1-2001

Rate-Matching Packet Scheduler for Real-Rate Applications

Kang Li
Oregon Graduate Institute of Science & Technology

Jonathan Walpole
Oregon Graduate Institute of Science & Technology

Dylan McNamee
Oregon Graduate Institute of Science & Technology

Calton Pu
Oregon Graduate Institute of Science & Technology

David Steere
Oregon Graduate Institute of Science & Technology

Citation Details

Kang Li ; Jonathan Walpole ; Dylan McNamee ; Calton Pu and David C. Steere, "Rate-matching packet scheduler for real-rate applications", Proc. SPIE 4312, Multimedia Computing and Networking 2001, 49 (December 22, 2000); doi:10.1117/12.410913; http://dx.doi.org/10.1117/12.410913

A Rate-Matching Packet Scheduler for Real-Rate Applications

Kang Li, Jonathan Walpole, Dylan McNamee, Calton Pu and David C. Steere*
Department of Computer Science and Engineering
Oregon Graduate Institute
{kangli, walpole, dylan, calton, dcs}@cse.ogi.edu

Abstract

A packet scheduler is an operating system component that controls the allocation of network interface bandwidth to outgoing network flows. By deciding which packet to send next, packet schedulers not only determine how bandwidth is shared among flows, but also play a key role in determining the rate and timing behavior of individual flows. The recent explosion of rate and timing-sensitive flows, particularly in the context of multimedia applications, has focused new interest on packet schedulers. Next generation packet schedulers must not only ensure separation among flows and meet real-time performance constraints, they must also support dynamic fine-grain reallocation of bandwidth for flows with variable-bit-rate requirements. Unfortunately, today's packet schedulers either do not support rate and timing sensitive flows, or do so with reservation systems that are relatively coarse-grain and inflexible.

This paper makes two contributions. First it shows how bandwidth requirements can be *inferred* directly from real-rate flows, without requiring explicit specifications from the application. Second, it presents the design, implementation and performance evaluation of a rate-matching packet scheduler that uses these inferred requirements to *automatically* and *dynamically* control the bandwidth allocation to flows.

1. Introduction

A packet scheduler is an operating system component that allocates the limited bandwidth of an outgoing network interface among competing flows. It manages interface buffers and controls which packet to send next. As the Internet evolves, applications such as Internet streaming video players that generate multimedia network flows, are expected to be widely supported. These applications are timing sensitive, and they require packet schedulers to support isolation among flows and allocate the right amount of bandwidth to meet their timing-constraints.

Today's packet schedulers either isolate timing sensitive flows from bursty ones by using priority schemes [3, 13], or they implement coarse-grain sharing of the bandwidth among competing flows using reservations [5, 6, 10 and 15]. Unfortunately, priority schemes do not take into account the timing requirements of the individual flow. Reservation-based schedulers do take timing requirements into account, but do not generally support fine-grain dynamic readjustments of reservations. The consequence of using static reservations is that bandwidth must be over-reserved based on the worst case requirement, which can cause significant inefficiency when variable bit-rate flows are present.

* This research was supported in part by DARPA contracts/grants N66001-97-C-8522 and N66001-97-C-8523, and by Tektronix Inc., and Intel Corporation.

In this paper we describe a RAte Matching Packet (RAMP) scheduler. Instead of asking a user to specify a resource-level reservation, such as a bit-rate value over a certain period, the RAMP scheduler automatically and dynamically infers bandwidth requirements and allocates bandwidth to match them.

The novel aspects of the RAMP scheduler are the way it infers bandwidth requirements from flows and the way it controls the allocation of bandwidth using them. It infers a flow's bandwidth requirements from per-flow state information in the packet scheduler. For instance, a per-flow queue *fill-level* in the packet scheduler indicates the mismatch between the amount of data that a flow[*] wants to send and the packet scheduler's bandwidth allocation to it. A feedback control system is used to monitor this queue fill-level and adjust the bandwidth allocation to the flow in order to keep the queue fill-level at a target level. In this way, variable bit-rate multimedia flows can run successfully without over-reserving bandwidth.

In this paper, in addition to quantitatively evaluating the RAMP scheduler we investigate the interactions between it and TCP congestion and flow control [8]. TCP congestion and flow control ensures that a flow's sending rate does not exceed the end-to-end network bottleneck or the flow's receiving rate. Thus, congestion and flow control may impose practical limits on the target bandwidth even though the flow might desire more bandwidth. In this case the RAMP scheduler does not attempt to allocate more bandwidth than can be used.

The paper is organized as follows. In Section 2 we first discuss the various bandwidth requirements of flows. Then we focus on a specific class of flows, called real-rate flows, and discuss the right bandwidth allocation for them. Based on the concept of the appropriate target bandwidth allocation for the real-rate flows, we explain the design of the RAMP scheduler in Section 3. Section 4 describes the implementation of the RAMP scheduler and section 5 provides an evaluation of it. In Section 6, we discuss the interactions between the RAMP scheduler and TCP's congestion and flow control. Section 7 discusses related work. Finally, Section 8 presents our conclusions and describes future work.

2. What is the Right Bandwidth Allocation?

In order to discuss the choice of bandwidth allocation for a flow, we classify flows according to their bandwidth requirement bounds. Our discussion focuses on *real-rate* flows [1]. Real-rate flows are flows with specific rate or throughput requirements in which the rate is driven by real-world demands. Examples of the real-rate flows are sensor-based monitoring systems and video or audio pipelines whose data is played at a certain frame rate over a network. A key characteristic of real-rate flows is that they only require enough bandwidth to match their rate requirements and will not use additional bandwidth even if it is made available to them. In contrast to real-rate flows, *greedy* flows try to acquire as much bandwidth as possible. A greedy flow does not have an upper limit on its required bandwidth. Examples of greedy flows are bulk data transfers, such as ftp and http applications.

The right bandwidth allocation for a real-rate flow is the amount of bandwidth that exactly matches its rate requirement. Over-allocation will result in unused bandwidth, whereas under-allocation will prevent the flow from meeting its real-rate target. For greedy flows it is convenient to assume that they accept whatever bandwidth allocation they get.

Beyond the application-level requirements, other factors such as the limits imposed by network congestion and flow control influence the right bandwidth allocation for a flow. Congestion control ensures that a flow's bandwidth does not exceed the end-to-end bottleneck of the connection. Similarly, flow control ensures that the sending rate for a flow does not exceed the rate at which the flow can be received. These two controls prevent a flow from congesting the network or overwhelming a slow receiver. Since the send-

*From a packet scheduler's point of view, transport layer protocol controls, such as TCP's congestion and flow control, are part of the flow, and hence contribute to the specification of the flow's requirements.

ing application is unable to utilize more bandwidth than the lower of the flow and congestion control limits, a packet scheduler's upper bound on bandwidth allocation for a flow is calculated as the minimum of these two limits and the bandwidth requirements of the flow.

Many present-day UDP-based video streamers either do not use a congestion control protocol, or use an application-specific congestion control scheme. In this case, the target bandwidth allocation in RAMP is simply the bandwidth being requested by the application. Given the consensus among the Internet community that congestion control is fundamental to the health of the Internet [24], we believe most UDP streaming applications will eventually use congestion control. Thus, in this paper, we focus our discussion on RAMP and its use by congestion-controlled real-rate flows. Furthermore, we assume that they use TCP-friendly congestion and flow control.

A packet scheduler controls the allocation to a flow according to available bandwidth at the local interface, and the factors discussed above. Briefly, the correct allocation to a flow is limited by the minimum of the flow's application-level requirement, its tcp congestion control limit, its tcp flow control limit, and the available interface bandwidth. Therefore, if enough local interface bandwidth is available, the scheduler should allocate bandwidth that matches TCP's output. If the local interface bandwidth is insufficient, then the scheduler should allocate bandwidth according to the available local interface bandwidth. Eventually, TCP congestion control will adapt to this bottleneck.

RAMP has an additional goal of maintaining fine-grain timing requirements for competing flows. RAMP uses two parameters, *proportion* and *period*, to specify bandwidth allocations for flows. Proportion is a percentage of the total bandwidth. Period specifies the interval over which the dispatcher guarantees the proportion value. The proportion value for a flow is derived according to the approach discussed above. RAMP implements proportions using a reservation mechanism that ensures isolation among flows. Period is related to the timing requirement of the flow. Currently, RAMP uses a system default period value for all flows.

3. The Design of the RAMP Scheduler

The RAMP scheduler, illustrated in Figure 1, is composed of three parts: (i) a requirement monitor that infers the mismatch between a flow's current bandwidth requirement and its current bandwidth allocation, and converts the mismatch to a pressure value for each flow[*], (ii) a feedback controller that uses these pressure values to adjust each flow's bandwidth allocation automatically, and (iii) a low-level packet dispatcher that supports lightweight, per-flow bandwidth reservations based on the values specified by the controller. A salient feature of the RAMP scheduler is its high-frequency feedback loop that links these components in order to continually monitor requirements and adjust allocations. Each of these components is described in the subsections below.

*Here the word "pressure" is borrowed from fluid control systems, in which higher mismatches result in higher pressures.

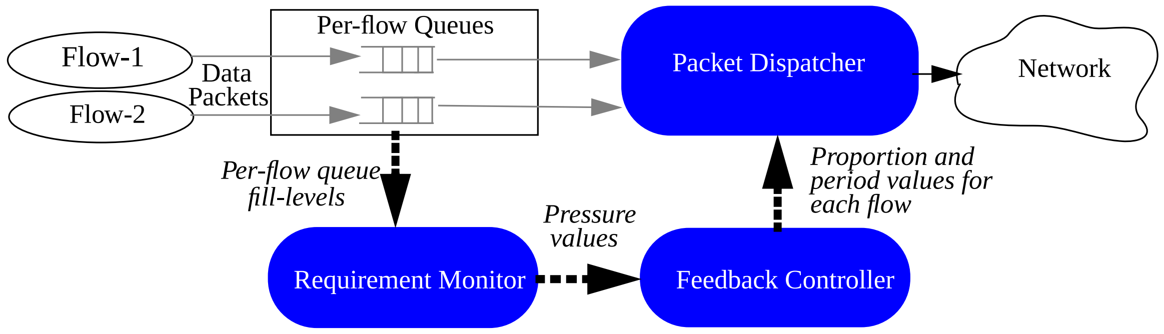

Figure 1: The Components of the Rate-Matching Packet Scheduler: the Requirement Monitor, the Feedback Controller and the Packet Dispatcher

3.1. The Requirement Monitor

The RAMP scheduler maintains a separate queue for each real-rate flow. The requirement monitor uses these queues to infer mismatches between current bandwidth requirements and allocations for each flow, and to monitor the scheduler-related delay for flows. When a flow uses TCP the input rate to the flow's queue is determined by TCP's control output. For UDP flows the input rate is determined directly by the flow's sending rate. The output rate from the queue is determined by the allocation RAMP gives to the flow. If these two rates match the fill-level of the queue will remain constant, and its value is an indication of the delay RAMP is imposing on the flow. If the fill-level is dropping then RAMP is overallocating bandwidth to the flow. Similarly, if the fill-level is rising RAMP is underallocating bandwidth for the flow. The control target for RAMP is to maintain a constant fill-level at a level determined by the flow's latency tolerance. Flows can either specify a target fill-level, based on their latency tolerance, or use the default value. The requirement monitor uses the difference between a flow's sampled queue fill-level and its target fill-level as the pressure, and sends these values to the feedback-based controller.

Not all the flows' requirements are derived in the way described above. We classify all flows into two categories according to the discussion in the Section 2. *Real-Rate* category is for real-rate flows, which include real-rate TCP flows and real-rate UDP flows [26, 27]. *Greedy* category is for greedy flows, which include greedy TCP flows and non-congestion-controlled non-real-rate UDP flows. The above monitoring mechanism applies directly to flows in Real-Rate Category. For flows in Greedy Category, the requirement monitor treats all of them together as a single real-rate flow, whose minimum allocation is based on a system default requirement value. In other words, real-rate flows are treated preferentially and greedy flows compete for the remaining bandwidth in the same way they do in conventional systems. The current prototype of the RAMP scheduler needs users to specify the type (Real-Rate or Greedy) of flows. By default, the RAMP scheduler classifies a non-type-specified flow into Greedy category.

3.2. The Feedback Controller

The controller takes the Requirement Monitor's results (the pressures) combines them with stored history information, and uses standard control components to match the bandwidth allocations to the flows' requirements. Those control components are widely used in classical control systems. By adjusting the bandwidth allocation to keep the per-flow queue fill-level at a target fill-level, the packet scheduler matches the queue's consumption rate to the rate at which the flow is producing data.

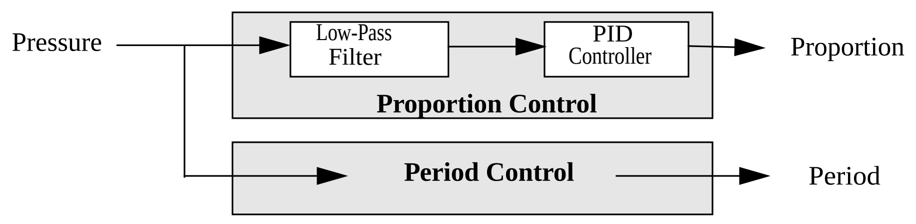

Figure 2: The Feedback Controller

Figure 2 shows the structure of the controller's control circuit for one real-rate flow. The controller takes the per-flow queue's pressure as input, and sends it to a low-pass filter. Low-pass filters give an exponential averaging over the past history and are generally used in control systems to smooth the input to avoid generating oscillations in the controller's output. An aging parameter *L* to the Low-Pass Filter defines the weight on the current input. The value *1 - L* is the weight given to past history.

The low-pass filter's output connects to a *proportional-integral-derivative* (PID) control component. PID controls are widely used in many control applications to provide acceptable response together with stability and damping. It combines the proportion of the summed pressures with the integral and the derivative of the pressures. It has parameters *P*, *I*, and *D*, which control the weight of each part in the output. The proportion part P contributes to the output based on the instant error between the monitored fill-level and the target fill-level. Thus, parameter-P decides how fast the allocation catches up with the requirement. The integral part I contributes to the output based on the accumulated error. Thus, parameter-I decides how fast the allocation erases the accumulated requirement-allocation-mismatch and reaches a stable state. The derivative part D slows down the output reaction in proportion to the most recent pressure variation. Thus the parameter-D decides the damping factor of the control reaction, which avoids the over allocation of bandwidth in the case of a sudden change in a flow's requirement.

In the RAMP scheduler, if the allocation reacts too slowly, the scheduler will inhibit TCP's ability to probe for additional bandwidth. On the other hand, if the allocation reacts too fast, the scheduler may over allocate bandwidth.

The output of the PID is the proportion value, which specifies the data rate required by the flow. Another aspect of the bandwidth allocation is the period, which is the minimum temporal granularity over which the bandwidth allocation must be enforced. Dynamic period adjustments are not implemented in our current prototype, which assigns the same period to all flows.

3.2.1. Overload Policy

So far the discussion has assumed that the RAMP scheduler interface is only partially loaded. In this subsection we address the overload issue. Since we treat the greedy flows as a single real-rate flow, the interface is only truly overloaded when the sum of all the real-rate requirements exceeds the total bandwidth. Otherwise, RAMP can always allocate bandwidth to meet these requirements before allocating the remain bandwidth to greedy flows.

When the total of the real-rate flows' requirements exceeds the interface's capacity, it is not possible to meet all requirements. In this case, the flows must adapt their bandwidth consumption requirements either by reducing the rate at which they submit packets, or by dropping packets. Both cases represent a quality adaptation policy, and the policy could be flow-specific. The RAMP scheduler detects this overload condition early enough to deliver a QoS exception to the application. Applications can use QoS exceptions to perform adaptation early, thus, reducing their requirements. Further discussion of QoS adaptation approaches are beyond the scope of this paper, but are discussed in [2].

3.3. The Packet Dispatcher

The lowest level of the RAMP scheduler is a reservation-based packet dispatcher. The packet dispatcher implements the bandwidth allocations that are assigned by the feedback controller and determines the global order of packet transmission through the interface. Reservation requirements to the packet dispatcher are described by two parameters: proportion and period. Proportion is a percentage of the total bandwidth. Period specifies the interval over which the dispatcher guarantees the proportion value. Within a period, the dispatcher can either send a flow's packets in a single burst or send them smoothly. The RAMP scheduler's packet dispatcher implements an earliest deadline first (EDF) policy [9], in which the end of period is the deadline value for each flow (see section 4).

4. Implementation

The RAMP scheduler is implemented as a component of an end-host* operating system. Most of it is implemented in the Linux 2.0.35 operating system kernel. The only exception is the feedback controller, which is currently implemented at user-level to allow rapid prototyping using the SWiFT software feedback toolkit [7]. SWiFT supports an approach to building adaptive system software based on control theory. It supplies basic control components such as the control blocks shown in Figure 3, and the feedback controller is built with these components. The controller's PID component and low-pass filter component have parameters that can be set to define the behavior of the controller. For the evaluation described in this paper, we use the set (P=0.35, I=0.005, D=0.01) as the PID component's setting, and (L=0.5) for the low-pass filter's setting.

Because of the user-level implementation of the controller, monitoring is implemented using system calls which expose the queue fill-levels. The requirement monitor samples the queue fill-levels by calling the monitoring system calls and produces the pressures. The sampling frequency in the prototype system is 100HZ. Since these system calls are made frequently, the user-level implementation is a source of inefficiency, and we plan to ultimately move the controller into the kernel.

The packet dispatcher enforces the bandwidth allocations specified by the controller. Since the built-in packet scheduler in Linux 2.0.35 does not support bandwidth reservation, we replaced it with our own reservation-based packet dispatcher. In contrast to the Linux built-in packet scheduler, our packet dispatcher manages a queue for each flow, or more accurately, for each socket. When a flow creates a socket, the packet dispatcher creates a per-flow queue and attaches it to the socket†. Figure 3 illustrates the relationship between the flows and the packet dispatcher. Flow-1 and Flow-2 use TCP sockets. TCP controls when the packets go from the socket buffer to the per-flow queue in the scheduler. Flow-3 uses a UDP socket. Since UDP sockets do not accumulate data (no socket buffers), their packets directly enter the per-flow queues in the scheduler. The packet dispatcher selects packets from these per-flow queues based on each flow's proportion and period values that are set by the controller. Then it sends the selected packet to the interface queue. If the interface queue is full, the dispatcher is stalled until free space is available. Packets are removed from the interface queue and sent to the network by device interrupts.

Conceptually, the packet dispatcher is driven by timer events. However, in practice, explicit packet related events occur more frequently. RAMP's packet dispatcher is driven by device interrupt events and by sys-

*The RAMP scheduler does not use any specific end-host information, and the algorithm could be used in router operating systems. However, there are scalability issues associated with supporting per-flow states in current router operating systems.

†Linux kernels use the *sk_buff* structures to manage network queues in a similar way to the *mbuf* structures [16] in BSD UNIX, which avoid data copying when moving data in the kernel. Thus, adding per-flow queues will not increase data copying costs in this implementation.

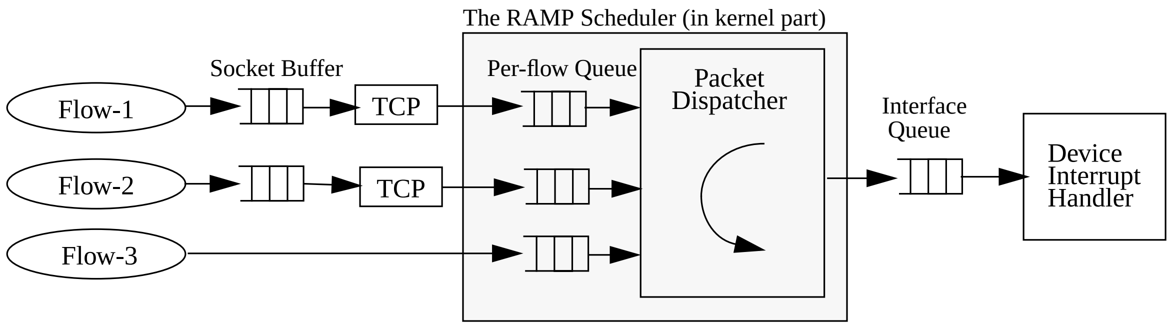

Figure 3: The Packet Dispatcher in an End-host's OS

tem events such as *ip_output* that happen when a packet enters its corresponding per-flow queue. When none of the above system events happen, we use a system timer to ensure that the dispatcher runs at least every 0.1msec. If some system events happen before the system timer fires, we reset the system timer to another 0.1msec. This approach is similar to the one described in [32]. Our prototype system's CPU tick size is set to 0.1 msec, thus the smallest time interval we get from the system is 0.1msec.

Since we have not yet completed the dynamic period control component of the scheduler, the period allocation for each flow is currently set to 0.1 msec, which, being dependent on the system timer, is the smallest possible value in our implementation. We plan to implement dynamic period control [4] in the future.

5. Evaluation

Two sets of experiments are designed to evaluate the RAMP scheduler. One evaluates RAMP's benefits and capabilities, and is discussed in this section. The other explores the interactions between RAMP and TCP's congestion and flow control, and is discussed in Section 6.

In order to validate the benefits of the RAMP scheduler, we conducted experiments to: (1) show that it is able to separate flows and protect a congestion-controlled real-rate flow from a UDP-based greedy flow; and (2) test it's ability to dynamically detect changes in the requirements, and use them to dynamically adjust allocations. In order to do these experiments, we used two benchmark applications. The first is the Quasar video player [28]. It plays a video clip through a network using TCP. The second was constructed to allow tighter control over the bandwidth requirements. This benchmark application also sends data across the network through a TCP socket. Users of the benchmark application can set its bandwidth requirement. By changing the requirement and monitoring the resulting allocation, we can evaluate the responsiveness and accuracy of the RAMP scheduler.

All experiments were run on two 400Mhz Pentium II machines, both with 128MB of memory and 3c905 Ethernet Cards. The sender machine ran our modified version of Linux 2.0.35 with our RAMP scheduler, and we set the Ethernet interface to work in the 10Mbit/s mode. The receiver machine ran Linux 2.0.35 without our packet scheduler[*], and the Ethernet interface worked in the 100Mbit/s mode to ensure it is not the bottleneck. A 100Mbit/S Ethernet switch connected the sender machine and the receiver machine. We disable the controller's dynamic period control and focus on the proportion allocation. In the following experiments, all flows' period allocations are assigned to 0.1msec. The target fill-levels used by the RAMP scheduler are all 16Kbytes, and the scheduler samples queue fill-levels every 10msec.

*Our experiments are designed to model the use of the RAMP scheduler in an environment that is mixed with other schedulers, such as FCFS schedulers. The motivation behind this approach is two folds: because of two reasons: (1) We assume only one bottleneck in the system, and a RAMP scheduler in an unloaded interface performs similarly to a FCFS scheduler; (2) Even in the future, FCFS schedulers are likely to exist in most systems [16,17].

5.1. Bandwidth Sharing Result with UDP Cross Traffic

This experiment evaluates RAMP's ability to protect real-rate flows in the presence of competing workload. In this experiment, we demonstrate the bandwidth sharing result of the RAMP scheduler when a TCP video flow (in this case generated by the Quasar player) competes for bandwidth with UDP flows, and compare it with the well-known bandwidth sharing results for FCFS packet schedulers applied to the same situation.

The target of this experiment is to show the effect of competing UDP traffic on the real-rate TCP traffic. We show this by comparing the bandwidth usage of the real-rate TCP under three cases: (a) the real-rate TCP runs through a FCFS scheduler with no other competing traffic; (b) the real-rate TCP runs through a RAMP scheduler with competing greedy UDP traffic; (c) the real-rate TCP run through a FCFS scheduler with competing greedy UDP traffic. Since RAMP derives the real-rate TCP's requirement and dynamically reserves bandwidth for it, we expect the TCP flow under RAMP to receive the same bandwidth it gets when no competing traffic exists.

In order to measure the effectiveness of RAMP's isolation for real-rate flows, we calculate the accumulated bandwidth usage of the real-rate flow in the three cases. We use its accumulated usage in case (a) as the requirement, and compare it to the allocations given by the RAMP and the FCFS scheduler. Figure 4 shows the accumulated error in allocation for a FCFS scheduler and a RAMP scheduler. Line *RAMP* in Figure 4 remains close to 0 as time increases, indicating that the bandwidth allocation meets the requirement. The *FCFS* leaves 0 as time increases, indicating that the allocation with a FCFS scheduler under the same condition does not keep up with the requirement. In summary, this first experiment shows that RAMP is able to implement traffic separation for dynamically varying real-rate flows.

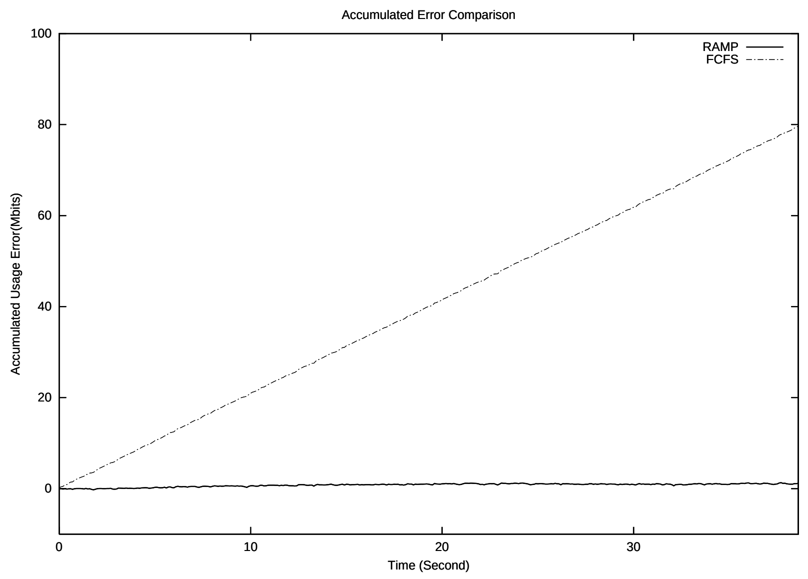

Figure 4: Bandwidth sharing for a real-rate video flow and competing traffic.

5.2. Response to Dynamic Changes in Bandwidth Requirements

In this subsection, we show the ability of RAMP to dynamically detect changes in the flow's requirement and use them to adjust the bandwidth allocation. The experiment is designed to let us control a flow's bandwidth requirements, and log the RAMP scheduler's bandwidth allocation. By comparing the allocation with the requirement, we can quantify how well the RAMP's allocation automatically tracks the flow's requirement. The details of the experiment are as follows: We run our benchmark application on an otherwise idle network. The sender side network interface is controlled by the RAMP scheduler. The benchmark application varies its rate requirements according to a step function (with steps up and down). This step function serves as the input for the experiment. We log the per-flow queue fill-levels, which are the inputs to the RAMP scheduler, and the bandwidth allocation, which is the output of the RAMP scheduler.

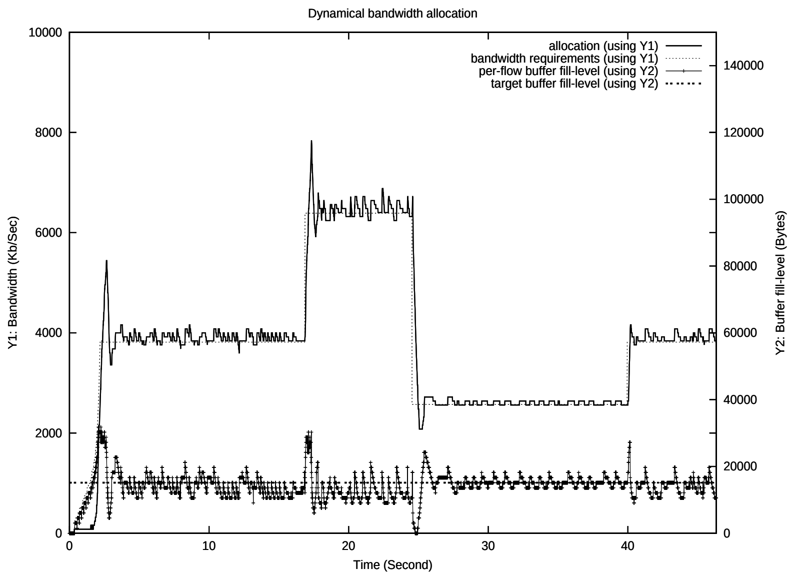

Figure 5: RAMP scheduler's allocations versus the benchmark application's requirements variations

Figure 5 shows the result of this experiment. The first observation is that the per-flow queue fill-levels expose the flow's bandwidth requirement information. For example, the per-flow queue's fill-level starts to increase at the time the flow increases its requirement. The RAMP scheduler uses the fill-level information to drive the bandwidth allocation to match the flow's requirement. As shown in the figure, RAMP responds to the change in the queue fill-level variations by rapidly adjusting the allocation. The experiment result matches our expectations: the RAMP scheduler's allocation roughly follows the benchmark application's requirement variations. However, notice that RAMP over-reacts to the step changes and takes some time to stabilize.

In classic control systems, a system's behavior can be described by its step-input response character, which is defined by its response time[*], settling time[†] and overshoot[‡] [35]. Step-input is the worst possible change in the input signal because it contains infinitely high frequencies. In practice, variations in the input signal will be better than this. According to control theory, once the above character values are known, we can predict the system's behavior to any other input. In order to quantify the characteristics of the RAMP scheduler, we amplify the response of the RAMP scheduler to a step-input in Figure 6. Figure 6 shows that: for the current settings of the PID control component RAMP's response time is about 150 msec, its settling

*The response time refers the period from the time that the allocation first reaches 10% of the new requirement value to the time that the allocation first reaches 90% of the new requirement value.

†The settling time refers the time from the requirement value change to the time that the allocation first reaches a stable value (the absolute error with the average is within the range of 5% of the average).

‡The overshoot refers the maximum amount of bandwidth over-allocated (under-allocated if the requirement decreases) to the flow.

time is about 1000 msec, and its average overshoot is 65% of the step height. The significance of overshoot for RAMP is that it will attempt to allocate more bandwidth than is really needed. One way to deal with this is to ensure that the interface is always under utilized in order to allow the overshoot. The problem with this approach is its inefficiency, especially considering the fact that the measured overshoot with RAMP's current settings is 65% of the step size. This approach would require us to operate at only 1/1.65 of the available interface bandwidth, which is not acceptable. An alternate approach is to allow RAMP to temporarily over-allocate the interface bandwidth, which would cause an accumulation of packets in the device buffer (the interface queue in Figure 3). The size of this accumulation is dependent on the rate-variation characteristics of the input signal. For the step-input in Figure 6, the maxumum accumulation is determined by the difference bwtween Area-B and Area-A, which introduces less than 50 msec delay to the device buffer.

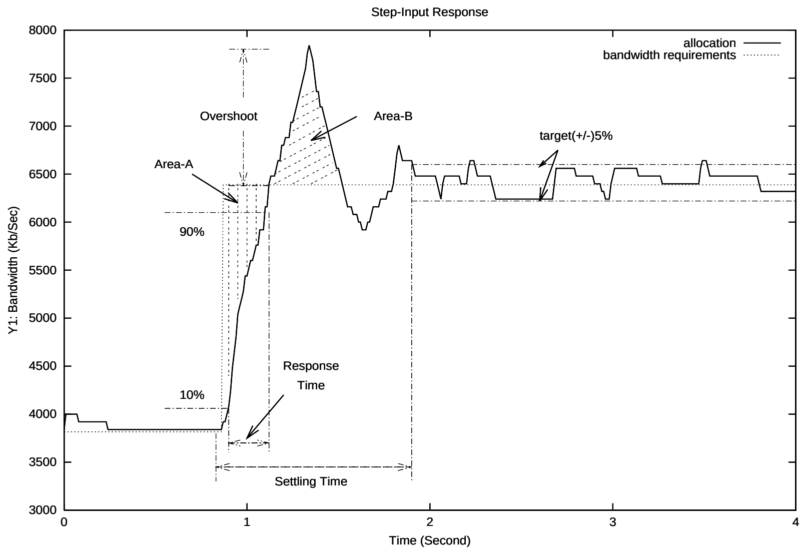

Figure 6: RAMP scheduler's step-input response

6. Interactions between the RAMP Scheduler and TCP

RAMP automatically adjusts local interface bandwidth allocation according to the TCP's output. However, TCP adapts its output rate according to the available end-to-end bandwidth, which can be limited by the local interface bandwidth allocation controlled by RAMP.

To study this interaction, we consider TCP's flow control and congestion control separately. We discuss the flow control first in subsection 6.1, then show the congestion control experiment and discuss the overall interaction in subsection 6.2.

6.1. TCP Flow Control and the RAMP Scheduler

TCP flow control imposes a limit on the amount of data that can be sent without having been acknowledged, in order to ensure that the receiver-side buffer will not overflow. The amount of data that can be sent at any time corresponds to the difference between the available buffer space of the receiver and the amount of unacknowledged data that is still in the network. Since the RAMP scheduler's queues are no different to router queues in this respect, data in them are considered as data in transition in the network. Hence the RAMP scheduler can not cause the receiver side buffers to overflow. Furthermore, the output rate adjustments caused by TCP flow control are no different to the rate adjustments caused by other application-level requirement variations as far as RAMP is concerned. Thus, the bandwidth allocation by the RAMP scheduler tracks the TCP flow control output automatically without interacting with it. We verified this result through experiments, which are presented in [33].

6.2. TCP Congestion Control and the RAMP Scheduler

The congestion control experiment is designed to show the interaction between TCP and RAMP when there is an end-to-end bottleneck other than the RAMP controlled local interface. When the end-to-end bottleneck changes its rate, TCP adapts its output rate to this end-to-end bottleneck rate. In this experiment we verify whether the RAMP scheduler limits TCP's ability to detect the available end-to-end bottleneck rate. The details of this study include: (a) verifying whether TCP is still able to probe for more bandwidth when it is available; (b) verifying whether RAMP causes additional timeouts by its potential effect on the TCP round-trip-time; and (c) verifying whether TCP's congestion window expands in the same way with a RAMP scheduler as it does with a FCFS scheduler.

In order to study the interaction, we simulate a congestion link and control its bandwidth variations. Then we use it to run a real-rate TCP flow with a RAMP scheduler and a FCFS scheduler respectively, and compare TCP's congestion window and timeout behaviors in the two cases.

The detailed steps of the experiment are: (a) We simulated a bottleneck link based on a leaky bucket model [31], which defines a link by a queue size and an output rate. We can control the queue size and the output rate through system calls, which specify the character of the bottleneck according to the parameters of the leaky bucket model. By controlling the bottleneck link parameters, we create a congestion generator. (b) The bottleneck is implemented as an additional queue between the packet scheduler and the interface queue shown in Figure 3. Thus, it is viewed as a network link by both TCP and the RAMP scheduler. (c) We run the real-rate TCP flow used in Section 5 with a FCFS packet scheduler. We start with a bottleneck rate larger than the real-rate flow's requirement, adjust the bottleneck rate down and up and log the TCP's congestion window behavior. (d) We repeat the experiment again, using the RAMP scheduler to control the bandwidth allocation instead of the FCFS scheduler. The queue size is set to 6K bytes throughout the experiment, and the bottleneck rate varies from 2Mb/Sec to 1.5Mb/Sec and 0.5Mb/Sec, and then goes back to 1.5Mb/Sec and 2Mb/Sec.

Figure 7 and Figure 8 show the result of this experiment. Figure 7 shows the result of TCP's congestion window size's behavior under the FCFS packet scheduler. When the bottleneck rate reduces from 2Mb/Sec to 1.5Mb/Sec, the congestion window size drops and keeps the sawtooth shape because of TCP congestion control's slowstart and exponential backoff behavior. When the bottleneck rate reduces further, TCP's congestion window maintains a similar sawtooth shape but varies with lower frequency because of TCP's self-clocking. Figure 8 shows the congestion window size's behavior under the RAMP scheduler, which also traces the bottleneck rate variations. Figure 7 shows that TCP's congestion window reaches a larger size with the RAMP scheduler (window sizes reach 25~30) than with the FCFS scheduler (window sizes reach 10). It also shows the average throughput with the two schedulers are close to each other. However, TCP expands its congestion window less aggressively with the RAMP scheduler than with the FCFS scheduler. This behavior is illustrated by the slowly varying sawtooth shape in Figure 8 (in the range of 0.4~0.8 saw-tooth/sec), compared to the rapid variations in Figure 7 (in the range of 0.94~2.26 saw-tooth/sec).

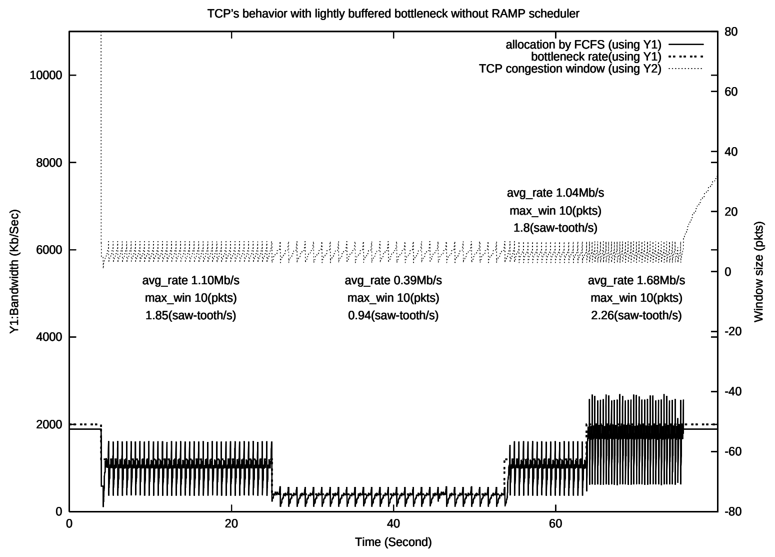

Figure 7: TCP's behavior with a FCFS scheduler

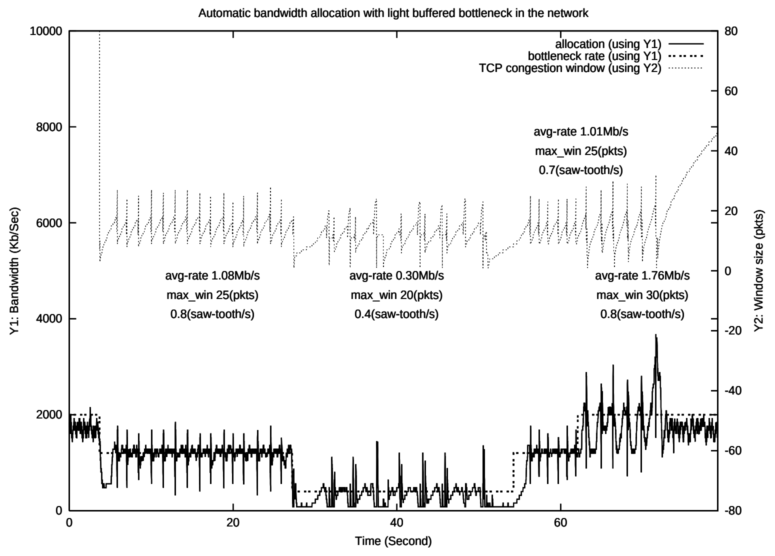

Figure 8: TCP's behavior with a RAMP scheduler

The results also show that the RAMP scheduler adds delay to the data transmission. TCP's round-trip-time is increased in average by 400ms, 600ms and 1000ms respectively for the bottleneck rate 2Mb/Sec, 1.5Mb/Sec, and 0.5Mb/Sec. The increment is mainly attributed by the buffering in the RAMP scheduler. However, no TCP-retransmission timeouts happen in either experiment. The reason is that TCP uses an adaptive timer to estimate the timeout period based on the monitored round-trip-time. Thus, even with the increase in the round-trip-time, all of the packet loses are still detected by duplicated acknowledgements before any TCP-retransmission timeout happens. The delay introduced by RAMP is different when the bottleneck rate changes. The reason of it is that current RAMP scheduler choose a constant bytes value as the target fill-level. In the future, we plan to use a constant time based fill-level, which will introduce same delay to TCP even when the bottleneck rate changes.

The results answer several of the questions raised earlier: (a) According to Figure 8, TCP with RAMP is still able to probe for the end-to-end bottleneck rate, and in average reaches same output rate as TCP with FCFS does. (b) According to the experiment log, RAMP scheduler did add rate dependant latency, but it didn't add any additional timeouts to TCP. (c) Clearly the congestion window expands and contracts with a different shape when TCP flows run through a RAMP scheduler. Using RAMP the congestion window expands slowly, but with more overshoot. The reason for this behavior is that RAMP predicts the TCP's requirement and allocates the bandwidth based on it. However it introduces delay and smoothing, whereas the FCFS scheduler immediately gives all the bandwidth allocation to the flow when there is no other competing traffic. According earlier research on TCP throughput [19, 24, and 34], long-delay and smoothed TCPs expand their window slowly compared to short-delay and bursty TCPs. The RAMP scheduler's delay and smoothing impact on TCP is determined by the controller's target queue fill-level and its responsiveness. We believe adjusting the target fill-level and responsiveness of the controller is a way of tuning the bandwidth sharing result for RAMP and TCP. In the near future, we plan to demonstrate these effects.

7. Related Work

We divide the related work into in several fields, including packet scheduling, multimedia transmission, network congestion and flow control, and scheduling for other resources.

We have demonstrated the advantage of the RAMP scheduler over FCFS and simple reservation based packet schedulers. Priority-based schedulers [3, 12, and 13] are similar to RAMP in the sense of separating flows. However, RAMP differs significantly from priority-based schedulers because it derives flow bandwidth requirements and supports proportional sharing based on these requirements. Priority-based schedulers have no knowledge of flow requirements, thus they can only do coarse grain allocations based on priorities, and they are subject to starvation. Another closely related approach is packet scheduling based on Fair Queueing [5]. Fair Queueing schedulers share interface bandwidth according to some notion of "fairness", and either allocated bandwidth equally to flows or based on some pre-specified weights. The RAMP scheduler can be classified as one kind of weighted-fair-queueing scheduler in which the weights are derived automatically based on inferred bandwidth requirements. Linux2.2 kernels supply a powerful packet scheduler infrastructure [6], allowing them to support several kinds of packet schedulers. However, none of these packet schedulers infer flow bandwidth requirements and use them to dynamically allocate bandwidth.

Media streaming and smoothing techniques [27, 30] smooth the bandwidth requirements for stored video by buffering data at the application level. These approaches make it easier to deploy multimedia applications in the Internet. The RAMP scheduler addresses the same problem at a different level. It smoothes streaming data at the network resource level. An advantage of smoothing at this level is that it allows isolation of real-rate flows from competing greedy flows.

The RAMP scheduler is also related to research on active queue management [18, 19, 20, 21, 23, 24, and 25]. Active queue management research is concerned with the interaction between scheduling mechanisms, such as RED [23] and ECN [18], and network congestion control. The RAMP scheduler can also be viewed as one form of TCP pacing [34]. The TCP pacing effect can be controlled by adjusting the responsiveness of the RAMP controller. However, active queue management research does not address the problem of automatically inferring a flow's bandwidth requirement.

The RAMP scheduler is designed to support real-rate flows at the resource management level. It is related to research on transport layer protocols for real-rate flows, such as SCP [26] and RAP [27], which detect the available bandwidth and trade reliability for less delay to meet the flows' timing requirements. We believe that successfully deployment of real-rate flows in the Internet needs coordinated support from both levels.

Finally, the RAMP scheduler uses a feedback-driven control system to manage network interface resources. Feedback-driven resource management has been successfully applied to other resources such as CPU [1, 14], and disk bandwidth [11].

8. Contributions and Future Work

This paper has described a packet scheduler that can automatically derive flow bandwidth requirements and dynamically adjust bandwidth based on them. The main contribution of the RAMP scheduler is that it uses a feedback-based controller to allocate interface bandwidth to flows without requiring them to communicate their explicit bandwidth requirements. The RAMP scheduler infers the flow requirements from queue state information and consequently moves part of the burden of real-rate application programming to the resource management level. Our experimental evaluation of RAMP demonstrates its ability to separate traffic, and dynamically respond to varying real-rate requirements.

The experiment results also illustrate some interesting interactions between TCP and RAMP. We plan to further characterize the RAMP scheduler and TCP congestion and flow control using feedback control models, in order to explore these interactions more thoroughly in the future. Other future work includes extending the system to control the period allocations, and investigating its effect on end-to-end delay.

9. References

[1] David C.Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu and Jonathan Walpole, “A Feedback-driven Proportion Allocator for Real-Rate Scheduling”, In Proceedings of *Operating Systems Design and Implementation (OSDI)*, *ACM Operating System Review, Special Issue-Winter 1998*, Page 145-258.

[2] Charles Krasic, Jonathan Walpole, Mark Jefferys, Dylan McNamee, David Steere and Calton Pu. “Dynamic QoS Adaptation in Shared Heterogeneous Environments.” OGI-Tech-Report-CSE-99-011, available at ftp://cse.ogi.edu/pub/tech-reports/1990/.

[3] Sally Floyd. Notes on CBQ and Guaranteed Service. Draft document, July 1995. Available at http://www.aciri.org/floyd/cbq.html.

[4] David C. Steere, Joshua Gruenberg, Dylan McNamee, Calton Pu and Jonathan Walpole. “Fine-grain Period Adaptation in Soft Real-Time Environments”. OGI-TECH-REPORT-CSE-99-012, available at ftp://cse.ogi.edu/pub/tech-reports/1999/.

[5] Alan Demers, Srinivasan Keshav, and Scott Shenker. “Analysis and Simulation of a Fair Queueing Algorithm”. In *Proceeding of ACM SIGCOMM'89*.

[6] Werner Almesberger. “Linux Traffic Control -- Implementation Overview”. Available at http://icawww1.epfl.ch/linux-diffserv/.

[7] Ashvin Goel, David Steere, Calton Pu, and Jonathan Walpole. “SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit”. Technical Report CSE-98-009, Department of Computer Science and Engineering, Oregon Graduate Institute. June 1998.

[8] Van Jacobson, and Michael Karels. “Congestion Avoidance and Control”. *Proceeding of ACM SIGCOMM’88*, pages 79-88, August 1988.

[9] C.L.Liu and J.W.Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment” *JACM*, Vol.20, No.1, pages 46-61, 1973.

[10]David D. Clark, Scott Shenker, and Lixia Zhang. “Support Real-Time Applications in an Integrated Service Packet Network: Architecture and Mechanism”. *Proceeding of ACM SIGCOMM’*92.

[11]Dan Revel, Dylan McNamee, Calton Pu, David C.Steere, and Jonathan Walpole. "Feedback-based Dynamic Proportion Allocation for Disk I/O". OGI-TECH-REPORT-CSE-99-001, available at ftp://cse.ogi.edu/pub/tech-reports/1999/.

[12]Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter. "An Architecture for Differentiated Services". RFC 2475. December 1998.

[13]Philip Almquist. "Type of Service in the Internet Protocol Suite". RFC1349, July 1992.

[14]John A.Stankovic, Chenyang Lu, Sang H.Son and Gao Tao. "The Case for Feedback Control Real-Time Scheduling". *EuroMicro Conference on Real-Time Systems*, June 1999.

[15]Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. "RSVP: A New Resource ReSerVation Protocol". *IEEE Network*. September 1993.

[16]Marshal Kirk McKusick, Keith Bostic, Michael J.Karels and John S.Quarterman. *The Design and Implementation of the 4.4 BSD Operating System*. Addison-Wesley, 1996.

[17]Linux2.0 kernel source code. Available at http://www.kernelnotes.org/.

[18]B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang. "Recommendations on Queue Management and Congestion Avoidance in the Internet". RFC2309, April 1998.

[19]S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking*, vol.1, pp.397-413, August 1993.

[20]B. Suter, T. V. Lakshman, D. Stiliadis, and A. K. Choudhury. "Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-flow Queueing". *IEEE Communications*, August, 1999.

[21]Pittsburgh Supercomputing Center. "Enabling High Performance Data Transfers on Hosts". Available at http://www.psc.edu/networking//perf_tune.html.

[22]V. Jacobson, R. Braden, and D. Borman. "TCP Extensions for High Performance", RFC1323, May 1992.

[23]S. Floyd. The RED Web Page. http://ftp.ee.lbl.gov/floyd/red/html.

[24]S. Floyd and K. Fall. "Promoting the use of end-to-end congestion control in the Internet". *IEEE/ACM Transactions on Networking*, August 1999.

[25]S. Floyd. "TCP and Explicit Congestion Notification". *ACM Computer Communication Review*, V.24 N.5, October 1994, p.10-23.

[26]S. Cen, C. Pu, J. Walpole. "Flow and Congestion Control for Internet Media Streaming Applications". *Proceedings of Multimedia Computing and Networking (MMCN)*, January 1998.

[27]R. Rejaie, M. Handley and D. Estrin. "RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet". *IEEE INFOCOM'99*.

[28]Quasar video player. Available at http://www.cse.ogi.edu/sysl/.

[29]L. Zhang, S. Shenker, and David D. Clark. "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two Way Traffic". *Proceeding of ACM SIGCOMM'91*.

[30]Wu-chi Feng, Brijesh Krishnaswami, Arvind Prabhudev. "Proactive Buffer Management for the Streamed Delivery of Stored Video". *Proceeding of ACM Multimedia'98*. Bristol. UK.

[31]Srinivasan, Keshav. *An Engineering Approach to Computer Networking*. page 237~240. Addison Wesley Longman, Inc. 1997.

[32]Mohit Aron, Peter Druschel. "Soft timers: efficient microsecond software timer support for network processing". 17th ACM Symposium on Operating Systems Principles. Published as *Operating Systems Review*, 34(5), page 232-246, Dec. 1999.

[33]RAMP Experiments Web Page. http://kang.cse.ogi.edu:80/syslnet/archieved.htm.

[34]A. Aggarwal, S. Savage, T. Anderson. "Understanding the performance of TCP Pacing". *IEEE INFOCOM'2000*.

[35]Gene F. Franklin, J. David Powell. *Feedback control of dynamic systems*. Addison-Wesley Inc. 1994.