Review of Traffic Scheduler Features on General Purpose Platforms

Eduardo Magaña, Edurne Izkue y Jesús Villadangos
Universidad Pública de Navarra
Departamento de Automática y Computación
Campus de Arrosadía, 31006 Pamplona, SPAIN
Tel. +34 948 169853
eduardo.magana@unavarra.es, edurne@izkue.com, jesusv@unavarra.es

ABSTRACT

In this study, we review the features of a traffic scheduler running on a general-purpose platform, specifically a PC with a Linux operating system. The traffic scheduler was configured to manage a CBQ (Class Based Queueing) queue discipline, and by means of a series of experiments, we proved that the limitations of the system are due to the accuracy of the clock. After we modified this factor, we reviewed the features of the scheduler again. We were able to prove that the features of the scheduler improve greatly at the expense of a very small increase (around 10%) in the use of the system's CPU.

Keywords

Traffic Scheduler, Quality of Service, DiffServ, CBQ, TBF, Linux, congestion window, accuracy of clock.

1. INTRODUCTION

Data networks transport packets belonging to different types of services. One of the usual criteria is to classify them according to their time requirements. In such cases, services are differentiated by those who have very strict time requirements, called real time services, and other services. An example of the first type of services is voice transmission over IP (VoIP), and video on demand (VoD), whereas e-mail and file transfer using FTP are examples of the second type. In addition, the quality of real time services is very sensitive to packet loss, since this can cause interruptions in reproducing the information.

In short, providing real time, quality, and attractive services to the user, necessarily implies that the provider has to guarantee a sufficient quality of service for these services, trying to avoid the loss of packets, minimizing delays and delay variations (*jitter*).

At present, the Internet does not guarantee any quality of service, since it is based on a Best Effort mechanism. This means that the flow transmitting the greater amount of packets has a greater chance of seeing its packets reach their destination. IP, the Internet protocol, does not guarantee as such, any quality of service.

The idea of being able to guarantee quality of service has led to considering the need of a traffic scheduler following the differentiated services (DiffServ) proposal of the IETF [2].

In this article, we study the possibilities of using a general-purpose architecture as a scheduling platform. This reflects the situation of an Internet Service Provider (ISP) who wants to provide real time services with a guarantee of adequate quality of service. One of the main features of ISPs is that they usually contract a lower bandwidth than could be required by all their subscribers simultaneously in order to maximize benefits. This fact can bring about situations in which real time services are affected by the use of network resources by the remaining users. Therefore, ISPs must use scheduling systems to limit the rate of users and to guarantee a minimum rate to real time services.

We have tried to observe the greatest amount of different details produced by the use of a scheduler, and we show the results in the following sections. First, in Section 2 we present the CBQ and TBF queue disciplines used. In Section 3, we describe the environment of the experiment. In Section 4, we show the effect of scheduling when the sizes of the packets and the times between arrivals follow different distributions. Next, Section 5 shows the effects of the scheduler on UDP and TCP flows. Section 6 is dedicated to showing the effects of modifying the accuracy of the system clock. In this section, we analyze the improvements achieved with an increased accuracy of the clock and the computing cost this implies. Finally, in Section 7, we present related papers and in Section 8, the conclusions obtained.

SIGCOMM – Latin America & Caribbean 4/01 San Jose, Costa Rica

© 2001 ACM ISBN 1- 58113-354-5/01/0004…$5.00

2. TRAFFIC CONTROL IN LINUX

The Linux operating system incorporates in recent kernel versions traffic control support that provides the basic functionalities of queue disciplines, classes and filters, in order to be able to implement a quality of service scheduler. The Linux traffic control system is outlined in Figure 1 [5].

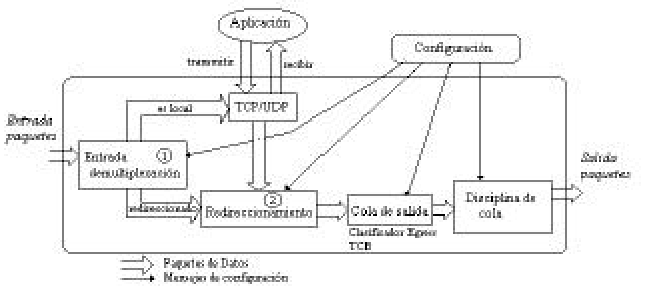

Figure 1. Outline of operation of the network section of the LINUX kernel.

The operating system processes can transmit and receive packets making use of the kernel level network functionalities. One of these functionalities is in charge of traffic control in Linux. Figure 1 shows its block structure.

The demultiplexation input ① examines the arriving packets and determines whether the packet is for the local machine or if it needs to be redirected to the following machine. If the packet is for the local machine, it is sent to the upper layer to be processed. If not, it is passed on to the redirection block ②. This redirection block ② also receives packets to be transmitted by the network generated by the local machine in upper layers.

Redirection ② checks the routing tables, looks for the following stage, selects the proper output interface for this route and encapsulates the packet. Once all this is done, and after determining the destination node, it stores the packet in the output queue, that is the proper output buffer for this service, route or type of information, so it can be transmitted as soon as possible, after accessing the network (which depends on the network topology; for example, if it is 802.2 Ethernet it shall be contained by the CSMA/CD protocol, and if it is Token Ring it will wait to take the token). Moreover, this is where the Linux traffic control acts. Among others things, it decides if the packet is put in a queue or if it is discarded (for example, because the queues have a limited length and are already full, or because traffic exceeds some transmission bit rate limit). The traffic control can decide in which order the packets are sent, giving priority, for example, to certain flows over others, and can delay the transmission of packets, for example, limiting the network output traffic bit rate with a Token Leaky Bucket. The traffic control Linux offers can be used to build complex combinations of queues disciplines, classes and filters which are useful to apply QoS disciplines on packets sent to some output interface.

In the output interface a shaper device can also be configured, meaning a traffic molder that determines the type of traffic leaving the network. In addition, an explicit routing can be made by labeling packets with the ipchains command [8]. The output interface queue follows a queue discipline such as CBQ, FIFO, a rate limiting type like TBF (Token Bucket Filter), etc.

At the base of the operation of the Linux traffic control are three main blocks:

- *Qdisc*: Queue discipline, which would be the queue collecting the packets and taking them out according to the chosen discipline. There is at least one in the output interface. A queue discipline can be specified for each final class (where packets end up once they are classified).

- *Class*: The class determines the type of traffic. More concretely, it says what type of service or what classification corresponds to the packet in the scheduler, so these packets picked up by the scheduler can receive the proper treatment later. Each class has its own features related to the type of queue discipline one wishes to associate to it.

- *Classifiers or filters*: The filter or classifier is where packet discrimination criterion is determined, the origin and/or destination addresses, the type of information carried (according to the origin and/or destination port), or other fields of the headers, with full flexibility.

These three blocks are the fundamental base to understanding the principles behind a traffic controller in Linux. Management of bandwidth is hierarchical, which means that classes follow a hierarchy that bears relation to the bandwidth to be determined in each class.

There can only be one root *qdisc* per device. This *qdisc* is associated with the output device, which owns the full bandwidth offered by it.

We will now survey the queue disciplines used in this paper: CBQ and TBF.

2.1 CBQ Type Scheduler

CBQ (*Class Based Queuing*) [6] [7] [16] is a traffic control mechanism based on sharing the bandwidth of a link in order to maintain resources. It is characterized by enabling a link to be shared among multiple agents, protocol families, or types of traffic.

It offers a hierarchical structure for every link, where classes can be found, which is the corresponding structure to some type of traffic aggregate. The hierarchical structure itself specifies the desired policy for this link, expressed in bandwidth percentages during periods of congestion. Reservation of bandwidth can be static (assigned by the manager), or dynamic (using a protocol for reservation of resources as, for example, RSVP [3]). All packets going through the router are associated with a class, which is a leaf of the tree in the hierarchical structure, such as nodes D, E, B, or C in Figure 2. The intermediate nodes of the tree do no transmit any packet, as could be the case of node A in Figure 2. These nodes only provide information on how to associate the surplus bandwidth and keep statistics of the nodes hanging from them, since the packets these leaf nodes transmit go through them.

We should stress as the main objective of CBQ that a class receives at least the bandwidth associated to it, even in situations of congestion. Therefore, if it is a class with a high demand it receives at least the associated bandwidth, and can receive more if the rules so permit, as will be explained further on. If a class has no associated bandwidth, CBQ does not guarantee any bandwidth in the event of congestions. The associated bandwidth depends on the scheduling mechanism for packets of the router, as we will see further on. A proper selection of associated bandwidths makes the scheduler more useful from the user's point of view.

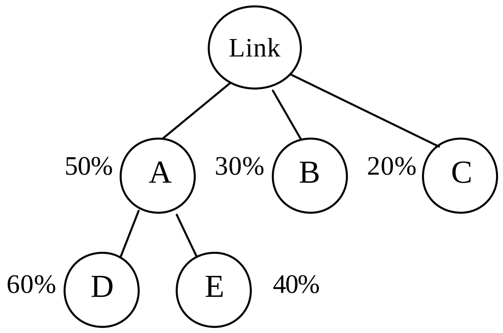

Figure 2. Example of configuration with CBQ.

Another important feature of CBQ is bandwidth borrowing. Redistribution of bandwidth not used by some class is not arbitrary. Taking as an example the configuration in Figure 2, Node D must take at least 60% of the associated bandwidth belonging to Node A, and Node E 40%. The indication of a minimum bandwidth is in order to enable a class to take the surplus bandwidth from the upper class from which it hangs. So if Node D were exceeding the associated bandwidth, which means it needs more than 60% of the bandwidth associated to A, its father, and this Node gave it permission, it would take the surplus associated bandwidth in order to cover the lack of bandwidth of D. Node E would have the surplus bandwidth of Node A. If usage of the bandwidth of Nodes E and D were not above the bandwidth of Node A, this surplus bandwidth would be shared out according to percentages with Nodes C and D if the upper node or link permits this.

Every class can have an associated router output queue discipline. The only class that must have an output queue discipline is the root, which is linked directly to the output network card. In every class two packet schedulers are defined, generic and link sharing:

- The generic scheduler is in charge of transmitting packets when the bandwidth required by the flow of that class is below the associated bandwidth. In this way, it needs an appropriate scheduler from the user's point of view. For example, when there is real time traffic a scheduler would be needed to redirect this flow without exceeding the maximum delay.
- The link-sharing scheduler is in charge of transmitting the packets when the flow bandwidth of this class surpasses the associated bandwidth. This scheduler has the same capacities as the general scheduler, but we must also add the task of limiting bandwidth in order to accommodate the bandwidth of the class to the associated bandwidth, and it must incorporate some strategy to discard packets if this were necessary.

With the bandwidth limiter (which make use of the bandwidth estimator which is explained later), it is possible to guarantee that no class will use the surplus bandwidth corresponding to another class requiring it. We can conclude that CBQ guarantees a minimum bandwidth to those classes that have this predetermined.

Packet schedulers use the concepts of selector and delayer. The selector defines the class that in every moment has permission to send the next packet. For example, if classes are classified according to priorities and we begin with the highest priority one and continue through the following classes using a Round Robin, we say that CBQ is a WRR (Weight Round Robin) type scheduler. The delayer calculates the moment when the class that is surpassing the associated bandwidth transmits the next packet. The class that is delayed has its bandwidth rate limited to the associated bandwidth in the hierarchy.

CBQ also requires the following mechanisms:

- A packet classifier, to classify packets that go through the router into the appropriate output link class. CBQ does not specify any special requirement for this mechanism. The classifier defines the functionality of the scheduler for the user, associating the flows with classes.
- A bandwidth estimator, which estimates the bandwidth used by each class in a specific time interval, in order to determine if it is receiving or not the bandwidth associated to that class. The time interval is a key parameter to determine the accuracy with which the router fulfills the hierarchical relationships in sharing the bandwidth of the link.

Among the classes there are three special types: the *exempt* class, which has no restrictions on the use of bandwidth; the *bounded* class, which does not take the surplus bandwidth from its father node, the bandwidth not used by contiguous classes; and the *isolated* class, which does not lend bandwidth to its children nodes, making each daughter class of the *bounded* type.

2.2 TBF Type Scheduler

The Linux traffic control offers the possibility of putting a queue discipline in each class. Among them, the most typical are FIFO, the priorities discipline, the familiar CBQ, and TBF. TBF (*Token Bucket Flow*) is a rate limiter based on passing packets to the network with a Token. Tokens are generated at a constant rate, the rate at which we wish to limit the flow output.

Therefore, its application consists in limiting the maximum output rate of flows without guaranteeing any minimum bandwidth or borrowing between classes.

3. EXPERIMENTAL ENVIRONMENT

In this article, we study the features of a scheduler by means of the following experimental setup. Two networks, A and B, of 100 Mbps and 10 Mbps respectively, are joined by means of a router in which the scheduler is configured, as seen in Figure 3.

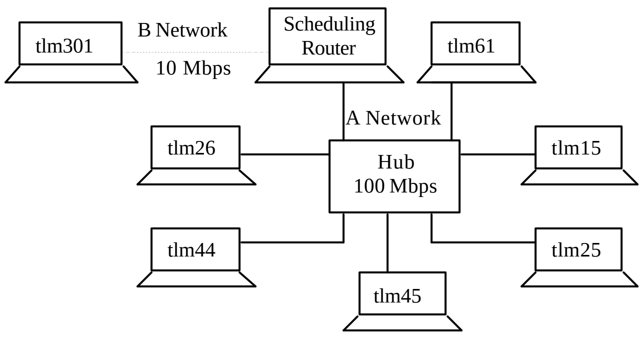

Figure 3. Work environment.

This scheduler is implemented on a PC that carries out router functions. It runs on a Pentium III PC platform at 550 MHz, Linux RedHat v6.1 operating system and kernel version 2.2.12-20. The scheduler is implemented with the traffic control provided by Linux *iproute2-2.2.4* [1] [13] configured by means of the *tc* tool [7].

The scheduler deals with flows originating from machines of Network A and directs them toward the destination machine, tlm301, placed in Network B. Since this Network B has a lower capacity, the scheduler needs to prioritize some flows over others, guaranteeing quality of service. In order to differentiate flows in the scheduler we have used the number of the destination port, associating a type of service to each one (real time, minimum delay, maximum reliability, etc.).

In order to show the efficiency of the scheduler we developed, we implemented several tools. The objective was to prove the efficiency of the activated filters in the scheduler, either by the type of traffic (TCP or UDP), such as the port accessed by this traffic, or by the type of service activated in the TOS (Type of Service) octet, or even by the origin and/or destination IP addresses. First, it is necessary to generate a packet flow where we could choose the distribution both by packet size and the time between packets. To make this possible we made a program that can generate traffic according to different statistical distributions (Poisson, Pareto, or Deterministic) and from a tracing file originated from a real traffic capture. In the destination machine, we implemented a server to receive and analyze the generated traffic.

With the analyses we carried out, we showed the independence of the scheduler in different types of distributions, both in packet size as well as in time between packets. We will now show the effect of scheduling when TCP and UDP flows interfere, which shows in turn the evolution of the size of the congestion window. Finally, we modified the accuracy of the system clock. This modification enabled us to obtain the main contributions to the study, since it shows that the features of the scheduler improve its benefits without a noticeable increase the use of the CPU of the system. This means the system can carry out very precise scheduling tasks without needing to monopolize the use of the CPU for scheduling tasks or for time control tasks.

4. EFFECT OF SIZE AND DISTRIBUTION OF PACKETS

The traffic circulating through data networks has a great variability in its behavior pattern, so for this reason we analyze in this section the behavior of the scheduler when it faces different combinations of flow distributions regarding time between packets and their sizes. A typical model used for time between packets has been the Pareto distribution, since it is the easiest *heavy-tailed* distribution. However, other self-similar models model data traffic better [9] [12]. In any case, a Pareto type distribution in small scales of seconds to minutes has a self-similar appearance. In this case, we analyze the system considering that the time between packets follows a Pareto type distribution and a packet size that follows an Exponential distribution.

Usually services with Quality of Service demands (VoIP, video, etc.) are services under UDP transport protocol, and we have carried out a simulation with UDP flows for this reason. In Figure 4, between seconds 0 and 200, we show a series of flows of 300 Kbps, 500 Kbps, 750 Kbps, and 1 Mbps, with guaranteed bandwidth at these same rates, over which a 12 Mbps UDP flow interferes at 12 Mbps. These flows appear to be independent of the 12 Mbps reference flow, since no reduction at all is noted in the flow rate when this 12 Mbps UDP flow interferes. The flows are independent of the 12 Mbps flow interference and suffer no loss at all except the 12 Mbps flow due to the bandwidth limitation of the destination Network B and because it does not have a guaranteed bandwidth.

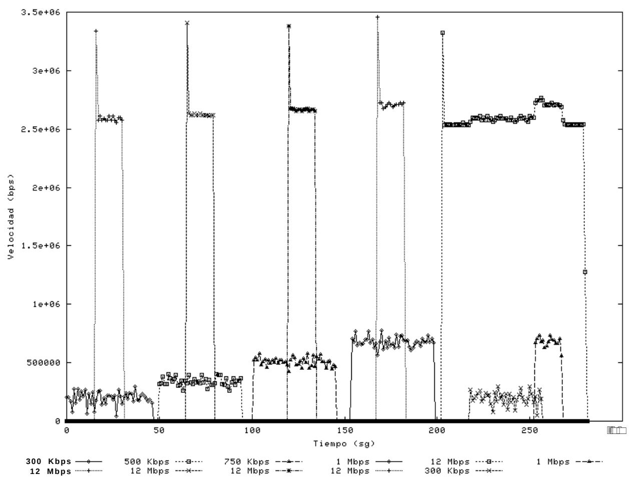

Figure 4. Time between Pareto packets and exponential size, with a CBQ type scheduler.

You can also see in Figure 4, between seconds 200 and 300 approximately, how even when there is a 12 Mbps interfering flow without guaranteed bandwidth, we can carry out flow transfers at lower rates, from 300 Kbps to 1 Mbps, with guaranteed bandwidth, again independent of the network load thanks to the CBQ type scheduler of the router.

CBQ scheduling works well mainly for low speeds and, to be sure, at low bit rates it is independent of the distribution the flow to be analyzed might have. This is due, as will be shown later, to the lack of accuracy of the system clock of the scheduler.

We carried out tests with the 9 possible combinations of Deterministic, Exponential, and Pareto distributions, packet sizes, and times between packets, in order to better assess the possible dependence of results on the distributions chosen. Our conclusion is that the different combinations, both in the time distribution of packets and their sizes, do not affect the CBQ type scheduler of the router, since the same results are obtained consistently.

4.1 Time Distribution between Packets, Before and After the Scheduler

An aspect of great concern in real time services is the delay a router can introduce, especially the *jitter* effect, or changes in delays between packets.

Since we have the possibility of seeing the time between packets at the origin and destination, we have wanted to highlight the possible changes suffered as they go through the router with a CBQ type scheduler.

We take the case in which the time between packets follows a Pareto distribution while the packet size follows an Exponential distribution. As Figure 5 shows, the time distribution between packets has changed slightly.

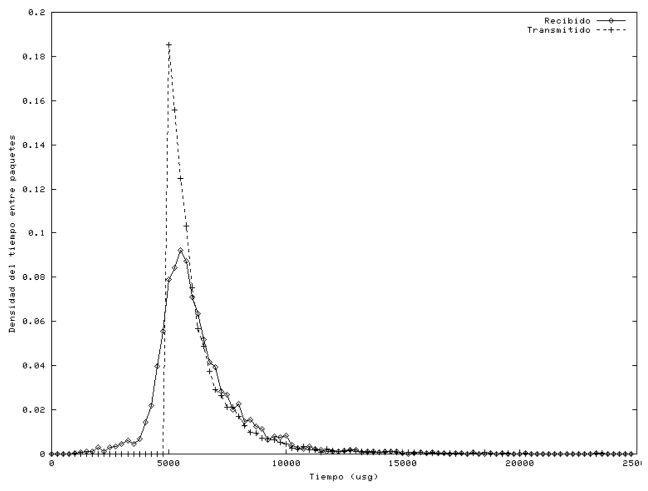

Figure 5. Time density function between packets of a 1 Mbps flow at transmission and reception.

The experiment consisted of transmitting a 1 Mbps UDP flow, adding a 12 Mbps flow in the middle of the transmission. This 12 Mbps flow was limited to a rate below 2 Mbps. The 1 Mbps flow has did not diminish its transmission speed nor did it suffer losses during the time the 12 Mbps flow was been transmitted simultaneously, so there is only one way to see the reason for this change in the time distribution of packets. Figure 6, below, shows the time between packets together with the sequence number of the transmitted packets. We can see clearly how the distribution that follows the time changes between packets when the 12 Mbps interfering flow appears, between packets 2200 and 4800. The reduction in time between packets is what causes the change in the density function from 0 to 5000 µsec, and affects the distribution function. While some time values between packets decrease, other increase, and that also shows in the density function, at 10 msec in Figure 5, for example.

The time between transmitted packets coincides with the time between packets received in those moments when there is no interference from the 12 Mbps flow. When the 12 Mbps flow acts, in some cases the time between packets at reception is lower than the time between packets when they were transmitted, at the expense of delays in the queue of previous packets, and this indicates that packets are stored before they are transmitted. CBQ maintains queues for each class and having two classes transmitting it tries to provide the minimum associated bandwidth to each class. Therefore, the transmittal time between packets is such that it results in the desired transmission bit rate.

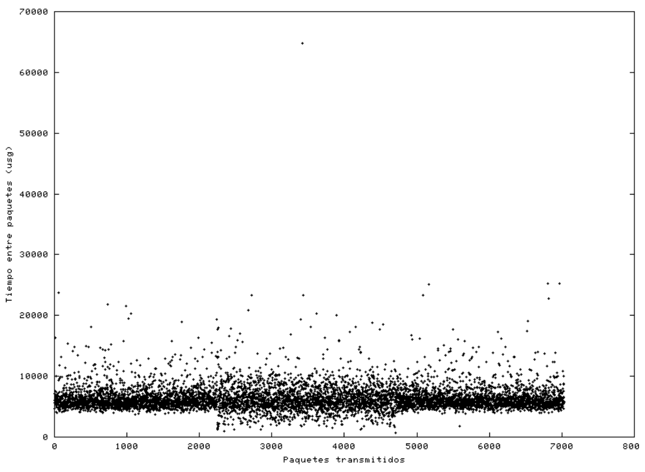

Figure 6. Time between received packets that have been transmitted with a Pareto distribution at 1 Mbps and go through a scheduler together with a 12 Mbps flow in the middle of the transmission.

We can also see a peak, approximately in packet 3500, which is not from the transmission but is generated by the scheduler of the router possibly due to some additional task of the operating system. With the interfering flow, the scheduler has to look after the two queues, and with the lack of accuracy of the clock there is a granularity which means some packet of the 1 Mbps flow has to wait more time in the queue than desired which causes an increase in the time between packets with the previously transmitted packet. If the next packet of this flow is transmitted at this time, the time between packets with the previous one is lower than the time between packets with which they arrived at the scheduler, and this creates lower times between packets at reception compared to times between packets at transmission. In this way, we can appreciate a difference in the time density function between packets at reception compared to transmission.

With the other flows used in the test, of 300 Kbps, 500 Kbps, and 750 Kbps, we get the same effect, but it is not as distinct as with the 1 Mbps flow.

5. EFFECTS OF SCHEDULING ON UDP AND TCP FLOWS

Detection of congestion in TCP flows is based on the loss of packets and therefore on the lack of acknowledgements. This is seen as a reduction to zero in the size of the transmission window of the emitter, and a momentary interruption of data transmission [15]. This is confirmed by the following test: a 1 Mbps TCP flow is transmitted, to which several interfering UDP flows of 2 Mbps, 1.5 Mbps, 3 Mbps are added consecutively, and finally a high rate UDP flow of 5 Mbps. All these flows transmitted simultaneously require a 12.5 Mbps bandwidth and therefore congest Network B. Figure 7 shows the congestion effect in Network B after the router without any scheduler configured. Later this router response is compared with a CBQ type scheduling.

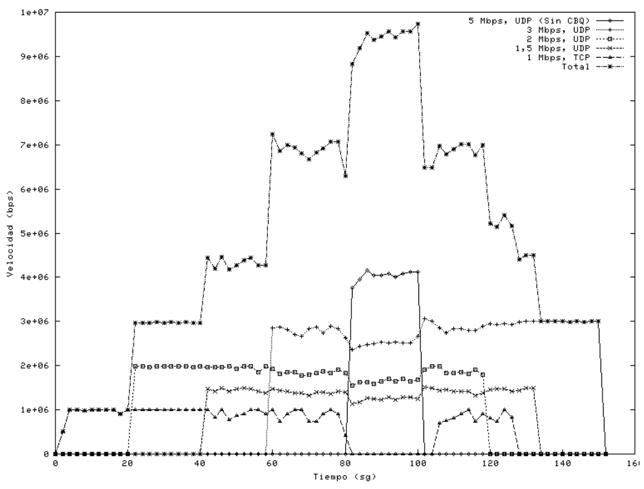

Figure 7. TCP flow with congestion and without a scheduler.

The absence of a scheduler means the TCP flow does not have a guaranteed rate and, therefore, when congestion is detected in the network the emitter reduces its transmission window to zero and stopping transmission. In Figure 7 we confirm that a maximum bit rate of 9.5 Mbps is reached, starting from second 82, limited by Network B. At this point, the network is congested and for this reason the TCP flow stops transmitting and then continues when it no longer detects congestion, approximately in second 102. The TCP flow will continue to transmit as long as it receives the acknowledgements of the packets it has sent already.

In the event of having input traffic to the router which surpasses the bandwidth of the output network (12.5 Mbps compared to 10 Mbps of the network), we can appreciate how the behavior is one of Best Effort, the default behavior of the Internet, which results in greater transmission speed of the flow whose output rate is higher, the UDP 5 Mbps flow.

With a configuration in the router of a CBQ type scheduler, which ensures a 1 Mbps rate to the 1 Mbps TCP flow by limiting the 5 Mbps UDP flow to an output rate of 500 Kbps, the TCP connection is not interrupted even though it detects congestion, as we can see in Figure 8. In this case, the maximum speed reached is 7.5 Mbps due to the following flows: 1 Mbps, 5 Mbps (limited to 500 Kbps), 1.5 Mbps (limited to 1 Mbps), 2 Mbps, and 3 Mbps. The TCP flow does not transmit at a constant rate because it really detects some congestion. This again casts doubts on the efficiency of the CBQ scheduler in complying with the requirements of the current configuration.

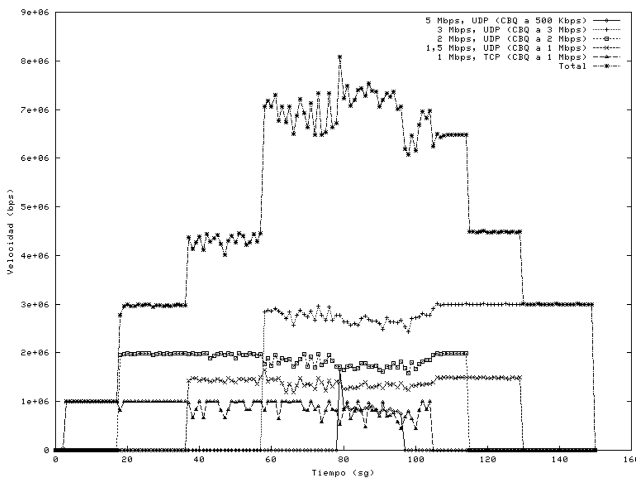

Figure 8. TCP flow with congestion and with a scheduler.

In Figure 8, we can see how the TCP flow has not stopped its transfer and the UDP flows have not decreased their transmission bit rate when a new 5 Mbps interfering flow arrives that would have congested all the 10 Mbps network. Therefore, the effects observed are due to the scheduler and the bandwidth limitation of the 10 Mbps Network B.

5.1 Effects of Scheduling on the Congestion Window in TCP Flows

In this case we analyze the evolution of the size of the congestion window of a TCP connection, that is the number of packets transmitted at each moment by the transport protocol without waiting for an acknowledgement. This value is obtained directly from the kernel of the operating system of the computer that is transmitting the data with TCP protocol. In order to obtain this value we have modified the kernel of the operating system by getting the /proc file in the directory of the computer where the congestion window will be measured. In the created file, we stored the sizes of the congestion windows of all the open TCP sockets in the machine, instantaneously, in order to be able to represent and analyze this data later. In order to add this file we have had to modify several TCP protocol files and the proc system of the kernel.

We confirmed how the TCP flows stops transmission when there is a high rate interfering flow (for example, in Figure 7, between seconds 80 and 102, you can see the effect of a 5 Mbps interfering flow which congests Network B). When a CBQ type scheduler is placed in this router, we can see (in Figure 8, between seconds 80 and 100) how the TCP flow, with a guaranteed bandwidth, does not stop and maintains its transmission bit rate. It would be interesting to be able to observe the congestion control window in both situations.

In the following test, without any type of scheduler in the router, we have transmitted a 1.5 Mbps TCP flow, and in the middle of this transmission we have added a 12 Mbps UDP flow. When this flow coincides with the UDP flow at second 17[1] of Figure 9, the congestion control of the TCP flow stops the transmission and continues once the UDP flow has stopped, at second 42 of Figure 9. Due to the interfering 12 Mbps UDP flow the TCP flow has stopped its transmission and therefore there appears a plane zone in the growth of the congestion window between seconds 27 and 52 of Figure 10. Even though the UDP flow finalizes its transmission at second 33 of Figure 9, the TCP flow does not start transmitting again until second 42 of Figure 9. This is due to the fact that before it continues to transmit it must receive the acknowledgements it did not receive due to the network congestion and has to detect there is no such congestion any more.

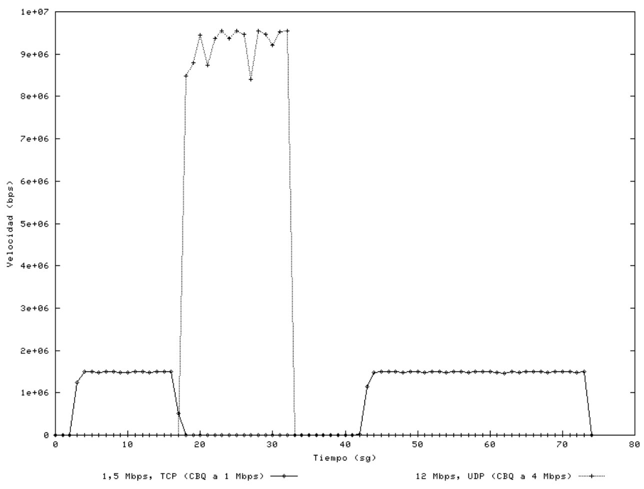

Figure 9. TCP and UDP flows without a scheduler.

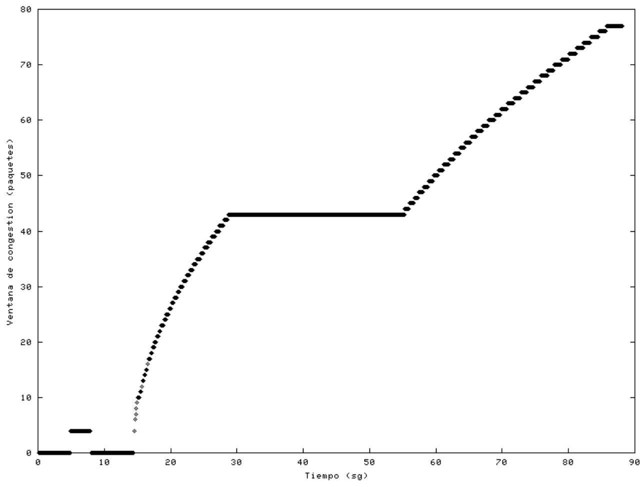

Figure 10. Congestion window without a scheduler.

When we observe the evolution of the size of the congestion window, we can appreciate the effect of the interference of the

[1] Figure 9 and 10 (as well as Figure 12 and 11) have different timing because different computers were used, but match the same experiment.

UDP flow. In Figure 10, we observe how the value of the congestion window increases during the transmission (due to the Slow Start effect) and stops when it coincides with the interfering UDP flow between seconds 27 and 52. The value of the congestion window does not decrease to a null value at this stage since it indicates the number of packets sent awaiting acknowledgement from the receiver. When it coincides with the interfering UDP flow, the packets sent are lost or the acknowledgements are lost or cannot be transmitted, and due to this, the TCP flow stops transmitting more packets and continues to have the same number of packets without acknowledgement.

With the same situation and using a CBQ scheduler that guarantees the transmission bit rate of TCP from 1.5 Mbps to 1Mbps and the UDP flow from 12 Mbps to 4 Mbps, transmission is done in the same order as in the previous case.

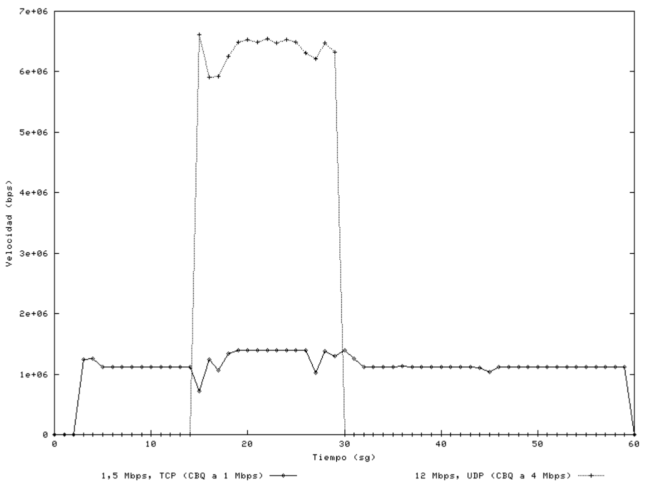

Figure 11. TCP and UDP flows with scheduler.

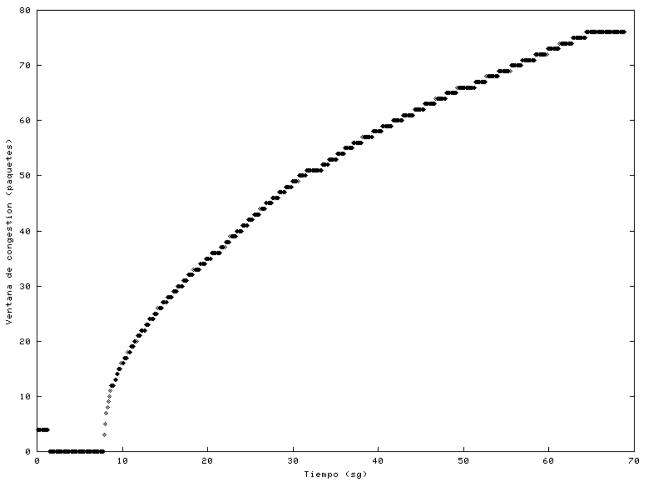

Figure 12. Congestion window with scheduler.

In this new situation, the TCP flow does not stop transmitting in spite of the interference of the 12 Mpbs UDP flow. In Figure 12 we see the value of the congestion window corresponding to this test. We cannot detect the effect of the UDP interfering flow on the congestion window nor on the transmission bit rate as observed in Figure 11.

The analysis of the results led us to consider a series of modifications on the system in which the scheduler has been implemented in order to guarantee a certain quality of service. These modifications and their effects are shown below.

6. EFFECTS OF THE ACCURACY OF THE CLOCK IN SCHEDULING OF NETWORK SERVICES

In this section, we propose a number of modifications on the scheduling system related to the accuracy of the clock of the Linux operating system. These modifications have a positive effect on service scheduling, but they reduce the use of the CPU available for the scheduler. However, as this section shows, the use of the CPU is not large and does not affect the scheduler much, but increases its accuracy when the time comes to limit and guarantee rates for different flows, improving these guarantees for flows with greater rates.

To modify the accuracy of the clock we made use of two variables belonging to the kernel that make it possible to adjust the accuracy of the clock. These variables are PSCHED_CLOCK_SOURCE and HZ.

The PSCHED_CLOCK_SOURCE parameter can adopt three values:

- PSCHED_JIFFIES (default value). The accuracy of the clock is based on small time intervals (jiffies). In PC architectures it has clock granularity with HZ=100, of hundredths of seconds, and is not sufficient for real time tasks. This granularity, for example, is with which the system process scheduler functions.

- PSCHED_GETTIMEOFDAY: Requests the time of day and, for this, needs several instructions from the assembler to complete an unnatural structure that separates seconds from microseconds.

- PSCHED_CPU: This is a value not all processors can take, only last generation Alfa and Pentium processors. It requests the time of day with a single instruction of the assembler, and resulting in a value offering the most accurate clock value.

Regarding the HZ variable, this indicates the number of interruptions that occur in the kernel in one second. Its default value is 100. Obviously, the higher this value the higher the accuracy obtained. This means, in turn, a greater use of the CPU by the operating system to control times and interruptions.

We now use the modifications explained above to analyze the features of the scheduler regarding the effect on the end to end delay, the advantages on the output rate limitation of the router, and we determine the low cost for the CPU in providing a greater accuracy and improving the operation of the scheduler.

6.1 End-to-End Delay

By means a function similar to ping an end-to-end delay study can be done. To do this we activate port 7 (echo) in the receiver and we send packets with the transmission time-stamp in order to compare it with the reception time. This packet is transmitted every second, and we transmit as many packets as seconds go by.

Once the transmission of packets is initiated to determine the maximum end-to-end delay (RTT), ten seconds later a 1 Mbps UDP interfering flow is added. After another ten seconds have passed, another 1 Mbps UDP interfering flow is added, so we have a total of 2 Mbps interfering flow. This continues consecutively, every 10 seconds, until we have 10 UDP flows of 1 Mbps each simultaneously, which produce a total interfering flow of 10 Mbps (the congestion limit of Network B).

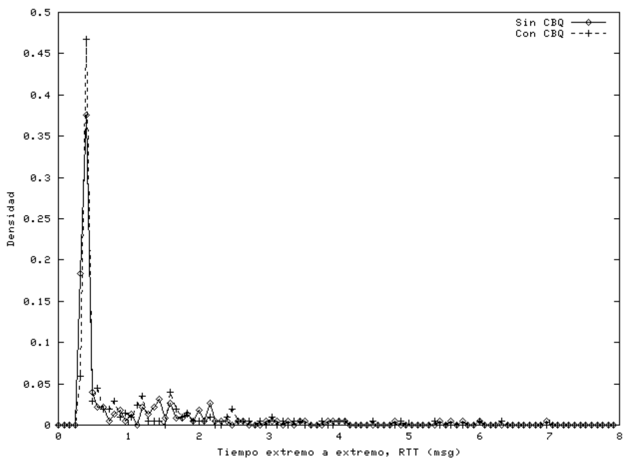

Figure 13. End-to-end time density function with and without a scheduler and with an interfering flow.

In Figure 13 we have the density function of the end-to-end time values, albeit without any type scheduling such as a CBQ type scheduler. We can see how, even though there is a predominant end-to-end time of 0.4 msec, there is a notable variation in the increase of end-to-end time due to interfering flows. This confirms that the use of a scheduler improves very little the end-to-end time, with both curves being very similar.

The CBQ scheduler does not introduce a great improvement in the end-to-end time, unless the clock accuracy of the system where it is implemented is improved. For this, it suffices to see the data represented in Table 1.

Table 1. Median RTT values with different accuracy clock values.

PSCHED_CLOCK _SOURCE	PSCHED_JIFFIES	PSCHED_CPU
HZ=100	3,14 msec	1,82 msec
HZ=1024		1,60 msec
HZ=10000		0,85 msec

We can see that placing the CBQ scheduler improves the RTT very little due to the lack of clock accuracy of the scheduler. The modification of the accuracy of the clock makes it possible to achieve some RTT median values, which appear in Table 1. This mean value has been calculated under the same conditions together with a 12 Mbps interfering flow that has a minimum guaranteed bandwidth of 2 Mbps by means of a CBQ queue discipline.

We can see how the first change of the PSCHED_CLOCK_SOURCE variable from the initial value PSCHED_JIFFIES to a value of PSCHED_CPU improves the average considerably, lowering the average RTT value 3.14 msec to 1.82 msec. Later, with the change of value of the HZ variable, it is less affected by the interfering flow. When the accuracy of the clock in the scheduling system is improved, the packets arriving at port 7 every second are treated more quickly in the scheduler, even though there is a 12 Mbps interfering flow. Therefore we can be sure the CBQ type scheduler fulfills its requirements better when clock accuracy is greater, and there is a notable improvement in the quality of service when the CBQ scheduler is implemented in the router.

6.2 The Clock Accuracy and Scheduling

As we have already seen, the traffic control of the Linux operating system offers the possibility of placing a queue discipline in the network interface. If this queue discipline is of the CBQ type, we can select another different queue discipline in every output class. Among the queue disciplines that can be added are the FIFO discipline, the priorities one, the CBQ type and TBF.

6.2.1 TBF Rate Limiter

In this section, we analyze the effect of clock accuracy when the time comes to limit the maximum output rate offered by a TBF type queue discipline in Linux.

In a first test, in order to know the features of the scheduler, the accuracy of the clock of the operating system is the default one and therefore, has the values of PSCHED_CLOCK_SOURCE with PSCHED_JIFFIES and HZ equal to 100 [10][11]. For this simulation, we have transmitted a series of 1 Mbps UDP flows, going to different classes, in an initial CBQ queue discipline with each queue of each class ending in a TBF queue discipline. Each class has a different rate limitation, starting with 100 Kbps and increasing 100 Kbps each time until 900 Kbps. This means that flows must leave limited to 100 Kbps, 200 Kbps, 300 Kbps, ... until 900 Kbps. Also, three 1.4 Mbps UDP flows are transmitted to limit their output rate to 1 Mbps, 1.1 Mbps, and 1.2 Mbps respectively. Finally, we transmitted a 3 Mbps flow to limit it to 2 Mbps and observe the limitation at higher output rates. Each limited flow is a burst of Figure 14.

The rate limitation up to 400 Kbps has been satisfactory, but it has not been possible to limit it to 500 Kbps, and instead it has been limited it to rate below 400 Kbps. This also happens with the limitations of 700 Kbps, 800 Kbps, 900 Kbps, and 1 Mbps, which have been limited to slightly below 600 Kbps. The size of packets is constant, 1400 bytes, for all flows, in order to require times, between packets, that are not too small for high rates. With this data, we can see that the choices of these bit rates at which flows are limited are due to the accuracy of the system clock where the scheduler is. These output rates are always, with times between packets, multiples of 0.01 sec (HZ = 100), and because a constant packet size is transmitted for all flows, we get the result shown.

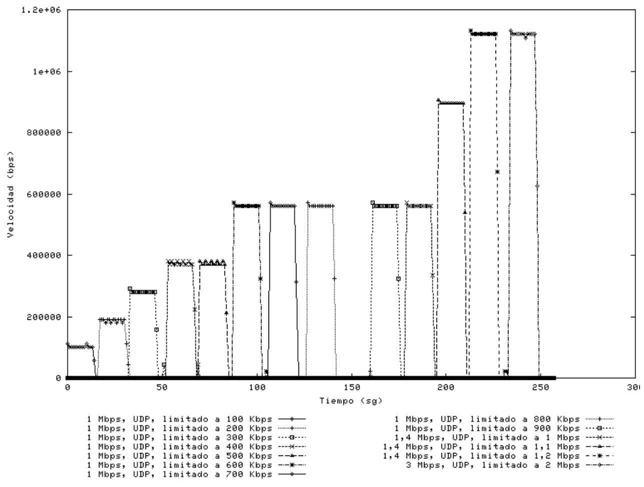

Figure 14. Rate limitation by TBF.

The maximum output rates provided by the TBF queue discipline is seen to be improved by a greater accuracy of the clock, as observed in Figure 15.

When we change the PSCHED_CLOCK_SOURCE variable in PSCHED_JIFFIES (default) value to the value in PSCHED_CPU (already an improvement can be seen, but not as much as desired, confirmed by Table 1) and the value of the HZ variable from 100 to 1024, we can see in Figure 15 how the desired output rates are obtained. Even as the flow should have reached a 2 Mbps rate, it is unable to do this and remains at 1.9 Mbps rate. This flow also obtains its desired output rate of 2 Mbps with a new change in the HZ variable to 10000.

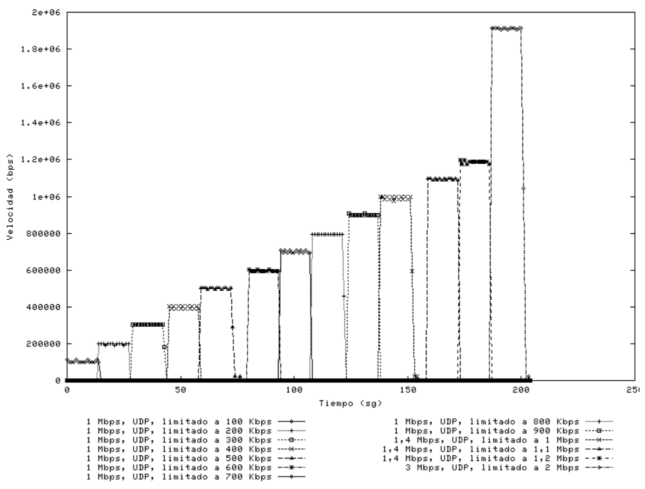

Figure 15. TBF output rates with PSCHED_CPU and HZ=1024.

With these figures, we can conclude that the lack of accuracy of the clock is what stops us from getting the maximum output rates desired, regardless of the calculations and internal procedures of the scheduler. Therefore, we can be certain that the TBF rate limiter is sensitive to the accuracy of the clock, but perfectly fulfills the requirements if this accuracy is increased.

6.2.2 CBQ Minimum Bandwidth

In Figure 16, clock accuracy is achieved with the values of the variables in PSCHED_CPU and HZ=10000 and we can see the rate limitation achieved with CBQ. In it we transmit a 1.5 Mbps UDP flow limited with CBQ to 1 Mbps and this is interfered by another 12 Mbps UDP flow limited to 6 Mbps. We see how the interference of the second flow between seconds 14 and 30, causes the 1.5 Mbps to increase its rate above the Mbps to which it was limited.

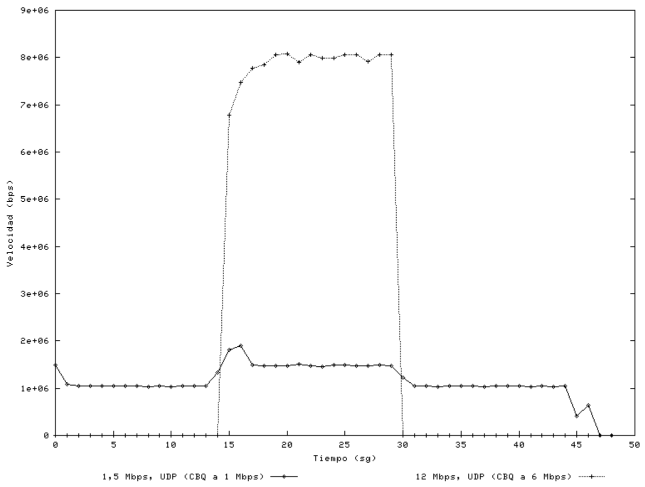

Figure 16. Output rate limitation with CBQ (PSCHED_CPU and HZ=10000).

Contrary to what happens with the TBF type scheduler, the maximum rate limitation does not improve in a CBQ type scheduling with a greater accuracy of the clock. This is because the characteristic of the CBQ scheduler is to guarantee a minimum bandwidth and not to limit the maximum output rate.

The flows with a small packet size require a time interval between packets, which is also small, in order to obtain a specific transmission bit rate. Therefore, with a greater accuracy of the clock, this type of flow with a small packet size makes it possible to better ensure its minimum bandwidth with the CBQ type scheduler. In order to demonstrate this we can see Figure 17 where the accuracy of the clock is determined by the values of PSCHED_CPU and HZ=100. The 2 Mbps UDP flow is the one that has a small packet size and is the flow most affected by the presence of the new 5 Mbps UDP flow. This phenomena can be seen in Figure 17 between seconds 40 and 61, where the 2 Mbps flow reduces its rate to nearly 1 Mbps when the 5 Mbps flow appears in this interval limited to 2 Mbps.

With variable HZ=1024, the accuracy is greater and with the CBQ scheduler the bandwidth associated with this flow of 2 Mbps can be guaranteed, with a small packet size, of 2 Mbps. As can be observed, between seconds 40 and 60 of Figure 18, how the 2 Mbps UDP flow does not decrease its transmission bit rate, which is what we wanted to guarantee.

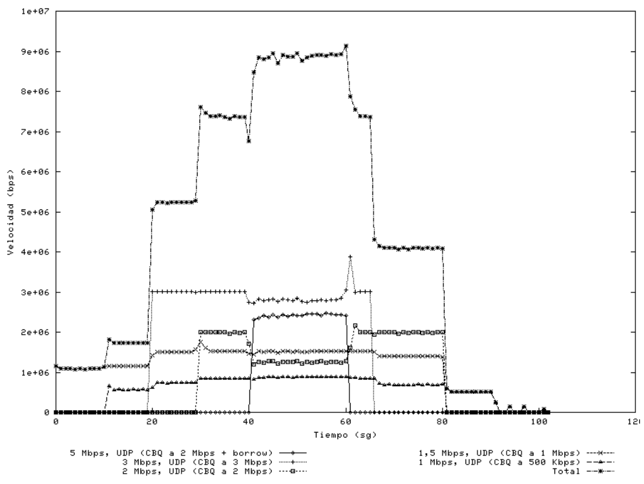

Figure 17. Different flows with CBQ (PSCHED_CPU and HZ=100).

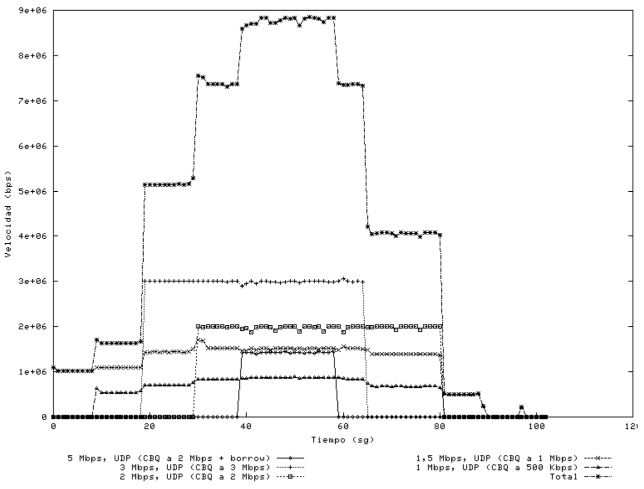

Figure 18. Different flows with CBQ (PSCHED_CPU and HZ=1024).

We see an increase of transmission bit rate in the first 1.5 Mbps UDP flows (with 2 Mbps guaranteed and limited with CBQ) and 1 Mbps (with 500 Kbps guaranteed and limited) with each new added flow. These increases are seen approximately in seconds 10, 20 and 30 in Figure 17, and with improved accuracy of the clock in Figure 18. It is an effect that the CBQ queue discipline is unable to control in spite of changing the accuracy of the clock. We can conclude that the CBQ queue discipline guarantees a minimum bandwidth that does depend on the accuracy of the clock, but does not limit the maximum bit rate of the output transmission.

6.3 Clock Accuracy and CPU usage

Once we confirm that the features of the scheduler (either with a TBF or a CBQ type) improve when the clock accuracy improves, we try to determine in what measure this improvement affects CPU usage.

Table 2. Percentage of CPU use without scheduling.

PSCHED_CLOCK _SOURCE	**PSCHED_JIFFIES**	**PSCHED_CPU**
HZ=100	①	②=①+0,6%
HZ=1024	①+1,2%	②+0,9%
HZ=10000		②+11,8%

In order to obtain a greater accuracy of the clock we use higher HZ values, and these require a greater use of the CPU, as should be expected. In order to measure the use of the CPU we take as a reference the use of the CPU due to management tasks of the operating system itself; that is, without any user process being executed and with the variable HZ=100 and PSCHED_CPU, we take the reference value called ② (Table 2).

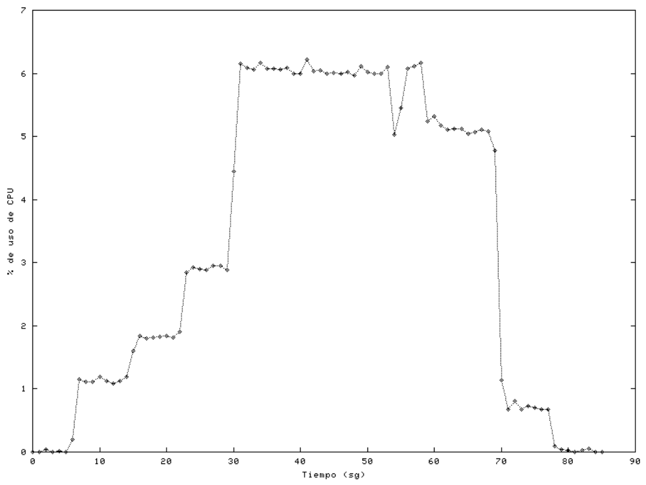

Figure 19. Relative percentage of use with PSCHED_CPU and HZ=100.

Because it can carry out scheduling, and more concretely in the test of Figure 17, the kernel requires CPU usage that is 6% above that of the idle state ② as shown in Figure 19 (when all the flows coincide simultaneously between seconds 40 and 60 in Figure 17). Regarding this reference ②, the very change of the HZ variable to a value of 10000 and without any ongoing user process running, CPU usage of 11.8% more is required compared to ②, only for tasks of the operating system itself. In the case of HZ=10000, when the scheduling process is done the peak of maximum use is 7% greater than the idle state. Regarding the reference point defined previously ②, the peak of maximum use (HZ=10000) implies an increase of 19.7% in the use of the CPU, compared to 6% for HZ=100. As the accuracy of the clock improves, the use of the CPU naturally increases. An improvement of accuracy of the clock to HZ=10000, 100 times greater, means an increase of 11.8% in the idle state, which is a fully acceptable increase of the use of the CPU. While for HZ=100 the scheduler involves a maximum use in scheduling of 6%, for HZ=10000 it involves a maximum use in scheduling of 19.7% compared to the idle state of HZ=100, and this being 12.9% over the maximum use in scheduling of HZ=100.

7. RELATED WORKS

Existing studies on scheduling such as articles [4] [7] [14] expound the function of CBQ schedulers on Linux and BSD systems, but limiting themselves to confirming their properties on different network interfaces such as Ethernet and ATM.

However, in this paper we have been able to improve the scheduler function on a general purpose operating system such as Linux at the expense of increasing the accuracy of the clock with different existing mechanisms.

As a line of study, we have still to check the use of external clocks of high accuracy cooperating with the operating system without implying an additional load to the CPU.

8. CONCLUSIONS

In this study, we have confirmed in the first place that the CBQ scheduling algorithm is not a good rate limiter, but it does guarantee the minimum bandwidth. This bandwidth is guaranteed with a greater efficiency if there is greater clock accuracy in the scheduler. The flows with small packets are those most affected if the accuracy of the clock is not very good, since in order to get a certain transmission bit rate they require a smaller time between packets and therefore a greater accuracy of the clock.

The CBQ type queue discipline scheduler has the option of loaning bandwidth, which improves the performance of the link. Bandwidth loaning is based on the fact that if there is available bandwidth in the link there is no need to lose packets for those flows that have no full of the necessary bandwidth guarantee. This supposes no restriction to the output rate of the flow of a specific class, if there is available bandwidth in the network.

This CBQ scheduler also enables flows of different transport protocols to coexist such as TCP and UDP. It also avoids decreasing the transmission bit rate due to the congestion control of TCP flows that are guaranteed a minimum bandwidth, when faced with UDP flows of a high transmission rate. We can be sure this means robustness in the presence of interfering flows.

Classification of flows in different classes is done by means of the configuration of filters, and these can be based on any header parameter in the packets. One of the characteristics of CBQ is hierarchical filtering. In this manner, we can configure better bandwidth distribution of the link, depending on as many parameters as desired.

The TBF rate limiter, as a queue discipline, which also offers traffic control in Linux, is strongly dependent on the accuracy of the clock when the time comes to limit the output rate, especially for high rates. The desired output transmission bit rates are achieved with a higher accuracy clock. The greater the accuracy of the system clock, the greater the adjustment of the output rate to the desired rate.

The major problem we have found is the accuracy of the clock. For scheduling tasks at high bit rates the accuracy of the existing clock in general purpose operating systems such as Linux is bad, and does not enable treating packets with a high temporal accuracy for greater performance.

In order to increase the accuracy of the clock we can modify some parameters of the kernel. The parameters we have tried to change have been the PSCHED_CLOCK_SOURCE variable and the HZ variable. The observed effect has been a small increase in the use of the CPU, but a great improvement regarding the accuracy of the output bit rate of the flows. The CBQ scheduler guarantees a minimum bandwidth better and TBF obtains an output rate more adjusted to what is required.

Clock accuracy can be modified and we can observe the different aspects of the different values of accuracy supplied by the clock.

The percentage of CPU usage has not increased as much as was expected for a greater clock accuracy. Therefore, this increase in the clock accuracy is very interesting.

With the tests carried out and the improvement in the accuracy of the clock we have seen that it is possible to implement a scheduler to guarantee quality of service, on a low cost platform as in a PC with Linux, at least at low and medium bit rates (10/100 Mbps).

9. REFERENCES

[1] W. Almesberger. Linux Network Traffic Control, Implementation Overview. Technical Report SSC/1998/037, EPFL ICA, November 1998. ftp://lrcftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss. An Architecture for Differentiated Services. IETF RFC2475, December 1998.

[3] R. Braden, L. Zhang, S. Berson, S. Herzog and S. Jamin. Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification. IETF RFC2205, September 1997.

[4] K. Cho. A Framework for Alternate Queueing: towards Traffic Management by PC-UNIX based routers. In USENIX Annual Technical Conference, New Orleans, Louisiana, 1998.

[5] G. Dhandapani and A. Sundaresan. Netlink Sockets Overview. Technical Report The University of Kansas, September 1999.

[6] S. Floyd and V. Jacobson. Link Sharing and Resource Management Models for Packet Networks. IEEE/ACM Transaction on Networking, Vol.3, No.4, pp 365-386, August 1995.

[7] A. Gómez-Skarmeta and A. López-Toledo. Control de tráfico mediante Class-Based Queueing gestionado por RSVP usando Linux como router con capacidad QoS. In Proceedings of II Jornadas de Ingeniería Telemática JITEL'99, Leganés (Madrid), September 1999.

[8] J. Hadi-Salim. IP Quality of Service on Linux. Technical Report Nortel Network, Ottawa, Canada, July 1999. http://www.davin.ottawa.on.ca/ols

[9] W. Leland, M.S. Taqqu, W. Willinger and D. Wilson. On the self-similar nature of ethernet traffic (extended version). IEEE/ACM Transactions on Networking, Vol.2, No.1, pp 1-15, February 1994.

[10] J. C. Mogul. Efficient Use of Workstations for Passive Monitoring of Local Area Networks. Proceedings of ACM SIGCOMM Symposium on Communications Architectures and Protocols, Philadelphia, PA, September 1990.

[11] V. Paxson, G. Almes, J. Mahdavi and M. Mathis. Framework for IP Performance Metrics, IETF RFC2330, May 1998.

[12] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. In SIGCOMM '94, pages 257-268, August 1994.

[13] S. Radhakrishnan. Linux-Advanced Networking Overview Versión 1. Technical Report KS 66045-2228 Information and Telecommunications Technology Center, Department of Electrical Engineering & Computer Science, The University of Kansas. Lawrence, September 1999. http://qos.ittc.ukans.edu/howto/howto.html

[14] F. Risso and P. Gevros. Operational and Performance Issues of a CBQ router. Computer Communication Review, Volume 29, Number 5, October 1999.

[15] N.J. Shah. Exploring TCP Stream Rate Control in LAN Environments. PhD Thesis, Universidad de California, Riverside, June 1999.

[16] G. Pawan and M.V. Harrick. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. IEEE/ACM Transactions on Networking, Vol. 5, nº 4, p.56661-71, August 1997.