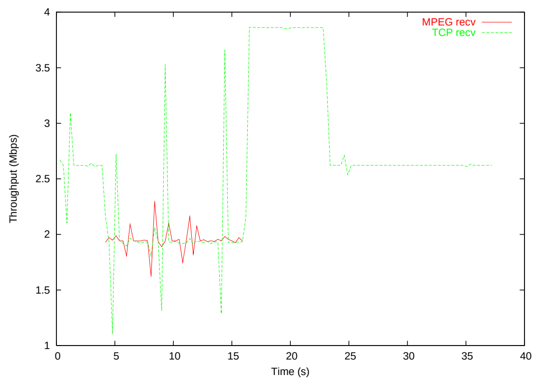

Figure 11: FIFO TCPvsTCP Throughput.

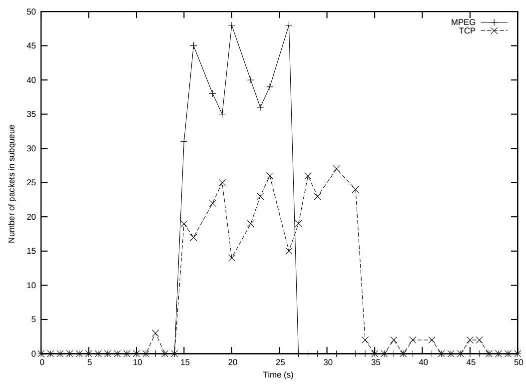

Figure 12: WFQ16 MPEGvsTCP Number of packets in subqueues.

3.5 More differences between Cisco and Linux implementations.

We have seen that Cisco and Linux don't always propose the same mechanisms, this is not really a problem since they propose algorithms that have similar behavior.

Moreover, thanks to QoS architecture such as DiffServ, we can set the DSCP field [8] of flows at the border of the network in order to homogenize the behavior of routers: for each value of the DSCP field corresponds a similar behavior of Cisco routers and Linux routers (even if they don't use the same algorithm).

Beyond that, there are major differences between QoS thanks to Cisco 1605-R routers and QoS thanks to Linux routers:

- about level of configuration: Most of the time, on the one hand, Cisco implementations of QoS algorithms propose only few options to parameterize: this simplifies the configuration of these routers. For example, when you use `IP Precedence` of packet header to choose QoS to apply to each packet and that you use WFQ as QoS mechanisms, weights are automatically selected. On the other hand, Linux implementations propose more options to parameterize: this allows us to better customize the router behavior.

- about mechanisms combination: All mechanisms presented in this paper are quite basic: for example, if we use a Strong Priority mechanism and put a TCP flow and an UDP flow in the same priority, in case of congestion, when packets from these flows will be discarded, only TCP flow will reduce his throughput. Thus, UDP will use more bandwidth.

  In order to prevent this situation, on Linux routers, we can easily combine PRIO with SFQ in order to make all flows from the same priority share equally the bandwidth. Moreover, on Linux routers, we can combine most mechanisms together; this is not the case of Cisco routers.

- about hardware: On a Linux router, we can have a lot of memory and a fast CPU so we can use more complex mechanisms (or combination of mechanisms). We can also increase the size of the subqueues. However we must care to not increase to much delay nor jitter in particular in case of multimedia flow or Real-Time flow where packets that arrives too late are useless.

  The relatively slow CPU may explain differences we saw in [2] between the real behavior of mechanisms on routers and what it should be in theory.

Cisco Routers mechanisms are often easier to setup while Linux ones allow more parameters to be set in order to best meet users requirement.

4 Future work

We want to use these analysis (in particular the monitoring of subqueues thanks to SNMP) in order to be able to make SNMP trap in case of congestion.

We propose that, thanks to internal monitoring of routers, we process an SNMP trap when the number of messages in subqueues increases quickly. This will inform us that packets will start to be discarded very soon.

These traps will be sent to the manager of our QoS service (presented in section 2) in order to reconfigure the QoS of network elements (for example, in order to increase the bandwidth reservation of some critical flows). Moreover, the manager of our QoS service will also be able to ask applications to adapt [9] their flows (for example, decrease the size of a video flow).

5 Conclusion

In this paper, we present quickly the CAliF (Cooperative AppLIcation Framework) QoS service: the service of our multimedia cooperative platform that manages QoS. We see that this service uses a policy repository to parameterize the various network elements. In order to improve the design of this policy repository, we must know the real behavior of QoS mechanisms available on routers.

For this reason, we have made many tests on these QoS mechanisms. They have provided many curves (available at `http://lifc.univ-fcomte.fr/~fuin/Recherche/`). We have also monitored internal behavior of routers (mainly Router subqueues). We have studied the effect of QoS mechanisms on TCP and UDP flows in term of bandwidth on different OS (Cisco IOS and Linux).

We have concluded that the real behavior of some mechanisms on routers is not exactly what it should be in theory. Moreover, we have identified compatible mechanisms on Cisco routers and Linux routers that make possible the use of our Policy repository with heterogeneous Routers.

References

[1] R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy-based Admission Control," in *IETF Network Working Group, Request for Comments 2753*, January 2000.

[2] E. Garcia, H. Guyennet, and JC. Lapayre, "Qos control mechanism evaluation to design a reliable policy repository," in *Proceedings of the 5th IEEE International Conference on High-Speed Networks and Multimedia Communications HSNMC'02, pp 300-304*, July 2002.

[3] A. Stephenson, "QoS: The IP Solution, Delivering End-to-End Quality of Service for the Future of IP," in *Lucent Technologies, WAN System Group, White paper 93-1001452-03*, December 1999.

[4] Y. Murayama and S. Yamaguchi, "DBS: Distributed Benchmark System," in *Nara Institute of Science and Technology, Japan*, 1998.

[5] Cisco Systems, "Cisco IOS Release 12.0 Quality of Service Solutions Configuration Guide," in *http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/*, 2001.

[6] Bert Hubert, "Linux Advanced Routing & Traffic Control," in *http://lartc.org/lartc.html#AEN935*, July 2003.

[7] Paul E. McKenney, "Stochastic fairness queuing," in *IEEE INFOCOM*, June 1990.

[8] K. Nichols, S. Blake, F. Baker, and D. Blake, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," in *RFC 2474*, December 1998.

[9] B. Goldschmidt and Z. László, "A Proxy Placement Algorithm for the Adaptative Multimedia Server," in *Euro-Par 2003, LNCS 2790, pp. 1199-1206*, 2003.