

An Experimental Implementation of Traffic Control for IP Networks

Martin May and Christophe Diot
INRIA
BP 93
06902 Sophia-Antipolis Cedex
France
{mmay,cdiot}@sophia.inria.fr

Abstract

The Integrated Services Packet Networks are designed to offer new network services to a wide variety of applications. The architecture behind the new services consists primarily of a packet scheduler, a packet classifier, admission control, and a reservation establishment protocol. This paper describes a complete implementation of this Traffic Control extensions. We also designed and implemented a kernel interface which permits to send Quality of Service (QoS) requests to the network layer. We explain how the elements of the architecture work in concert and which mechanisms we use to guarantee a requested amount of bandwidth. Our scheduler uses a variant of the Class Based Queuing mechanism (CBQ) of Sally Floyd and Van Jacobsen [6]. For the reservation signaling and establishment we use RSVP [2]. Finally this paper provides a performance evaluation of the proposed mechanisms.

1 Introduction

Today, the Internet is undergoing a period of revolutionary change towards real-time and multimedia. This deployment is based upon four enabling technologies.

- Most new workstations now include built-in multimedia hardware, including audio codecs and video frame grabbers, and the necessary video gear is nowadays inexpensive.
- For the new hardware resources, highly-sophisticated digital audio and video applications have been developed. These services, available in the Internet, are free of charge (vic, rendez-vous, vat, ivs, Freephone...).
- IP multicasting, which is not yet generally available in commercial routers, is being provided by the MBONE - a temporary "multicast backbone".
- Applications using multicast have been developed. These applications are not only video or audio-conferencing tools; the first multicast games are now available. The need for applications like distributed games or Distributed Interactive Simulation (DIS) is increasing.

These technologies allow widely-scaled use of audio and video applications over the Internet. The results obtained by these experiments show that time-constrained applications often do not work well in the Internet because of variable delays and network congestion.

Time-constrained applications are quite different from standard data applications, and require service that cannot be delivered within the typical data service architecture. One salient characteristic of real-time applications is that they require a bound on the delivery delay of each packet. While this bound may be statistical, in the sense that some small fraction of the packets may fail to arrive by this bound, the bound itself must be known a priori.

The Internet uses datagram switching as a means of dynamically allocating network resources on a demand basis. Connectionless datagram switching facilitates the interconnection of networks with different architectures and it provides flexible resource allocation and good reliability against node and link failure. However, it provides little control over the packet delay and loss processes at the switches.

To handle the lack of control over delay and loss processes, an architecture with an enhanced scheduling mechanism is still missing. Work is underway in various IETF (Internet Engineering Task Force) working groups to include new services [1] for applications that require a specific quality of service. They also define new protocols (like IPv6 [8] or RSVP [2]) to simplify the integration of resource control in the current Internet.

But real-time QoS is not the only reason for a next generation of traffic management in the Internet. Network operators request the ability to control the sharing of bandwidth on a particular link among different traffic classes. They want to be able to divide traffic into a few administrative classes and assign to each a minimum percentage of the link bandwidth under conditions of overload, while allowing "unused" bandwidth to be available at other times. These classes may represent different user groups or different protocol families, for example. Such a management facility is commonly called controlled link-sharing [6].

The design and implementation of the INRIA Traffic Control kernel supports the features to allocate resources, to guarantee delay bounds, and to share the bandwidth among different traffic classes. The reservation establishment and signaling (in the user space) is done by the ReSerVation Setup Protocol (RSVP).

In this paper, we describe the Traffic Control kernel support for resource allocation, scheduling, and link-sharing. Section 2 gives an overview of our architecture. In section 3, we introduce the main components of our traffic control system. Then, in section 4, we present the results of the performance measurements. Section 5 concludes the paper.

2 Overview of the Traffic Control Architecture

Datagram-based networks (in our case IP-networks) are scalable and flexible, but they require more complex mechanisms and, in fact, even fundamental changes to their architecture to provide a larger set of services. Today the Internet uses a service model called "best effort". In this model, the network shares bandwidth among all the instantaneous users as best it can, and attempts to serve all of them without making any explicit commitment as to rate or any other service quality.

The services offered by the network depend on the router architecture and the implemented mechanisms in the network nodes. In the packet forwarding path of IP implementations, there is a very limited set of actions a router can take. Given a particular packet, the router must select a route for it; in addition the router can either forward it or drop it, and the router can reorder the outgoing queue with respect to other packets waiting to depart. The router can also hold the packet, even though the link is idle.

To support the new services we added some new elements to the classic architecture of today's IP implementations as illustrated in Figure 1. The first new element is a packet classifier. The task of the classifier is to examine all incoming packets and determine the corresponding traffic flow for it. If there is no corresponding traffic flow found, the packet is marked for best-effort handling.

Beside the classifier, we need a more sophisticated scheduler than a simple FIFO mechanism. This scheduler must handle the different flows according to the reservation made for that flow. Examples of such schedulers could be found in [15] [9] [10]. For the INRIA

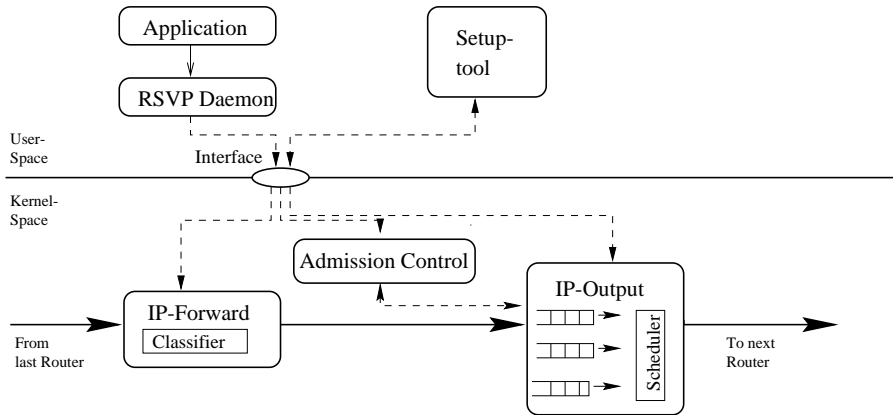


Figure 1: Architecture of the INRIA Traffic Control Kernel

Traffic Control kernel, we implemented a variant of the Classed Based Queueing algorithm [7].

For the reservation establishment and signaling, we used the ISI version of RSVP. The RSVP protocol implements the mechanisms to make reservations along the sending path of a data flow. Our RSVP Daemon communicates with our traffic control (TC) elements in the kernel via a new interface we have designed and added to RSVP.

Finally, we have implemented a simple Admission Control. The Admission Control module is lying in between RSVP and the kernel with the classifier and the scheduler. Admission Control implements a decision algorithm that the router uses to determine whether a requested service for an incoming flow can be guaranteed without impacting earlier guarantees or not. It can be implemented in kernel or user space. But since it is logically part of traffic control we placed Admission Control in the kernel.

We developed a simple Tcl/Tk tool to pass control information directly to the TC elements in the kernel. The Setup-Tool (see in Figure 1) is used to set up and maintain the elements of the architecture.

This architecture allows us to offer the services defined by IntServ [1]. The two important classes are the guaranteed quality of service [11] and the controlled load service [14].

Guaranteed service ensures that datagrams will arrive within the guaranteed delivery time and will not be discarded due to queue overflows, provided the flow's traffic stays within its specified traffic parameters. This service is intended for applications which need a firm guarantee that a datagram will arrive no later than a certain time after it was transmitted by its source.

The controlled load service is intended to support a broad class of applications which have been developed for use in today's Internet, but are highly sensitive to overloaded conditions. Important members of this class are the "adaptive real-time applications" currently offered by a number of vendors and researchers. These applications have been shown to work well on unloaded nets, but to degrade quickly under overloaded conditions.

3 Traffic Control Architecture: Classifier, Scheduler, and Admission Control

This section describes the design and implementation of significant components of the TC kernel: the scheduler (based on the Classed Based Queueing algorithm [7] introduced by Sally Floyd and Van Jacobson) and the classifier. Then, we explain the mechanisms in the kernel to calculate the parameters for the scheduler, and how the reservation request is passed

from the RSVP daemon¹ to the kernel.

3.1 The Classifier

As mentioned above, a today's router looks at the destination address of the incoming packet and selects a route for it. But the destination address is not sufficient to select the service class for each incoming packet. Our approach is to look at more fields in the packet header, such as source address and the destination and source ports. This allows us to distinguish between the different data flows, like UDP video or audio streams, or different TCP telnet or FTP flows.

For each data flow we add a filter to the classifier. The filter includes the source and destination address, the source and destination ports, and a pointer or flow handle to the corresponding service class.

When a packet arrives at the router we do a hash table lookup to find the corresponding filter entry with the pointer to appropriate outgoing queue. If no filter entry was found, the packet is classified for the best-effort service and met to the default outgoing queue.

The classifier implementation issues are complexity and processing overhead. To reduce the processing time for each packet this filter information is stored in a hash-table which permits a fast access times even for large table sizes.

3.2 The Packet Scheduling

Future integrated services networks [3] [5] will support multiple service classes that include real-time service, best-effort service, and others. In addition, they will need to support controlled link-sharing [6].

We implemented a variant of Class Based Queueing (CBQ) based on the implementation of Sally Floyd [7]. This scheduler integrates the postulated features. The conceptual framework for CBQ assumes two scheduling mechanisms: a general scheduler that schedules the packets from leaf classes without regard to link-sharing guidelines, and a link-sharing scheduler that surveys that the leaf classes do not exceed their link-sharing allocations. Both schedulers are integrated in one scheduling algorithm.

In our TC Kernel we use a separate queue for each class associated with the output link. After a packet is transmitted on the output link, the scheduler decides which class is allowed to send the next packet. This next class is selected in priority order. Within classes of the same priority, the scheduler uses a variant of weighted round-robin (proportional to the allocated bandwidth). The weights determine the number of bytes this class is allowed to send at each round. In Figure 2 we illustrate the architecture of the CBQ scheduler. The weighted round-robin ensures that each priority-one class receives its allocated bandwidth.

Before the general scheduler transmits a packet, the scheduler checks if there is time left to send, by comparing the time-to-send field with the current time. If the time-to-send field is greater than the current time, the scheduler can only send the packet if the link-sharing permits to borrow bandwidth from higher-level classes. The leaf classes can try to borrow bandwidth from all parent classes they are linked with. For more details about Class Based Queueing and link-sharing we refer to [6].

In Figure 2 we show an example for a link-sharing hierarchy. Consider for this example n organizations sharing on output link. The administrative policy surveys that organization 1 (or Share 1) gets at least 25% of the link bandwidth. In Share 1 15% of its bandwidth is reserved for real-time traffic. Beside 5% for best-effort traffic, the organization can reserve up to 5% of its bandwidth to other applications, like FTP or telnet.

In our implementation of CBQ we have a tree representation of the hierarchy. The physical link represents the root node of the tree and the leaf nodes represent a physical queue. Each leaf node is connected to one parent. We also included a pointer to the node the leaf

¹we modified the implementation of Bob Braden Release 4.0a9

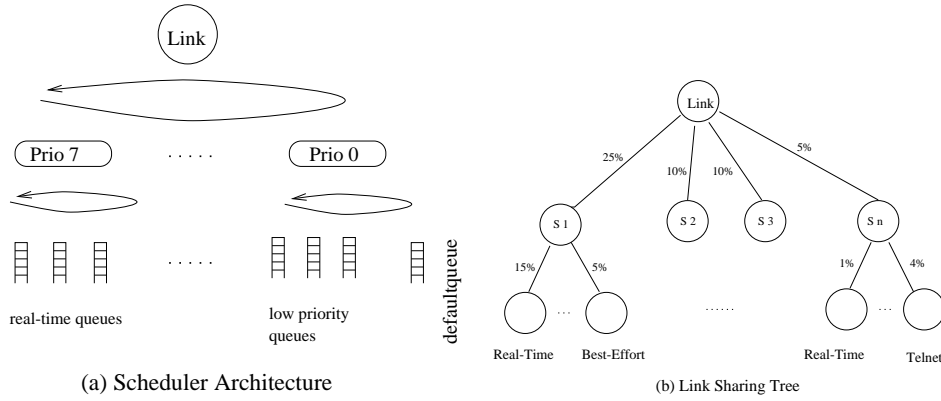


Figure 2: Link Sharing Example

can borrow bandwidth from. That for the link-sharing hierarchy may differ from the logical hierarchy.

For each new class we add to the scheduler we have to specify its characteristics. These are defined by the following values:

- *bandwidth* for the corresponding flow.
- *priority* of the class [0..7].
- max. queue length. With this information a maximal delay can be guaranteed by the scheduler.
- *offtime*. The parameter *offtime* gives the time interval that a overlimit class must wait before sending another packet. This parameter determines the steady-burst size for a class when the class is running over its limit.
- *maxidle*. The variable *idle* is the difference between the desired time and the measured actual time between the most recent packet transmission. The parameter *maxidle* limits the upper bound for the average idle time. Thus *maxidle* bounds the credit given to a class that has recently been under its allocation.

3.3 Admission Control

In order to insure that commitments made for certain classes of packets can be met, it is necessary that resources be explicitly requested, so that the request can be refused if the resources are not available. The decision about resource availability is called admission control.

Admission control uses the token bucket values for the requested bandwidth to decide, whether the bandwidth for the desired bandwidth can be guaranteed or not. Our simple admission control calculate the allocation for a guaranteed service with $\mu_i = tbr_i + tbd_i$, where μ_i is the requested bandwidth, tbr_i the token bucket rate, and tbd_i the token bucket depth. For a controlled load service the allocation corresponds to $\mu_i = tbr_i$. Admission control accepts the flow, if

$$\beta \geq \sum_{k=1}^n \mu_k + \mu_i$$

where β denotes the reserved bandwidth for the type of service. This amount of bandwidth is set during the setup phase of the router.

When we want to guarantee a maximum delay bound, admission control needs a more complex scheme. The admission control algorithm for such type of service demands whether a hard limitation of the number of guaranteed flows or a great share of the bandwidth, using the theoretical approach of Parekh and Gallager. To find out how much bandwidth we have to reserve for a certain delay bound we consult their well-known equation for the delay bound

$$d = \frac{tbd}{g} + \frac{(N - 1) * P_1max}{g} + \frac{N * P_2max}{r}$$

with *tbd* standing for the token bucket size, *g* the allocated bandwidth, *r* the link speed and *P_{1max}* the max. packet size of the sender and *P_{2max}* the maximum packet size of the network. *N* denotes the number of intermediate hops.

By inverting this formula, we get the rate we have to allocate to meet the delay bound.

$$g = \frac{tbd + (N - 1) * P_1max}{d - \frac{N * P_2max}{r}}$$

To guarantee a delay of 50 ms for a flow in our testbed, we would have to reserve about 346 kbyte per sec². When we assume a network with 10 routers the allocated bandwidth would rise to 539 kbytes per sec. These allocations seems to be very high compared to the wished delay. We implemented this approach, but it seems rather useful. Therefore we measured only sessions using the the controlled load service.

To calculate the max. queue length of the new class, the kernel needs information about the desired delay of the session. We use the value of the maximum delay to limit the queue size of the flow. This queue size is calculated with:

$$maxq = \frac{\mu_i * maxdelay}{avg.packetsize}$$

where μ_i is the requested bandwidth. The value of *maxdelay* is set by the user or application.

3.4 The Traffic Control Interface

A central novelty in IP networks, is the new TC interface we created for our architecture. This interface permits to pass QoS requests from user space to the kernel.

For future compatibility we have chosen a generic specification for the QoS requests we pass through our TC interface. The set of general control and characterization parameters used by network elements supporting the integrated services framework is defined in the Integrated Services Working Group [1]. We use their traffic specification (TSPEC) parameters defined in Reference [12], to describe the traffic characteristics of the data flow.

We use a single *tc_request* structure for all service requests. This structure includes the type of the request (e.g. *ADD_SHARE*, *ADD_FILTER*, *ENABLE_TC*, or *MOD_FLOW*) and the corresponding parameters. The requests are used by our Setup-Tool and RSVP.

The requests to manipulate the flows and filters are only used by the RSVP daemon. These requests use token bucket TSPEC values to describe the required flow characteristics. A token bucket specification includes an average or token rate *tbr* and a bucket depth *tbd*. Both *tbr* and *tbd* must be positive. The token bucket TSPEC takes the form of a token bucket specification plus a peak rate *tbp*, minimum policed unit *m*, and a maximum packet size *M*. For a guaranteed service the values for a maximum delay and service rate are added.

The *tc_request* also includes filter information. These are the source and destination address and for IPv4 the source and destination port numbers of the UDP or TCP headers.

²We assume a token bucket size of 16 kbytes. This is pretty small, and not far from what TCP uses in practice. For video applications, this would actually be much larger. The maximum packet sizes correspond to the example in the performance section (*P_{1max}*=500 kbyte and *P_{2max}*=1500 byte.)

For IPv6, we do not need the port numbers to classify the incoming packets, instead we use the flow-ID field.

Beside these requests, we can use the interface to ask the current state of the router. We collect statistics like the number of dropped packets, or the mean queue size of all flows.

3.5 Reservation Mechanisms

During the initialization of the router we build up the class hierarchy for CBQ. Therefore we add a shared RSVP class to the root of the link-sharing tree. In our tests we took 80 percent of the available bandwidth for RSVP traffic. All classes added to this share are from priority 7. Then we add another shared class for each service we offer. Up to now, this is a controlled load service share and a guaranteed service share. This service hierarchy is shown in Figure 3.

The queues for all new flows are then linked to the corresponding share tree nodes of the service classes.

The driving engine of our TC kernel is RSVP. Details about RSVP and the use of RSVP can be found in [2] or [13]. An application uses the API (Application Programming Interface) of RSVP to pass the desired traffic parameters towards the network. Then RSVP converts the API values to the general traffic specification values. These values are copied in our new `tc_request` structure and then passed through the TC interface to the kernel.

When a reservation message arrives at the router, RSVP tries to make a reservation for the new flow. The RSVP daemon passes a message to the kernel (to admission control) via the new Traffic Control interface. This request includes a service definition, the requested bandwidth in a token bucket notation, additional information like a maximum delay, the maximum packet size, and values for a maximum packet burst.

When the parameters for the new class are calculated, the class is linked to the link-sharing tree. Then the RSVP daemon passes the filter information to the classifier. Then we calculate the hash table keys and insert the filter in the hash table.

When a reservation is finished and a TEAR DOWN message arrives at the router, the class for the session is removed from the link-sharing tree. For admission control we add the reserved bandwidth to the available bandwidth. Finally we remove the filter information from the hash table.

4 Performance Evaluation

In this section, we present the results of our performance evaluation to illustrate the delay and throughput properties of the INRIA traffic control kernel. To demonstrate the tightness of our approach towards traffic control, we contrast the delay distribution and throughput of our integrated services router with a best-effort router. The experiments were conducted on our local test-bed at INRIA Sophia Antipolis. We used a pool of eight PCs (Pentium PPro with 64 MByte RAM) running a modified version of Linux version 2.1.23.

In Figure 3, we see the network setup we used to measure the packet delay and bandwidth characteristics with our INRIA TC kernel. The PC are connected to 10 MBit/s ethernet which were exclusively used by the PC. Some PC are linked to more than one network (like the routers or the PC we used for the delay measurements). The routers are also PC with the same configuration as the other machines.

We created a shared class for all RSVP flows. This class has a guaranteed share of 0.8 from its parent node which is the link itself (10 Mbit/s). Then, we built up a class for all controlled load service flows (CLS). This class reserved a share of 0.6 from the RSVP parent class. The link sharing setup is shown in Figure 3.

To setup the reservations in the intermediate routers, we used the RSVP-tools included in the ISI distribution of RSVP. The parameters for the reservation correspond to the token bucket notation of the RSVP specification.

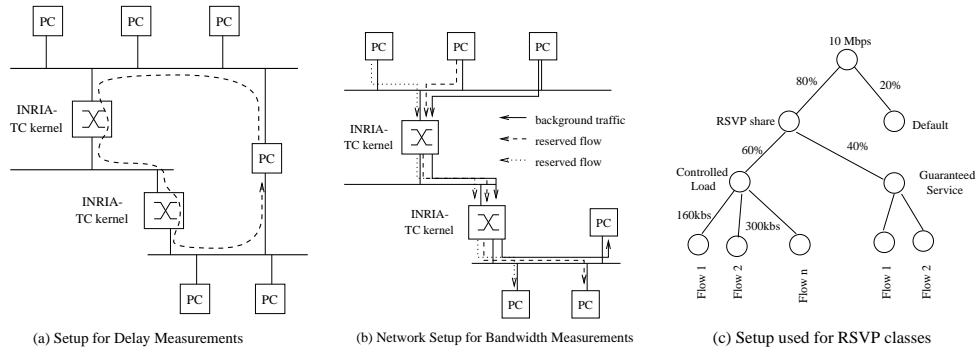


Figure 3: Network Setup for the Performance Evaluation

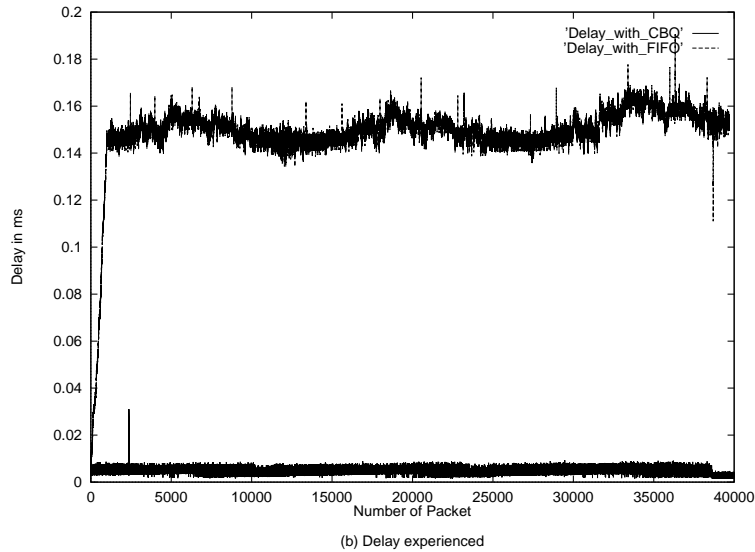


Figure 4: Absolute delay experienced by FIFO and CBQ scheduling

4.1 Delay Characteristics

In this section, we compare the packet delay distributions for a data flow under two scheduling mechanisms, FIFO and CBQ with a reservation for the flow. Therefore we use RSVP to create a controlled load session for the data packets. Since the controlled load service will be used by adaptive real-time applications using UDP, we measured the delay of a UDP session sending bursts about 4 packets in a row, each packet with a size around 500 bytes. The size of the packets are varied during the observation time.

We added a new class for the new flow to the CLS share tree node. This class allocated 200 kbyte/s. We measured the delay of the packets with one PC as sender and receiver. All packets are scheduled in both intermediate routers (compare Figure 3). To simulate a congested network we used the other PCs to generate the background traffic by sending TCP and UDP packets.

In Figure 4, we show the delay characteristics of the UDP packets from the measured flow. All packets of this flow traversing the two traffic control routers. We see that when we use FIFO scheduling in the routers, the worst case delay increases substantially. When the routers used our traffic control extensions, the delay remains stable. We also observe, that the delay variance of the un-scheduled FIFO packets is greater than the variance of the CBQ scheduled traffic.

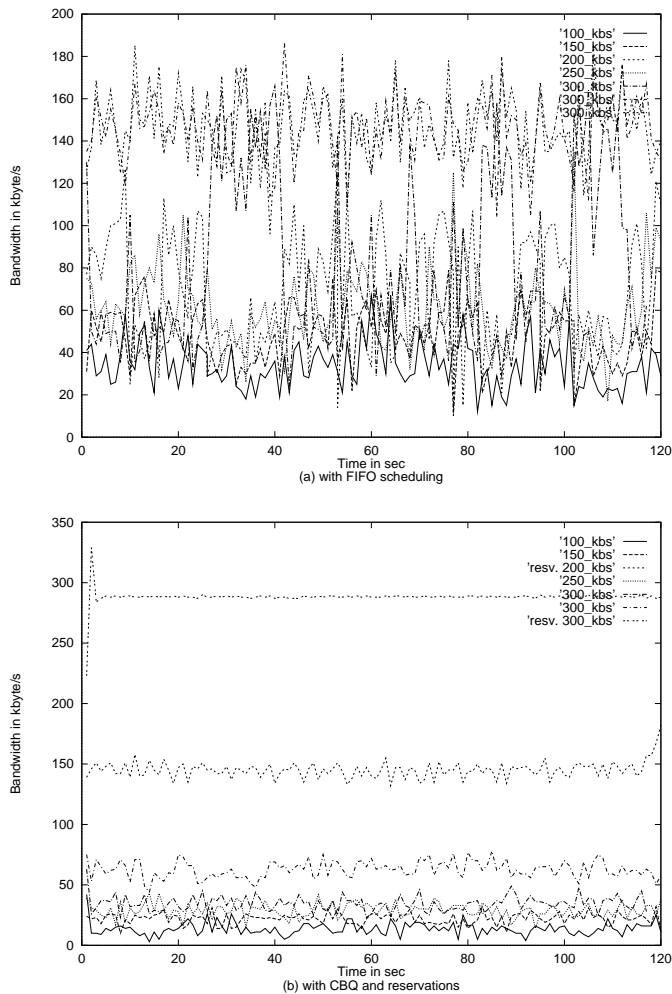


Figure 5: Measured Bandwidth with FIFO and CBQ Scheduling

4.2 Bandwidth Scheduling

In this section, we analyze how our traffic control implementation provides bandwidth reservation. For the bandwidth measurements we used the network setup illustrated in Figure 3. The setup for the scheduler classes is identical to the one we used for the delay measurements.

The advantages of bandwidth reservation is more visible when we reserve bandwidth for more than one flow. Therefore we created two sessions for two different flows. These flows requested a controlled load service. For the first session we reserved 300 kbyte/s and for the second 160 kbyte/s from the available bandwidth of the CLS share. Both sessions are sending UDP packets like in section 4.1. While session 1 sends 300 kbyte/s, session 2 sends more then the allocated bandwidth (200 kbyte/s instead of the allocated 160 kbyte/s).

The two new sessions are then added to the scheduler. Since the flow demanded a controlled load service they were linked to the CLS share class.

Figure 5 shows the bandwidth vs. time graphs for each UDP sessions under consideration. The sessions send with rates of 100 kbyte/s, 150 kbyte/s, 200 kbyte/s, 250 kbyte/s and three sessions with 300 kbyte/s. All senders together send about $\mu = 1600 * 8 = 12.8 Mbit/s$; that means that the $10 Mbit/s$ ethernet link is heavily loaded during the observation time.

Figure 5 shows the effective bandwidth used by each flow with an ordinary FIFO scheduling. In Figure 5 we show the results obtained with a reservation of 160 kbyte/s for the 200 kbyte/s (session 2) and a 300 kbyte/s allocation for one of the three 300 kbyte/s flows (session 1).

The observed results confirm what was expected. When the routers using a FIFO scheduling strategy all flows suffers under the congestion. The maximum rate a flow achieved was about 183 kbyte/s. The results with the reservations shows the advantages of sophisticated scheduling mechanisms. All reserved sessions kept their allocated bandwidth while the bandwidth of the non-reserved flows degrade to a minimum rate guaranteed by the default queue of the scheduler.

5 Conclusion

In this paper, we presented our approach towards Traffic Control in IP networks. We discussed how this was done and the experiences gained while implementing its elements.

While many papers present simulation results and theoretical approaches towards integrated services networks, we implemented and tested the behavior of this mechanisms in a real network. Our measurements confirmed that it is possible to guarantee a certain bandwidth for a limited number of sessions. For these sessions, we also achieved to keep a average delay on a low level.

This situation would change for a guaranteed delay service. When we use the equation described in section 3.3, the bandwidth would not be efficiently use. The classes for the guaranteed service would allocate too much of the available bandwidth.

In all our measurements, also with more than 2 routers and with a reservation of 200 kbytes per sec for that flow, the delay never raised above 40 ms. We achieved these results with our simple admission control. This confirms, that for the most applications much simpler mechanisms are sufficient to meet the application needs.

As mentioned before, RSVP is not the central element in our architecture. In contrast, we consider RSVP as a heavy, inflexible driving engine for reservation establishment, particularly for the weak service guarantees we can get (e.g. with guaranteed delay service).

In the future we intend to develop a light version of an Integrated Services Internet. We want to simplify the approach of the IntServ working group in terms of a reduced set of offered services. This architecture will also include something like a light version of RSVP. This architecture will include extensions that can provide discrimination in the service offered to different users in times of network congestion [4].

The main contribution of this work will be to realize the Integrated Services architecture with much simpler mechanisms then those proposed by the IntServ working group and also to integrate a pricing scheme from the start in the architecture.

References

- [1] Braden, R., Clark, D., and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", *RFC 1633, ISI, MIT, and PARC*, June 1994.
- [2] Braden, R., et al., "RSVP: A New Resource ReSerVation Protocol", *IEEE Network*, September 1993.
- [3] Clark, D., Shenker, S., Zhang, L. "Supporting real-time applications in an integrated services packet network: Architecture and Mechanisms", *Proceedings of ACM SIGCOMM'92*, pages 14-26, Baltimore, Maryland, August 1992.
- [4] Clark, D., "Adding Service Discrimination to the Internet", URL <http://anaweb.lcs.mit.edu/anaweb/papers.html>, September 1995.
- [5] Ferrari, D., Verma D., "A scheme for real-time channel establishment in wide area networks", *IEEE Journal on Selected Areas in Communication*, Vol. 8, 368-379, April 1990.

- [6] Floyd, S., "Link-sharing and Resource Management Models for Packet Networks", *Submitted to ACM/IEEE Transactions on Networking*.
- [7] Floyd, S., WWW page for CBQ, URL <http://www-nrg.ee.lbl.gov/floyd/cbq.html>
- [8] Huitema, C., "IPv6 The New Internet Protocol", *Prentice Hall, Upper Saddle River, New Jersey*, 1996.
- [9] Keshav, S., "On the Efficient Implementation of Fair Queueing", *Journal of Internetworking: Research and Experience*, V2, N3, September 1991.
- [10] Parekh, A. and Gallager, R., "A Generalized Processor Sharing Approach to Flow Control - The Single Node Case", *Technical Report LIDS-TR-2040*, Laboratory for Information and Decision Systems, MIT, 1991.
- [11] Shenker, S., Partridge, C., Guerin, R., "Specification of Guaranteed Quality of Service", *draft-ietf-intserv-guaranteed-svc-06.txt*, August 1996.
- [12] Wroclawski, J., Shenker, S., "General Characterization Parameters for Integrated Service Network Elements", *draft-ietf-intserv-charac-02.txt*, October 1996.
- [13] Wroclawski, J., "The Use of RSVP with IETF Integrated Services", *draft-ietf-intserv-rsvp-use-01.txt*, October 1996
- [14] Wroclawski, J., "Specification of the Controlled-Load Network Element Service", *draft-ietf-intserv-ctrl-load-svc-04.txt*, November 1996.
- [15] Zhang, L., "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks", *ACM Transactions on Computer Systems*, 9(2):101-124