Submitted for publication, May 1997.

Understanding and Improving TCP Performance over Networks with Minimum Rate Guarantees

Wu-chang Feng† Dilip D. Kandlur‡ Debanjan Saha‡ Kang G. Shin†

†Department of EECS
University of Michigan
Ann Arbor, MI 63130
{*wuchang,kgshin*}*@eecs.umich.edu*

‡Network Systems Department
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
{*kandlur,debanjan*}*@watson.ibm.com*

Abstract

A large number of Internet applications are sensitive to overload conditions in the network. While these applications have been designed to adapt somewhat to the varying conditions in the Internet, they can benefit greatly from an increased level of predictability in network services. We propose minor extensions to the packet queueing and discard mechanisms used in routers, coupled with simple control mechanisms at the source that enable the network to guarantee minimal levels of throughput to different sessions while sharing the residual network capacity in a *cooperative* manner. The service realized by the proposed mechanisms is an interpretation of the controlled-load service being standardized by the IETF. Although controlled-load service can be used in conjunction with any transport protocol, our focus in this paper is on understanding its interaction with TCP. Specifically, we study the dynamics of TCP traffic in an integrated services network that simultaneously supports both best-effort and controlled-load sessions. In light of this study, we propose and experiment with modifications to TCP's congestion control mechanisms in order to improve its performance in networks where a minimum transmission rate is guaranteed. We then investigate the effect of network transients, such as changes in traffic load and in service levels, on the performance of controlled-load as well as best-effort connections. To capture the evolution of integrated services in the Internet, we also consider situations where only a selective set of routers are capable of providing service differentiation between best-effort and controlled-load traffic. Finally, we show how the service mechanisms proposed here can be embedded within other packet and link scheduling frameworks in a fully-evolved integrated services Internet.

Wu-chang Feng and Kang G. Shin were supported in part by the Office of Naval Research under Grant N00014-94-1-0229.

1 Introduction

A large class of Internet applications, referred to as tolerant playback applications in [1, 3], can greatly benefit from an increased level of predictability in network services. These applications typically buffer a portion of the data on the client before starting the playback, and then operate in a streaming mode to prevent buffer underflow. Examples include increasingly popular applications such as PointCast, RealAudio, and VDOnet which stream text/image, audio, and video data over the Internet. The quality of playback for each of these applications can vary from excellent to intolerable, depending on the network load. These applications can benefit tremendously from a network service that guarantees a minimum level of throughput at all times, but allows for the possibility of higher throughput during periods of light loads. Such a service is also useful to more traditional, elastic applications [1, 3], such as `ftp` and `telnet`. Tasks such as booting a disk-less workstation over the network, backing up remote files, and synchronizing web proxies can be performed with more predictability and within a bounded time by guaranteeing a minimal bandwidth to the underlying sessions. This service can also be used to set up a virtual overlay network in the Internet, connecting business-critical servers and clients with virtual links of a minimum guaranteed bandwidth.

The controlled-load service [21] currently being standardized by the Internet Engineering Task Force (IETF) fits very well into the scenarios sketched above. It is part of an ambitious goal of defining a service architecture that is suitable for a diversity of applications. Our objective in this paper has been to follow an evolutionary path towards this goal. That is, to enhance network services within the framework of the IETF-defined service architecture, but with minimal enhancements to the network infrastructure, especially the routers.

We propose one possible implementation of controlled-load service using a simple extension to the queueing mechanisms in today's routers coupled with modifications to TCP's congestion control mechanisms. These modifications allow the network to guarantee a minimal level of end-to-end throughput to different network sessions. In addition, any residual network capacity is shared in a socially cooperative fashion, in a manner similar to the one in use in the Internet today by applications using TCP. In this scheme, each reserved session is associated with a traffic envelope. Traffic is policed at the source and packets conforming to the envelope are marked. Non-conformant traffic and best-effort traffic is injected into the network unmarked. At the routers we use an enhanced random early detection (RED) [7] and discard mechanism. Both marked and unmarked packets share the same FIFO queue. When the queue length at the router exceeds a certain threshold, packets are dropped randomly as done in RED gateways. However, unlike standard RED gateways where all packets have the same drop probability, in the enhanced RED (ERED) gateway, marked packets have a lower drop probability than the unmarked packets.

The service realized by the mechanism described above is an interpretation of the controlled-load service. By definition, traffic belonging to a controlled-load session and conforming to the associated traffic envelope sees very little loss and very little queueing delay through the network. Non-conformant controlled-load traffic is treated as best-effort traffic. By using a common queue for best-effort and conformant controlled-load traffic, we essentially relax the recommended delay targets for conformant controlled-load traffic. This laxity not only simplifies the implementation and reduces packet handling overheads at the routers, but also helps maintain packet ordering. Note that ERED ensures a low loss rate to conformant controlled-load traffic. We argue that elastic and tolerant playback applications can withstand a reasonable amount of queueing delay and can fully exploit the guarantees on throughput to improve performance.

We note that the controlled-load service or an equivalent service providing minimum rate guarantees can be realized using various alternative queueing mechanisms. Similarly, applications using transport protocols other than TCP may make use of such a service. In this paper, we focus on ERED queueing primarily because 1) it does not require per-flow queueing which may not scale well and 2) it maintains FIFO ordering of packets which is important to TCP's congestion control mechanisms. We examine TCP because: (1) an overwhelming number of applications use TCP as the transport protocol of choice, and (2) TCP has a well-developed congestion and flow control mechanism that makes it an interesting case study. While some of the tolerant playback applications may not use TCP, the mechanisms described here can easily be applied to other transport protocols, such as RTP.

Our objective in this paper is to understand and to modify end-to-end control mechanisms used in TCP in an integrated services Internet that supports both best-effort and controlled-load services. While the analysis of the behavior of unmodified TCP over a network which supports priority marking and handling has particular relevance to DiffServ proposals, we note that the proposed TCP modifications are made assuming that the network supports minimum rate guarantees and should only be deployed in this environment in order to prevent possible congestion collapse.

Integrated services in the Internet is a relatively new area of research. To the best of our knowledge, no published work addresses the specific issues discussed in this paper. In a more general sense, studies on supporting TCP over ABR/UBR services in ATM networks [11, 17] address similar issues. However, due to significant differences between the service architectures of ATM and the Internet, the nature and the focus of these studies are quite different. In [11], the authors propose using a modified switch buffer allocation policy in order to obtain peak throughput and fairness among TCP connections over ATM. In particular, each connection is effectively provided with a weighted fair share of the buffers in the switch. The switch uses per-connection accounting to determine which connections are overloading their allocation and drops cells accordingly. In contrast, most of the modifications we propose are with TCP's congestion control algorithm. This is in line with the Internet design philosophy of providing sophisticated end-to-end control in end hosts, coupled with a relatively simple control inside the network. Moreover, since these modifications are required only for senders, they may be deployed incrementally.

The rest of the paper is organized as follows. In Section 2 we briefly describe the proposed service architecture and the ERED mechanism. Performance of TCP in an integrated services environment and the effects of different service parameters are investigated in Section 3. In Section 4, we propose simple modifications to TCP's transmission and windowing mechanisms to exploit reservations in the network. Section 5 examines the overhead such modifications incur and the benefits which they provide. In Section 6, we study the impact of network transients, such as changes in service levels and changes in traffic loads on the end-to-end performance of controlled-load and best-effort sessions. Section 7 considers network scenarios in which a subset of the routers are ERED-capable in order to capture the path of evolution of

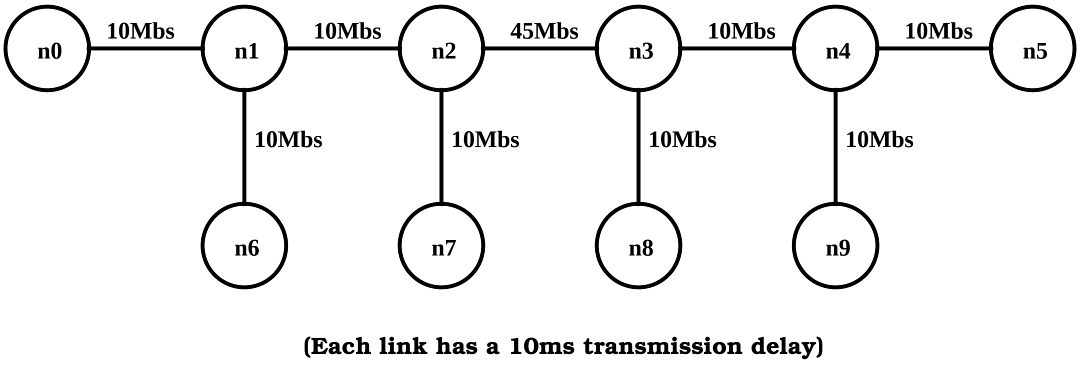


Figure 1: Network topology

integrated services in the Internet. Finally, Section 8 presents a discussion and experimentalresults on the possible integration of the ERED mechanism into a more elaborate packet and link scheduling architecture, such as class-based queueing. We conclude in Section 9.

2 Network and Service Models

The RSVP and the INTSERV working groups in the IETF are responsible for defining protocols and standards to support integrated services in the Internet. In this section we briefly review the relevant aspects of these standards and show how the proposed enhancements fit into the IETF-defined framework.

2.1 Policing and Marking

To avail itself of a reservation, a connection has to specify a traffic envelope, called *Tspec*. The Tspec includes a long-term average rate r_m, a short-term peak rate r_p, and the maximum size of a burst b of data generated by the application. For example, for an application generating MPEG-encoded video, the average rate could be the long-term data rate, the peak rate could be the link bandwidth at the source, and the burst size could be the maximum size of a frame. The Tspec also specifies the maximum and minimum packet sizes to be used by the application. Connections are monitored and policed at the network entry points. This could be either at the source, or at the boundary between the corporate or campus intranet and the Internet. Packet classification and service differentiation also takes place at the routers. The service priority given to a packet is a function of the Tspec, and in the case of some service classes, a separate service specification known as *Rspec*. For controlled-load service, no Rspec is specified.

In order to police traffic at the source, we use token buckets [18]. The token generation process follows the Tspec advertised by the source. That is, the long-term average rate of token generation is t_m, the short-term peak rate of token generation is t_p, and the depth of the token bucket is b. Each time a packet is injected into the network, if sufficient tokens are available, an equivalent number of tokens are considered consumed. If there aren't enough tokens present at the time of transmission, the packet is treated as non-conformant.

In addition to policing, we also propose to mark packets at the network entry point. Conformant controlled-load traffic is marked before being injected into the network. Non-conformant controlled-load traffic and best-effort traffic is injected into the network unmarked. Although marking is not currently supported in IP networks, there are sufficient hooks (specifically the type of service bits) in the IP header to add this feature easily. Note that marking is not a mandatory requirement, it just facilitates traffic classification at the routers. In the absence of a marking facility, IP datagrams have to be passed through a classifier at the source, as well as at the routers, to determine which flows they belong to and to determine whether they are in violation of, or in conformance with, the advertised Tspecs of the flows. In the presence of a marking facility, classification is only required at the network entry point and not at interior routers. In the rest of the paper, we assume that a marking facility is available.

2.2 Packet Handling

The routers perform admission control for controlled-load connections. Admission control algorithms are not discussed in this paper, but, for the purpose of the experiments, we assume that the aggregate reservation levels at the routers are within their capacities.

In addition to performing admission control, the routers also need to support service differentiation between marked (conformant controlled-load) and unmarked (non-conformant controlled-load and best-effort) packets. One obvious approach to providing different services to marked and unmarked packets is to maintain separate queues for each class and serving them according to their scheduling priority. However, we propose to use a common queue for both compliant and non-compliant traffic and serve them in FIFO order. A common FIFO queue not only simplifies the scheduling functionality at the router, it also helps maintain packet ordering in controlled-load connections. Although maintaining packet ordering is not a requirement, failure to do so may have serious performance impacts on transport protocols such as TCP.

Our approach to service differentiation between marked and unmarked packets relies on a selective packet discard mechanism. We use an enhanced version of the RED (Random Early Detection) algorithm for this purpose. In classical RED routers, a single FIFO queue is maintained for all packets. Packets are dropped randomly with a given probability when the average queue length exceeds a certain minimum threshold (min_{th}). The drop probability itself depends on the average queue length and the time elapsed since the last packet was dropped. When the average queue length exceeds a maximum threshold (max_{th}), all arriving packets are dropped.

Enhanced Random Early Detection (ERED) is a minor modification to the original RED algorithm. In ERED, the thresholds only apply to unmarked packets. Unmarked packets are randomly dropped when the average queue length exceeds $min_{th}(unmark)$ and are all dropped when the average queue length exceeds $max_{th}(unmark)$. Marked packets are only dropped when the queue is full. In order to ensure low loss of marked packets, min_{th} and max_{th} values should be set appropriately. For example, in a system with n controlled-load sessions with peak rates of $r_p^i, i = 1, 2, \ldots, n$, a service rate of L, and a buffer of length

B, the thresholds must be set so that they can roughly ensure that no marked packets are dropped [1]. In particular, the following equation should hold:

$$\left(\sum_{i=1}^{n} r_p^i - L\right) \times \frac{max_{th}(unmark)}{L} < B - max_{th}(unmark)$$

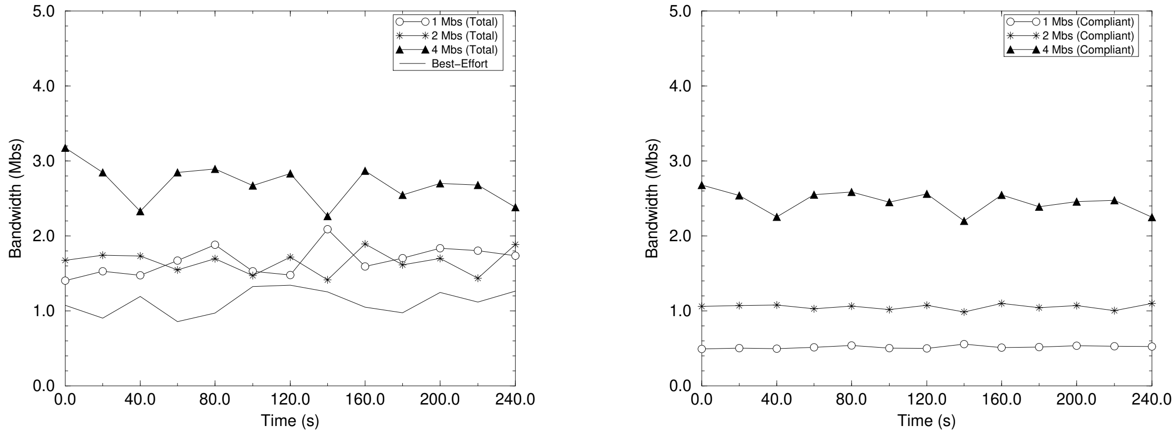
Note that the maximum number of unmarked packets that can be in the queue at any time is around (max_{th}(unmark)). It takes at most $max_{th}(unmark)/L$ to completely drain the queue of unmarked packets. Given the maximum aggregate arrival rate of marked packets $\sum r_p^i$ and the service rate L, the rate of increase of the queue occupancy $\sum r^i p - L$ [2]. Hence, the amount of excess buffer space needed to ensure no marked packets are dropped is the product of $(\sum r_p^i - L)$, the rate of increase in queue occupancy and $max_{th}(unmark)/L$, the time needed to drain the unmarked packets. Unfortunately, since the thresholds such as $maxth$ are triggered by an average queue length calculation, these settings can still lead to unnecessary losses in compliant packets. For example, additional queueing work [2] has shown the use of absolute thresholds for limiting unmarked packets can effectively prevent loss of marked packets. While ERED works well for the experiments in this study, a queue management algorithm which reserves a portion of the queue for marked packets while performing RED on the remaining portion might be ideal for controlling congestion while effectively supporting the priority marking.

An appropriately parameterized ERED queue can guarantee low loss rate to conformant controlled-load traffic. However, since it uses a common FIFO queue, the delay experienced by the conformant controlled-load traffic and best-effort traffic are the same. It is possible to parameterize ERED queues to control the queue size, and hence, the queueing delay. However, a small queue size may lead to high loss rates for unmarked packets. An alternative approach is to maintain separate queues for controlled-load and best-effort traffic. Separation of traffic classes is likely to improve the delay performance of controlled-load traffic. However, it complicates bandwidth sharing between non-conformant controlled-load and best-effort traffic. The router would have to use weighted fair queuing to ensure equal-fair share of excess bandwidth to non-conformant controlled-load and best-effort traffic. Consequently, it has to monitor the number of active connections in the controlled-load and best-effort classes and has to adjust the weights dynamically depending on the number of connections in each class.

Finally, there are many other ways of realizing controlled-load service. Some of these mechanisms, such as class-based queueing [8] and weighted fair queueing [1, 3, 5, 9, 10, 19, 20], can be used to accurately implement controlled-load and other service classes defined by the INTSERV working group. However, as mentioned earlier, the primary focus of this paper is not to propose and analyze packet queueing and scheduling mechanisms to realize integrated services on the Internet. Our objective is to study how we can offer a particularly useful network service with minimal changes to the routers and the end-hosts. We show, that the proposed simple enhancements can benefit a large class of elastic and tolerant playback applications that can withstand a reasonable amount of queueing delay and can really exploit the guarantees on throughput to improve performance. Although ERED by itself may not be used to realize other classes

[1] We assume that the duration of bursts is the same for all sources. This assumption can be relaxed for more precise admission control.

[2] Transmission of an unmarked packet makes room for an incoming marked packet.



(a) Aggregate throughput (b) Compliant throughput.
$(min_{th} = 20K, max_{th} = 80K, Q_{size} = 100K, BucketDepth = 50ms)$

Figure 2: Effect of reservation on end-to-end throughput.

of service, such as guaranteed delay, it can be easily integrated into more general service architectures. In Section 8, we discuss how it can be embedded in a class-based-queueing framework.

3 Understanding TCP Dynamics

This section is devoted to the study of TCP dynamics in an integrated services environment. For the purpose of the experiments, we modified the NS [16] simulator [3]. The NS simulator has been used extensively in a number of studies reported in the literature. While the simulator does not use production TCP code, it implements congestion and error control algorithms used in different implementations of TCP with remarkable accuracy. For most of the experiments reported here, we use a Reno-variant of TCP [12]. We modified the simulator by adding policing and extending the RED queueing discipline.

For the experiments in this section, we consider a simple network topology shown in Figure 1. The capacity of each bi-directional link is labeled and has a transmission delay of `10ms`. Connections requesting a reservation specify a peak and a mean rate of service, and the maximum size of a burst. At the source, tokens are generated at the service rate and are accumulated in a token bucket. The depth of the token bucket is the same as the maximum burst size specified by the source. Throughout this paper, the token bucket size is measured in units of time. In units of tokens, it is equivalent to token generation rate times the bucket size in units of time. The peak rate is set to the link speed by default. TCP segments belonging to reserved connections are transmitted as marked datagrams if there are sufficient tokens available in the token bucket at the time of transmission. Otherwise, they are sent as unmarked datagrams. TCP segments belonging to best-effort connections are sent as unmarked datagrams. We assume that sources are greedy,

[3] Version 1.1

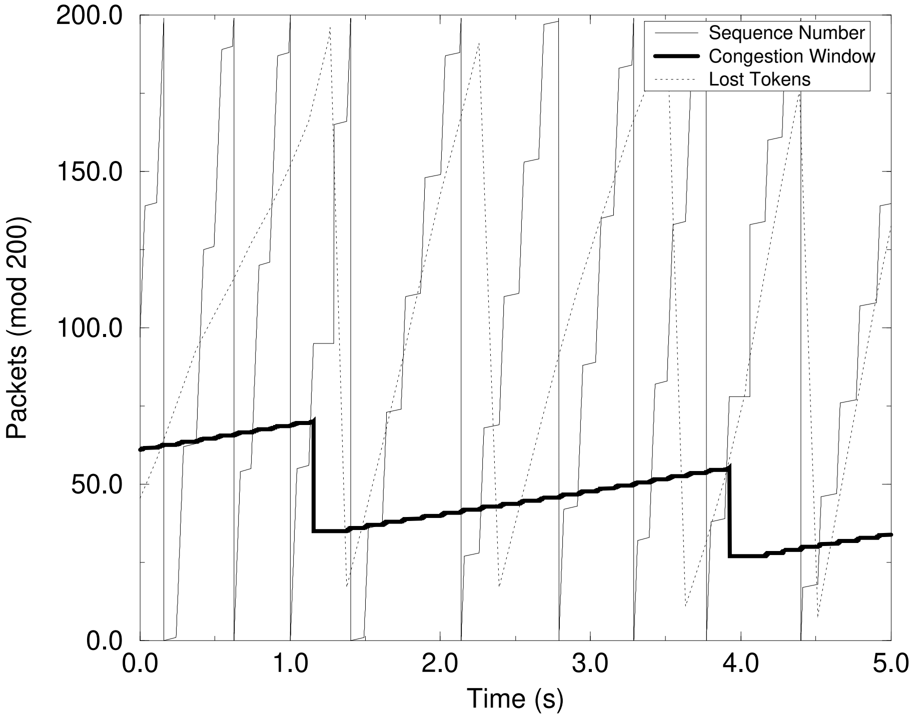


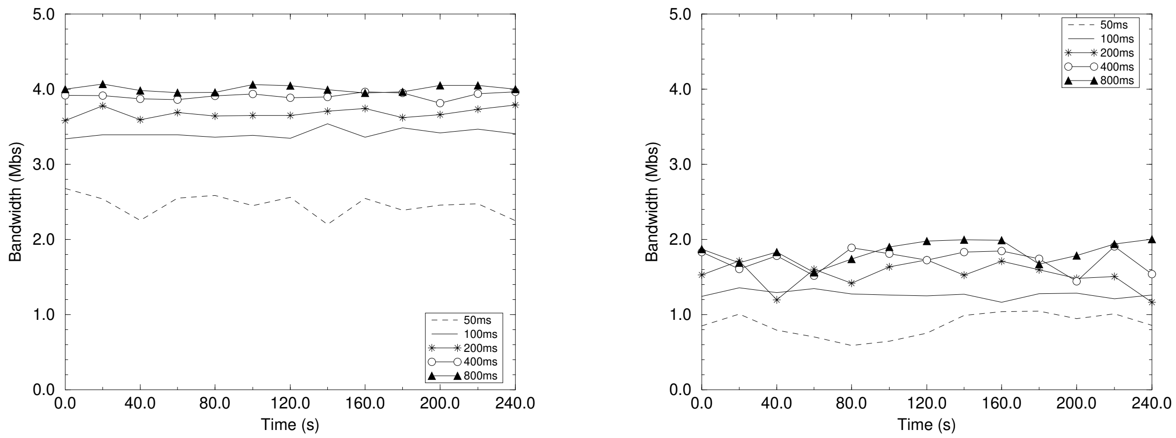
Figure 3: Packet trace of the connection with `4Mbs` reservation.

that is, they always have data to send.

3.1 Effect of Service Rate

This experiment is designed to investigate the effect of service rate on end-to-end throughput. For the purpose of this study we ran three connections with reservations of `1Mbs`, `2Mbs`, and `4Mbs`, and three best-effort connections from node `n0` to `n5`. Each controlled-load source used a token bucket of depth `50ms`. and each node has a `100KB` ERED queue with max_{th} of `80KB` and min_{th} of `20KB`. The maximum drop probability of the unmarked packets for this experiment was `0.02`. This probability is chosen in order to make early detection aggressive enough to control the length of the queue. Note that a drop probability which is too small makes the early detection mechanism ineffective while a drop probability which is too large can lead to underutilization of the link.

Figure 2(a) shows the throughput seen by each connection. Throughput is computed by measuring the data received at the receiver over an observation period and dividing it by the observation interval. Figure 2(b) plots the compliant throughput seen by connections with reservations. This is the portion of the throughput that is contributed by marked packets. Ideally, it should be equal to the reserved rate of service. From Figure 2(a), we observe that connections with higher reservations generally see better throughput than connections with lower or no reservations. However, from Figure 2(b) we observe that the compliant portions of the bandwidth received by all reserved connections are less than their respective



(a) One-way traffic. (b) Two-way traffic.

$(min_{th} = 20K, max_{th} = 80K, Q_{size} = 100K)$

Figure 4: Compliant throughput of the `4Mbs` connection over varying token bucket sizes.

service rates.

The explanation for the observations from Figure 2 lies in the flow and congestion control mechanisms used by TCP. The TCP sessions with reservations exercise their flow and congestion control mechanisms in the same way as best-effort connections. However, they have a lower probability of losing a packet at the routers since their marked packets have lower (in this case close to zero) probability of getting dropped. Because connections with higher reservations mark their packets at a higher rate, they have a decreased probability of having a packet dropped. This is why connections with higher reservations see higher throughput than connections with lower or no reservations. However, as observed from Figure 2(b), TCP fails to fully exploit the benefits of the reservation. The compliant part of the throughput is less than the reservation levels in all cases. Since marked packets are not dropped by the network it is apparent that the source is not generating sufficient number of marked packets to keep the reserved pipe full. Since the sender is a greedy source, it is the TCP congestion control mechanism that is responsible for throttling the source. The tokens, however, are generated at a rate commensurate with the reservation. If the source does not have enough or is unable to transmit packets, the token bucket fills up and ultimately overflows. We refer to this phenomenon as token loss. The rate of token loss is a good measure of loss of transmission credits wasted due to undesirable side-effect of TCP congestion control.

Figure 3 shows the packet trace of the connection with `4Mbs` reservation over a five-second interval. The plot shows the sequence number [4] (modulo 200), the congestion window of the sender, and the number of lost tokens (given in packets modulo 200) for the connection. A positive slope of the lost token curve indicates a non-zero token loss rate throughout the observation period. The windowing mechanism used by TCP is partly responsible for this phenomenon.

[4] We use segments of size `1KB`. The sequence number is the sender's packet sequence number.

```
(1) After every acknowledgment
        if (room under congestion and advertised windows)
                if (tokens available > packet size)
                        send packet as marked
                else
                        send packet as unmarked

(2) After every TIMER expiry
        if (room under advertised window)
                if (tokens available > packet size)
                        send packet as marked
        reset TIMER
```

Figure 5: Timed transmission algorithm.

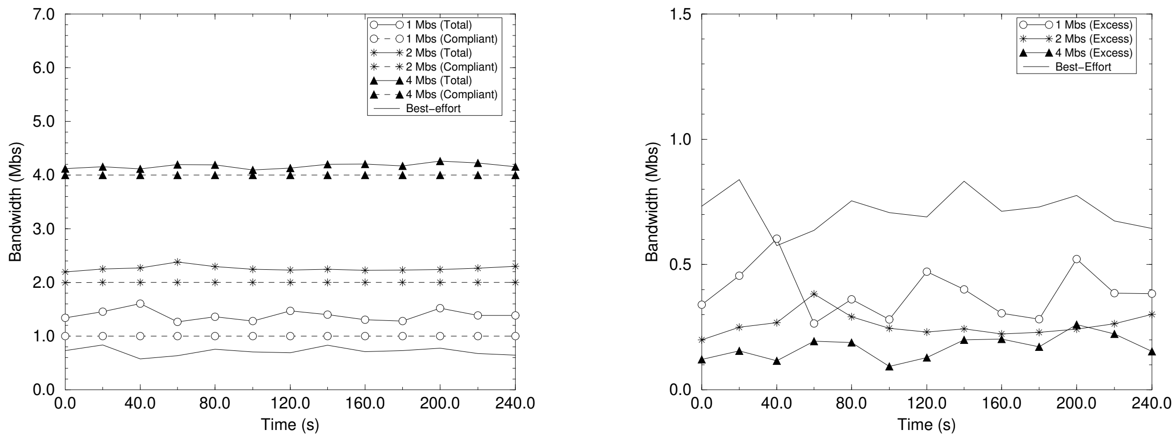
TCP uses two windows for the purpose of flow and congestion control. The receiver maintains and enforces an advertised window (AWND) as a measure of its buffering capacity. The sender enforces a congestion window (CWND) as a measure of the capacity of the network. The sender is prohibited from sending more than the minimum of AWND and CWND worth of unacknowledged data. When the loss of a segment is detected, TCP reduces the congestion window and initiates a *fast recovery* or a *slow start* phase. For the fast recovery phase, the congestion window is cut to half of its original size while in the slow start phase it is set to 1. For connections with reservations, this is an overly conservative behavior since it is insensitive to the reservation that a particular connection may have. Thus, even when tokens are present, and the sender is eligible to transmit a new segment, it may be throttled by the congestion window. As shown in Figure 3, the rate of token loss increases (as indicated by the change in slope in the lost token curve) significantly when a packet loss is detected (as indicated by the decrease in congestion window), and slowly decreases as the congestion window opens up.

Another cause for token loss is the presence of persistent gaps in the acknowledgment stream. Such gaps are part of a phenomenon commonly referred to as ack-compression [22]. Since TCP uses acknowledgments to trigger transmissions, any significant time gap between the receipt of successive acknowledgments causes the token bucket to overflow and results in a loss of transmission credits. The effects of these gaps can be seen in many places in the trace where the sequence number is frozen. There are several ways in which these gaps can develop. One is through the recovery process after a loss is detected using TCP's fast recovery and fast retransmit mechanisms. After detecting a loss (by the receipt of a given number of duplicate acknowledgments), TCP cuts its congestion window in half by halting additional transmissions until one half of the original window's packets have cleared the network. Freezing the sender for this period of time causes the token bucket to overflow, but more importantly, puts a gap in the data stream which results in a gap in the acknowledgment stream during the next round-trip interval. Gaps in the acknowledgments cause the token bucket to overflow and cause gaps in the data stream once again. Another way gaps can form is through the normal dynamics of network traffic. Congestion on the forward and/or reverse path, as well as additional queueing delays and jitter experienced as new connections come on-line, can also create significant gaps in the stream.

3.2 Effect of Token Bucket Depth

One way to alleviate the problem of token loss is to use a deeper token bucket. To investigate the impact of the token bucket depth on compliant throughput, we repeated the experiment described in the last section across a range of token bucket depths. Figure 4(a) shows the compliant throughput seen by the connection with a `4Mbs` reservation for token bucket sizes of `50ms`, `100ms`, `200ms`, `400ms`, and `800ms` using the same network topology and traffic. Increasing the token bucket depth improves the compliant throughput seen by a connection. However, it is only when the token buckets are very large (`400ms` and `800ms` in this case) that the compliant throughput seen by a connection remains at the reserved rate. Unfortunately, for a `4Mbs` connection, this bucket depth corresponds to a maximum burst of compliant packets of up to `200KB`. In order for the network to ensure that compliant packets are not dropped, it must have the capacity to buffer such bursts. Without sufficient buffer space, a significant amount of burst losses can occur, causing the performance of the TCP connection to deteriorate. To see the effect of this, we increased the traffic going through the network by adding identical traffic going in the reverse direction. That is, all connections are bidirectional. Adding traffic has several effects. One effect is that it spaces out acknowledgments even further. Another, more problematic effect, is that it adds more congestion to the network which causes queues to be fully occupied. Without sufficient buffer space to handle large token buckets worth of priority packets, reserved connections experience a substantial amount of burst losses. This causes the connection to eventually freeze since recovering from multiple losses using Reno TCP in particular requires multiple round-trip times. Because of this the source eventually runs out of room under the advertised window of the receiver and must stop sending until the lost packets are successfully retransmitted [6]. Note that as the bandwidth-delay is increased, the advertised window becomes a limiting factor in the performance of the source. If the advertised window is not large enough, whenever a single packet is lost, the source must freeze until the packet is successfully retransmitted. Figure 4(b) shows the result of this experiment. In contrast to Figure 4(a), large token buckets do not give any additional performance improvement. The connection never receives a compliant throughput more than half of its `4Mbs` reservation.

The use of large token buckets allows large bursts of marked packets into the network which can result in loss of marked packets, thus defeating the service differentiation mechanism provided by ERED. In a WFQ implementation of controlled-load service, this is akin to overflowing a flow's queue by allowing it to burst at a rate greater than the queue length. As with any service differentiation, in order to ensure correct behavior, admission control must be done with the sources to guarantee performance. For ERED queues, this essentially means that the min_{th} and max_{th} values must be set appropriately so that marked packets are not dropped due to an over-occupation of unmarked packets. For a given queue size there is some flexibility in terms of setting the thresholds depending on the traffic load. As the load due to controlled-load traffic increases, the max_{th} and min_{th} values can be lowered to ensure low or no loss of conformant controlled-load traffic. As the controlled-load traffic load decreases, the max_{th} and min_{th} values can be



(a) Total and compliant throughput. (b) Share of excess bandwidth.

$(min_{th} = 20K, max_{th} = 80K, Q_{size} = 100K, BucketDepth = 50ms, TimerInterval = 20ms)$

Figure 6: Throughput with timer-triggered transmissions.

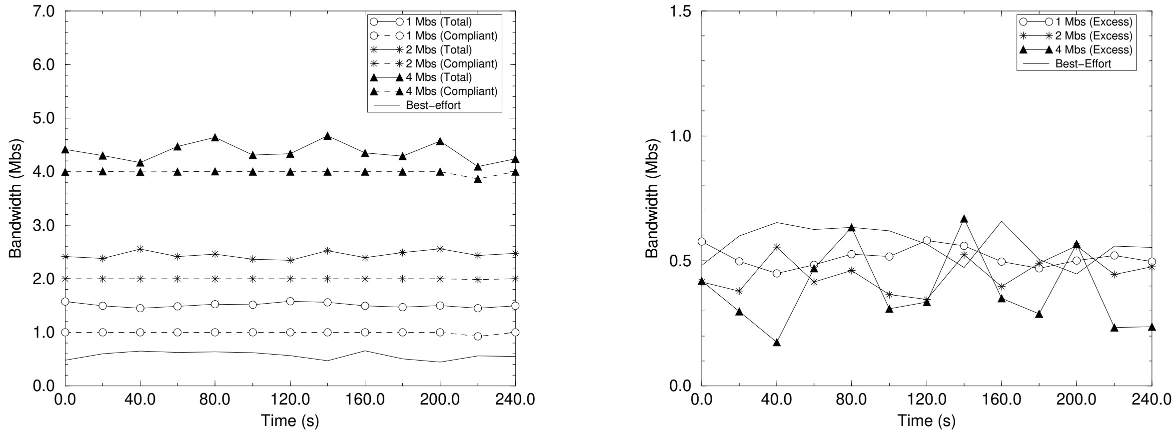
set higher to improve the throughput of best-effort traffic. However, the extent to which this flexibility in setting the queue thresholds can be used to ensure correct behavior is limited. As discussed in Section 2, for a given queue size and link speed, the aggregate controlled load traffic that can be admitted into the system is limited by

$$\left(\sum_{i=1}^{n} r_p^i - L\right) \times \frac{max_{th}(unmark)}{L} \leq B - max_{th}(unmark).$$

The above condition guarantees zero loss of conformant controlled-load traffic in the worst case scenario where all controlled-load sources burst traffic at the highest rate at the same time. In the rest of the discussion in this paper, we assume that the above condition is satisfied at every node and marked packets are never dropped. However, we note that the scenario captured in the above inequality is very pessimistic. It is extremely unlikely for all controlled-load sources to burst at the same time. Also, guaranteeing zero loss of conformant traffic is not a requirement for controlled-load service. Consequently, it is possible to relax this condition and admit more controlled-load sessions than deemed possible by the above inequality. We believe that empirical and statistical admission control mechanisms, such as measurement based admission control [15], can be used to operate the network at a high utilization and maintaining a low loss rates of conformant controlled-load traffic.

4 TCP Adaptations

In this section we propose and experiment with modifications to TCP's control mechanisms. These refinements can help TCP adapt better in an integrated services environment.



(a) Aggregate and compliant throughput. (b) Share of Excess Bandwidth.
$(min_{th} = 20K, max_{th} = 80K, Q_{size} = 100K, BucketDepth = 50ms, TimerInterval = 20ms)$

Figure 7: Throughput with timer-triggered transmissions and windowing modifications.

4.1 Timed Transmissions

Since deeper token buckets require larger buffers in routers and allow less flows to be admitted, it is desirable to keep the size of the token buckets small. To alleviate the effects of persistent gaps in acknowledgment without increasing the token bucket depth significantly, we experimented with two different schemes, *delayed* and *timed* transmissions. These mechanisms better adapt the acknowledgment-based transmit triggers to the rate-based reservation paradigm. In the delayed transmission mechanism a segment is held back for a random amount of time when there aren't enough tokens to transmit it as a marked packet. This, in effect, adds randomization to the data stream of the connection which can potentially eliminate persistent gaps in the acknowledgment stream. In addition, this scheme reduces the probability of the packets getting dropped inside the network since holding back packets increases the probability that they are sent as marked packets. While the delayed transmissions work reasonably well when the reverse path is lightly-loaded [6], additional experiments have shown that it is not very effective in the presence of reverse path congestion.

The second mechanism we examine involves the use of a periodic timer. In this scheme, we augment TCP's acknowledgment-triggered transmissions with a timer-triggered transmission mechanism. This timer-based triggering ensures that transmission opportunities are not lost while the connection is waiting for an acknowledgment. In the timed transmission mechanism, each reserved connection uses at most one timer which can have an interval which is customized. Connections can also share a single timer depending on the overhead on the end-host. In the timed transmission scheme, the acknowledgment-clocked transmission algorithm is left unmodified. However, whenever a periodic timer expires, the connection examines the tokens in the token bucket. If there are sufficient tokens available in the token bucket and there is room under the advertised window of the receiver, the sender sends the packet as marked, ignoring the value of the congestion window. The timer is then reset to wake up another timer interval later. Figure 5 presents the algorithm formally.

The intuition behind timed transmission is very simple. If there are enough tokens in the bucket, as per contract with the network, the sender is eligible to inject new data in the network. Hence, we temporarily disregard the congestion window under such circumstances. Note that it is also possible to disregard the congestion window for conformant sends triggered with an acknowledgment. However, the use of the timers helps prevent sending back-to-back packets, making the resulting traffic stream slightly smoother and more network-friendly. Regardless of how compliant sends are triggered, the connection still adheres to the advertised window constraint to avoid overflowing the receiver's buffers. In case of network disruption, the sending TCP freezes when the number of unacknowledged packets reaches the advertised window. Thus, the time-triggered sends do not continue to occur in the presence of network failure. Having a timer trigger transmissions alleviates the problem of lost tokens caused by gaps in the acknowledgments. In order to guarantee zero token loss, the timer interval should be equal to [TokenBucketDepth - (PacketSize - 1)]/TokenBucketRate. This takes care of the worst case where there are (PacketSize - 1) tokens in the bucket when a timer interrupt occurs.

Using this timer mechanism, we reran the same experiment described earlier. For the experiment, we used token buckets of depth `50ms`, and a timer granularity of `20ms`. Figure 6(a) plots the total bandwidth received by all connections and the compliant bandwidth received by the connections with reservations. As shown in the figure, each connection gets its reserved rate and a share of the excess bandwidth.

While the timed transmissions allow for temporary violations of the congestion window to occur, non-compliant packets are sent only when there is room under the congestion window. Thus, this mechanism does not alter the way TCP's congestion window is calculated. Using TCP's windowing algorithm can be a problem since upon detection of a loss, the congestion window is cut in half or reduced to 1 regardless of a connection's reservation. Thus, although the timed transmission mechanism allows the connection to receive its reserved rate, TCP's windowing mechanism can restrict the controlled-load connections from competing for the excess bandwidth [5] in the network. Figure 6(b) plots the throughput seen by a best-effort connection and the non-compliant throughput seen by each of the reserved connections using timed transmissions. The plots show that connections with reservations receive a smaller share of the residual capacity when compared to the best-effort connection. The connections with larger reservations are penalized to a greater extent since halving the congestion window of a connection with `4Mbs` reservation has a more drastic impact than halving the congestion window of a `1Mbs` connection.

4.2 Rate Adaptive Windowing

For reserved connections, the swings in the congestion window should always be above the window guaranteed by the reserved rate. To account for and exploit the reservation, we modified TCP's windowing algorithm. The key idea behind this modification is that for reserved connections, CWND consists of two

[5] Non-conformant controlled-load traffic is treated as best-effort traffic. Hence, residual network capacity should be fairly shared between best-effort traffic and non-conformant controlled-load traffic.

(1) After every new acknowledgment

if (CWND $<$ SSTHRESH)

CWND $\leftarrow$ CWND+(CWND-RWND)/CWND

else

CWND $\leftarrow$ CWND+1/CWND

(2) When NDUP exceeds a threshold

CWND $\leftarrow$ RWND+(CWND-RWND)/2+NDUP

SSTHRESH $\leftarrow$ RWND+(CWND-RWND)/2

(3) Upon RTO

CWND $\leftarrow$ RWND+1

SSTHRESH $\leftarrow$ RWND+(CWND-RWND)/2

Figure 8: Rate adaptive windowing algorithm.

parts: a reserved part equal to the product of the reserved rate and the estimated round-trip time, and a variable part that tries to estimate the residual capacity and share it with other active connections. Note that the reserved part of CWND is a function of the round-trip time. While we currently use the common TCP round-trip measurements to estimate this, measurements using the proposed TCP timestamps option (RTTM) [14] can provide a more accurate estimate.

Let us assume that the size of the reservation window is RWND. Hence, the size of the variable window is CWND-RWND. In the modified scheme, we adjust the size of the variable window using the traditional TCP windowing mechanism and simply add it to the calculated value of RWND. Specifically, the sender, instead of reducing CWND by half at the beginning of the fast recovery, sets it to RWND + (CWND − RWND)/2. At the beginning of a slow start after detection of a lost segment through the retransmission timeout, it sets CWND to RWND+1 instead of 1. In both cases, SSTHRESH is set to the minimum of RWND+(CWND − RWND)/2 and AWND instead of the minimum of CWND/2 and AWND. Finally, because packets sent under RWND should not clock congestion window increases, we scale all window increases by (CWND − RWND)/CWND. Note that even with these modifications to the windowing algorithm, the sender must still adhere to the AWND restriction. That is, it is prohibited from sending more than the minimum of AWND and CWND worth of unacknowledged data. Because of this, the size of the receiver's buffer must be at least the size of the reservation window in order to sustain the reserved rate using TCP. This control algorithm has been summarized in Figure 8.

We repeated the experiments described in the last section with the windowing modifications in place. Figure 7(a) shows the aggregate and compliant throughput seen by each reserved connection after modifications to the windowing mechanisms. It also shows throughput seen by a best-effort connection between the same source and destination. As seen in the figure, all connections perform as expected. Figure 7(b) plots the amount of excess bandwidth received by each reserved connection, as well as the bandwidth received by the best-effort connection. When compared to Figure 6(b), the reserved connections obtain a fairer share of the excess bandwidth.

A common concern with any modification to TCP's windowing mechanism is that the change may be too aggressive and thus, cause unnecessary congestion. The experiments we have conducted so far, including the ones reported in this paper, show no bias towards connections using the modified windowing mechanism. We experimented with different flavors of the windowing algorithm. They differ in the way RWND is computed and CWND is clocked. We compute RWND by multiplying the reserved rate with the estimated round-trip time. Depending on how conservative we want the windowing mechanism to be, we can use different estimates of round-trip time. We experimented with both best and average estimates of round-trip times. In all the experiments we have conducted, they perform equally well. However, in times of congestion, the estimated round-trip time tends to be large and thus, the rate-based window can also grow large during a period of time when the network needs a respite. Using the best observed round-trip time in this case, allows the connection to be on the conservative side in calculating its rate-based window.

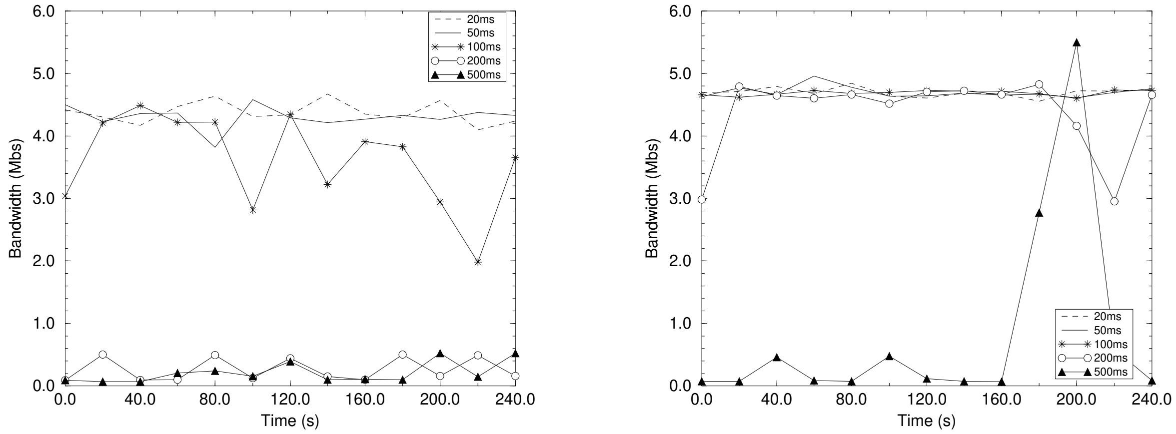
Another concern in deploying the modifications is that it may potentially lead to congestion collapse since the modified TCP sources maintain minimum transmission rates at all times. Although we assume that the network supports a minimum rate of service, the presence of legacy equipment may make such a promise difficult to honor. If the lack of network support for minimum rate guarantees go unnoticed by the source, the proposed modifications may lead to congestion collapse. Fortunately, preventive measures can be taken to avoid such mishaps. For example, the TCP source can and should turn off both the timer and window modifications whenever it detects any loss of marked (compliant) packets.

5 Fine-Grained Timers

This section explores the cost associated with deploying fine-grained timers into TCP as well as the benefits of using such a timer for sending data.

5.1 Timer Overheads

In our description of the timed transmission algorithm, we have assumed the existence of connection-specific timers. However, it is possible, and desirable, to use a common timer shared amongst all reserved connections. Such optimizations can be easily incorporated using techniques such as the protocol timer services in the BSD-style TCP/IP stack. One of the common criticisms against the use of timers is the overhead associated with handling timer interrupts. For that reason, TCP uses coarse-grained (typically 200ms and 500ms) timers. However, the state of the art in processor technology and operating systems has advanced considerably since the first design and implementation of TCP. Processors and timer implementations today are much faster, and consequently, the overheads of handling timer interrupts are much



(a) 80KB Buffers. (b) 160KB Buffers.

$(min_{th} = 20K, max_{th} = 80K, Q_{size} = 100K, BucketDepth = TimerInterval + 30ms)$

Figure 9: Aggregate throughput of 4Mbs connection over various timer interrupt granularities.

CPU Type	133MHz PowerPC	33MHz POWER
Timer setting	7.4μs	14.0μs
Timer handling	7.1μs	30.1μs
Timer canceling	6.5μs	9.6μs

Table 1: Timer overheads (AIX 4.2 kernel).

lower [4].

Table 1 shows the overheads of setting, canceling, and handling timers in two IBM RS/6000 machines running AIX 4.2, one equipped with a 33MHz POWER CPU and the other with a 133 MHz PowerPC CPU. We observe that the overheads of timer operations in modern systems (133 MHz PowerPC) are quite small. Even when older systems, such as the 33MHz RS/6000, are considered in this study, the overheads are well within acceptable limits. Note that these measurements were taken without any optimization to the timer data structures in the AIX kernel. In AIX 4.2 timer blocks are arranged in a linear array. The overhead of timer operations are expected to be even lower if the timer blocks are stored as a hash table. However, at this point such an optimization is not deemed necessary.

5.2 Buffer Requirements

While there are concrete costs associated with using fine-grained timers, there are also significant benefits. One benefit in using these timers is that it reduces the size of the token buckets used for each application. From the calculations in Section 4, given a certain timer interval, the token bucket depth should be at least $TimerInterval \times TokenBucketRate + (PacketSize - 1)$ to prevent token loss. Because the token bucket size grows linearly with the timer interval, using fine-grained timers allow applications to request smaller token buckets. Because each router must be able to buffer a burst the size of the entire token bucket for each application, the size of these buckets has a direct impact on the amount of buffer space required in network routers.

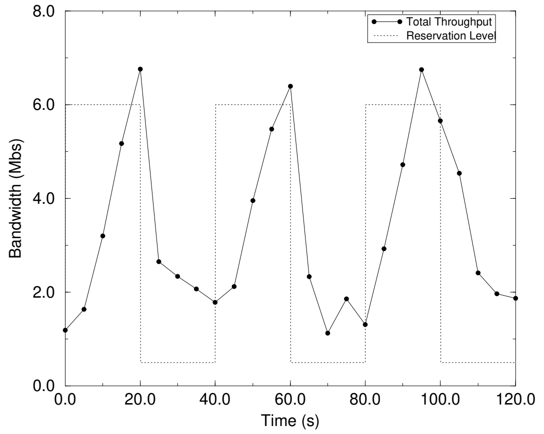
Figure 9 shows the impact that the timer-interval has on the throughput of the `4Mbs` connection using the same multi-hop network setup. The simulations were run using both the timer and windowing modifications as described in Section 4. As Figure 9(a) shows, as the timer interrupt interval increases, the throughput of this connection drops considerably. The reason why this drop is so dramatic is that the lack of buffer space in the network causes a significant amount of burst losses. Burst losses severely limit the throughput of Reno TCP-variants since it takes one round-trip time to recover from each loss. This causes the sending TCP to degenerate into a stop-and-wait protocol. Figure 9(b) shows the results of the same experiment using buffers which are twice the size (`160KB`). With significantly larger buffers, the connection is able to get its share of the bandwidth over a larger range of timer interrupt intervals.

6 Transient Behavior

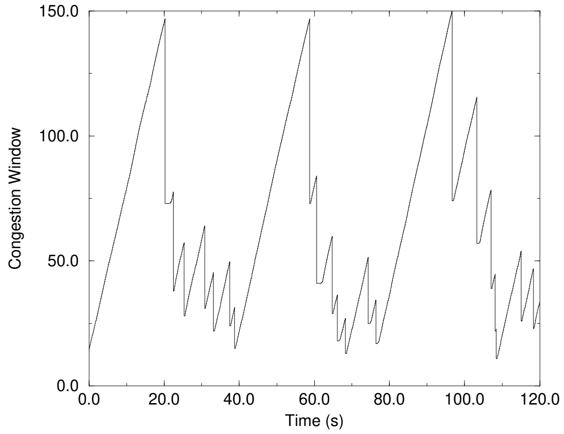
To take a closer look at the transient behavior, we took a controlled-load connection and varied its reservation in the presence of several best-effort connections. In particular, a controlled-load connection which toggled its reservation from 0.5Mbs to 6Mbs every 20 seconds was run between `n0` and `n5`. Four pairs of best-effort connections were also run between these two nodes. Figure 10(a) plots the total throughput of the controlled-load connection using unmodified TCP. This connection uses a token bucket depth of 800ms in order to prevent a large amount of token loss.[6] The plot also shows the reservation that the connection has over time. As shown in the figure, the bandwidth received by the connection reacts slowly to increases in the reservation while it reacts quickly to the decrease in reservation. This is directly attributed to the additive increase/multiplicative decrease property of TCP's windowing algorithm [13]. Figure 10(b) shows the congestion window trace for the normal TCP source. The graph shows the congestion window linearly increasing in response to an increase in reservation level at times `t=0s, t=40s,` and `t=80s`. Thus, by the time the window size reaches a size which can support the size of the reservation, almost the entire 20 second interval has elapsed. This is why the total throughput of this connection lags behind the reservation change.

Figure 11(a) shows the same experiment, but with the reserved connection using the timer and windowing modifications described earlier. Note that the throughput of the connection immediately reacts to both the increase and decrease in reservation levels. We have also plotted the congestion windows of the controlled-load connection for each experiment over the same time periods. Figure 11(b) shows the congestion window trace of the connection. The window size in this case reacts more quickly to the change and thus allows the connection to get its reserved rate. One advantage of rate-based windowing is that the congestion window immediately reflects any changes in reservation level, especially when the level is

[6]Note that the size of the buffers on each interface is 100KB, which is enough to absorb the large bursts that are caused by the deep token bucket.



(a) Throughput

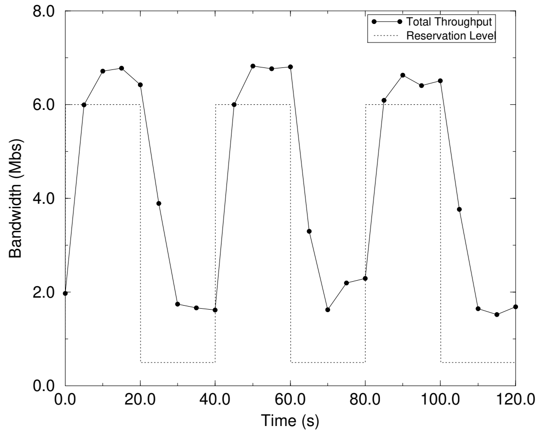


(b) Congestion window trace

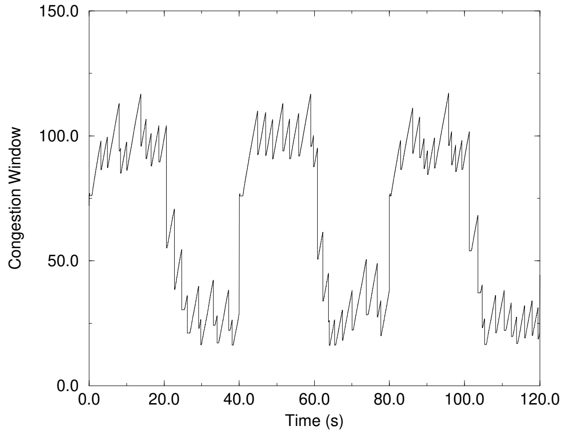
$(BucketDepth = 800ms)$

Figure 10: Dynamics of unmodified TCP.

increased. For TCP-variants which use fast retransmit and fast-recovery, the connection is often in a state where it is increasing its congestion window additively in order to find a congestion window that indicates the amount of bandwidth available to it in the network. Thus, when the reservation for these sources is increased, the congestion window reacts by additively increasing itself to a new window which will support the reserved rate. With a large change in reservation, this can take a fairly long time, especially for connections with large round-trip times.



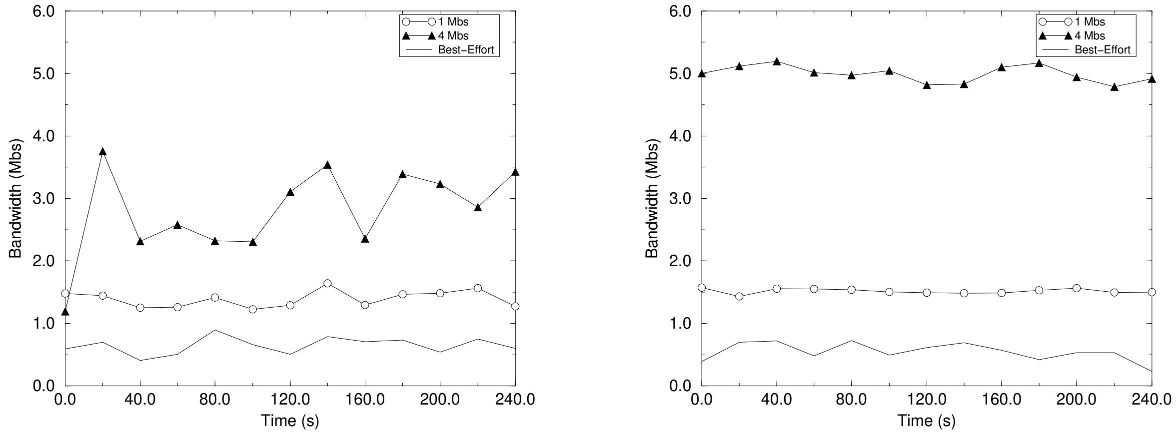
(a) Throughput



(b) Congestion window trace

$(BucketDepth = 50ms, TimerInterval = 20ms)$

Figure 11: TCP with timer and windowing modifications.



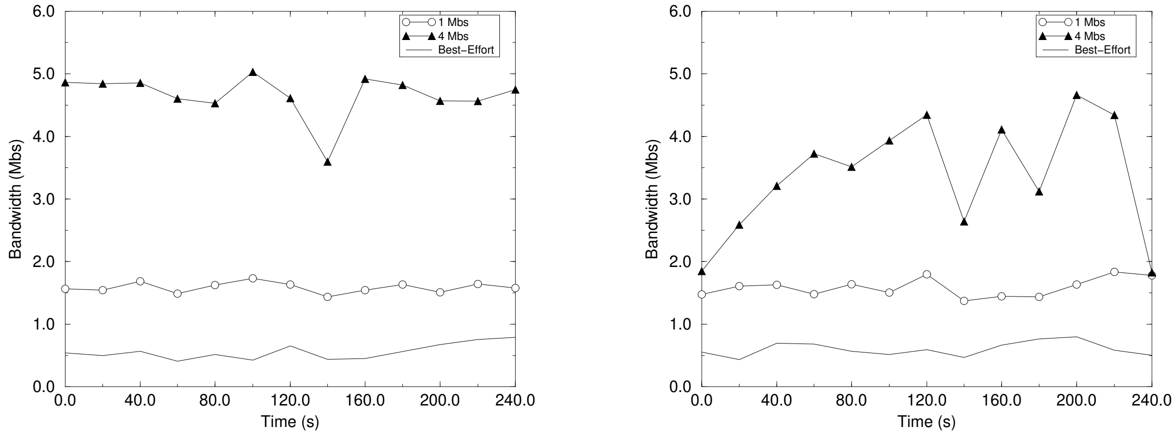
Reserved connections using RENO variant   Reserved connections using SACK TCP.

$(Q_{size} = 100K)$

Figure 12: All nodes use Drop-Tail queues.

7 Path of Evolution

For the experiments reported in previous sections, we have assumed that all routers on the path of a connection employ the ERED mechanism. While this is desirable, modifications to the Internet infrastructure will be incremental and evolutionary. To understand the impact of this heterogeneity in the network, we consider scenarios where none or only a selective subset of routers are ERED-capable.



(a) Bottleneck links (`n1-n2`) and (`n3-n4`) using ERED.   (b) Non-bottleneck links (`n0-n1`) and (`n4-n5`) using ERED.

$(min_{th} = 20K, max_{th} = 80K, Q_{size} = 100K)$

Figure 13: Effect of selective reservation.

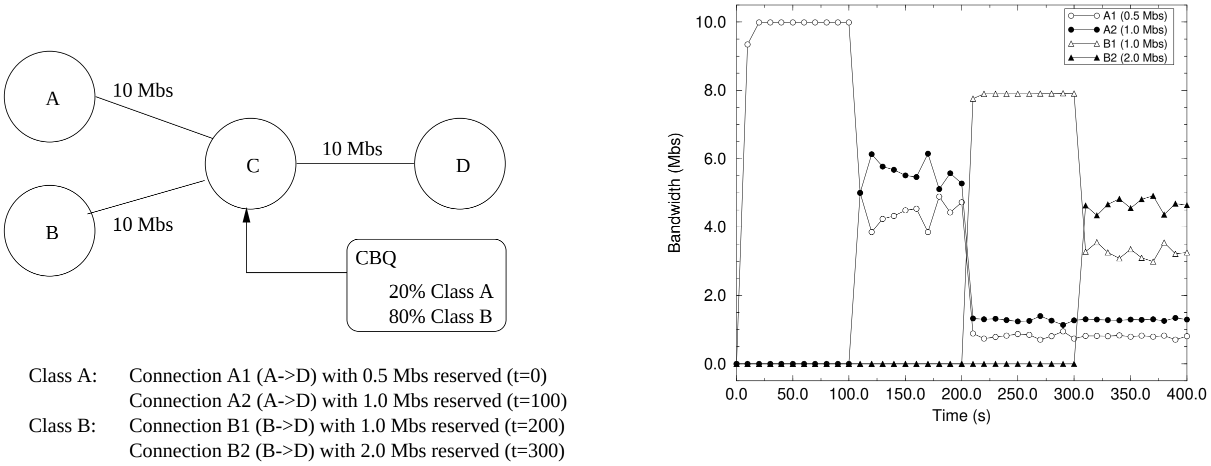
The purpose of the first experiment is to examine the impact of non-ERED gateways on reserved connections. In this experiment, we have two TCP connections with reservations of 1Mb/s and 4Mb/s as well as a best-effort connection between nodes n0 and n5. Pairs of best-effort connections run between nodes n6 and n7 and between n8 and n9. Plots in Figure 12(a) show the throughput seen by these connections when all the routers are drop-tail routers which do not distinguish between marked and unmarked packets. The connections with reservations use the timed transmission mechanism and the modified windowing scheme. Note that while no service differentiation is being done on the links, it is assumed that adequate admission control and provisioning is being done in order to ensure that the offered load does not overwhelm network links and cause congestion collapse.

As shown in Figure 12(a), the reserved connections indeed see higher throughput than the best-effort connection between the same nodes. However, the absence of ERED mechanisms in routers adversely impacts the performance of the connections with reservation. This situation worsens with the increase in the reservation level. This is because a connection with a higher reservation transmits a larger amount of data and thus, the number of packet drops it experiences is proportionally higher. In addition, the connection with a larger reservation transmits data in larger bursts and is susceptible to burst losses in the drop-tail queues. This is reflected in the oscillation in the bandwidth curve for the connection with a 4Mb/s reservation. Burst losses, as described earlier, interact poorly with RENO sources since it takes a full round-trip time to recover from each loss. This freezes the sender and prevents it from getting its reserved rate. A packet trace of this connection verifies this behavior. Figure 12(b) shows the same experiment with the reserved connections using SACK TCP. As shown in the figure, despite the absence of ERED queueing, all connections manage to achieve aggregate throughputs close to their respective desired levels.

From the results of these experiments, it is fair to conclude that even without sophisticated scheduling mechanisms it may be possible to extend the paradigm of equal sharing of network capacity to one where it is shared in accordance with allocations. Note, however, that in the experiments here, the aggregate reservation is lower than the network capacity. We also assume cooperation among different TCP connections. This means that admission control and voluntary adherence to the socially cooperative congestion control is still required. The ERED mechanism provides additional protection to compliant controlled-load traffic against best-effort and non-compliant controlled-load traffic.

To examine the impact of ERED-capable routers at selected places, we repeated the experiments described above using RENO sources with only a few selected routers employing ERED. Figure 13(a) shows the throughput of the same set of connections when interfaces only at the bottleneck links between n1 and n2 and between n3 and n4 employ ERED queues while the rest of the interfaces use drop-tail queues. As the plot shows, placing ERED queues at bottleneck points in the network is sufficient to provide connections their allocated share of the bandwidth. However, as shown in the figure, the connection sometimes dips below its reservation level due to burst losses which can occur in the presence of drop-tail queues. Figure 13(b) shows the throughput of the connections when the ERED queues are placed on non-bottleneck links. As with the drop-tail experiments in Figure 12(a), the performance of the high bandwidth connection suffers throughout.

The results from these experiments demonstrate that there is an effective path of evolution of integrated



(a) Network Topology. (b) Throughput.

Figure 14: CBQ experiment.

services in the Internet. For the integrated services to be useful, it is not required to upgrade the entire infrastructure at the same time. There is substantial value in following an evolutionary path where at first the control mechanisms at the end-hosts are modified and routers support admission control. Enhanced queuing mechanisms, such as ERED, can then be deployed at observed bottlenecks and then gradually throughout the network. In the next section, we discuss how this evolutionary approach can be continued beyond the ERED mechanism.

8 CBQ and ERED

The experiments in this paper have shown how to effectively provide minimal bandwidth guarantees in the network using ERED gateways. While this service may be useful to a large class of applications, in a fully-evolved integrated services Internet, such a mechanism must coexist with other mechanisms for providing a range of alternative services. This will allow applications written using such a service, to continue to work as the Internet infrastructure is upgraded and more sophisticated packet and link scheduling support is put into place.

The ERED mechanism can be easily embedded into more sophisticated scheduling disciplines such as class-based queueing (CBQ) [8]. CBQ is one of the more popular mechanisms proposed for packet and link scheduling in an integrated services Internet. In CBQ, datagrams belonging to different classes are put in different queues in the routers. The queues are serviced in different priority order based on the allocation given to the associated traffic class. Embedded as a class in CBQ, this ERED can be used to provide weighted bandwidth sharing between connections of a class. By aggregating connections with varying bandwidth requirements in one class, we reduce the total number of classes in a class-based queue and thus, the overhead in the link scheduling. To examine this possibility, we embedded the ERED queue into the CBQ implementation of the NS simulator. We then examined its performance in the network shown in Figure 14(a). This network consists of two agencies, A and B, who share a common link (between nodes C and D) to a provider's network. In this setup, agency A is entitled to 20% of the link's bandwidth while agency B is entitled to the remaining 80% of it. Node C uses CBQ with 20% of the link share allocated to traffic from agency A, and 80% allocated to traffic from agency B. Note that while either one of the two agencies is idle, the other, active agency, is entitled to use the entire link for itself. In addition, since the allocation is relative, admission control must assume the worst case scenario in admitting flows to a class. That is, it must assume all other classes are using their share of the link. Both queues within the CBQ system use the ERED mechanism to share bandwidth between individual connections.

Figure 14(b) shows the throughput seen by connections originating from A and B and traversing the link between C and D. Connections A1 and A2 originate from agency A and have reserved rates of `0.5Mbs` and `1Mbs`, respectively. They start at times `0s` and `100s`. Connections B1 and B2 originate from agency B and have reserved rates of `1Mbs` and `2Mbs`. These connections start at times `200s` and `300s`, respectively. As the graph shows, between `0s` and `100s`, connection A1 gets all of the bandwidth since it is the only active connection. Between `100s` and `200s` (after connection A2 starts) the link's bandwidth is shared between connections A1 and A2. However, since A2 has a `1Mbs` reservation, it gets slightly more total bandwidth than A1. When B1 starts at `200s`, it is the only active connection from agency B. Hence, it receives the entire 80% of the link's bandwidth (`8Mbs`). The two connections from agency A then share the remaining bandwidth (`2Mbs`) according to their reservations. Finally, at `300s`, connection B2 starts and the `8Mbs` allocated to agency B is split between connection B1 and B2 in accordance with their reservations, that is, B2 gets approximately `1.0Mbs` more than B1. What happens throughout the course of this experiment is that when the class is allowed to be overlimited, the ERED queue is drained at a sufficient rate so as to support higher rates of input data. As soon as the class becomes regulated, the queue builds up, the ERED queue drops unmarked packets and the connections in the class resumes sending at a lower rate.

9 Conclusions

We have examined ways of providing a large class of overload-sensitive Internet applications with a useful service using minimal enhancements to the network infrastructure. Towards this end, we have proposed a simple extension to the packet queueing and scheduling mechanism at the routers. Assuming that the network supports minimum rate guarantees, we have also proposed and experimented with modifications to TCP's congestion control algorithm. The proposed modifications allow connections with reservations to obtain their reserved rates and share the residual unreserved bandwidth with the best effort connections. It is important to note that the modified congestion control algorithm should be exercised if and only if the network supports minimum rate guarantees through end-to-end signaling, admission control, and resource reservation. Without such mechanisms in place, the use of modified TCP sources may cause congestion collapse in networks.

The study reported in this paper can be extended in many ways. We are working on implementing and experimenting with the mechanisms proposed here in a network testbed. We are also considering applications of this work in the context of other transport protocols, especially RTP and UDP. Many multimedia applications do not require the reliable delivery that TCP provides. While this study focuses on TCP, implementing a similar scheme using RTP and UDP fitted with TCP's flow-control mechanism is possible.

Another key area of future work is the admission control policies for such a service. While we have not addressed this aspect here, we plan on using our observations on token bucket depths, router buffer sizes, source burstiness, and ERED-parameterization to develop admission control policies for this service.

We also plan on examining the effect of round-trip time on end-to-end throughput. A number of studies have shown that relative throughput seen by best-effort connections sharing a common bottleneck link has a strong dependence on their round-trip times. Preliminary results (not reported here) from our experiments show that the compliant throughput of connections with reservations is largely independent of their round-trip times. However, the non-compliant part of the throughput and the throughput seen by best-effort connections are sensitive to round-trip time. A thorough investigation is underway to explore this aspect in more detail.

On a final note, any allocation-based sharing of network capacity has to be associated with a policy and/or pricing scheme. We believe that the proposal of prioritizing a part of a traffic stream with marking and competing with best-effort traffic for sharing the residual capacity fits in very well with pricing schemes which are currently being considered for the Internet. Users paying for a certain level of marking see incrementally better performance over those who do not. During times of light loads, when the incremental costs of congestion are low, the user can decrease his/her level of bandwidth reservation and costs until an acceptable level of aggregate throughput is observed.

References

[1] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: An Overview. *RFC 1633*, June 1994. ISI/MIT/PARC.

[2] I. Cidon, R. Guerin, and A. Khamisy. Protective Buffer Management Policies. *IEEE/ACM Transactions on Networking*, 2(3), June 1994.

[3] D. D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Proc. of ACM SIGCOMM*, pages 14–26, August 1992.

[4] A. Costello and G. Varghese. Redesigning the BSD Callout and Timer Facilities. Technical Report WUCS 95-23, Washington Univ. of St. Louis, November 1995.

[5] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of Fair Queuing Algorithm. In *Proceedings of SIGCOMM*, 1989.

[6] W. Feng, D. Kandlur, D. Saha, and K. Shin. TCP Enhancements for an Integrated Services Internet. Technical Report RC 20618, IBM Research, November 1996.

[7] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *ACM/IEEE Transactions on Networking*, 1(4):397–413, August 1993.

[8] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[9] S. Golestani. A Self-Clocked Fair Queuing Scheme for Broadband Applications. In *Proceedings of INFOCOM*, June 1994.

[10] P. Goyal, H. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM*, pages 157–168, August 1996.

[11] R. Goyal, R. Jain, S. Kalyanaraman, and S. Fahmy. UBR+: Improving Performance of TCP over ATM-UBR Service. Submitted to ICC 1997: http://www.cis.ohio-state.edu/ jain/icc97.ps, 1996.

[12] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *Proceedings of ACM SIGCOMM*, pages 270–280, August 1996.

[13] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM*, pages 314–329, August 1988.

[14] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. *RFC 1323*, May 1992. LBL/ISI/BSDI.

[15] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A Measurement-based Admission Control Algorithm for Integrated Services Packet Networks. In *Proc. of ACM SIGCOMM*, pages 2–13, August 1995.

[16] S. McCanne and S. Floyd. http://www-nrg.ee.lbl.gov/ns/. ns-LBNL Network Simulator, 1996.

[17] P. Mishra. Effect of Leaky Bucket Policing on TCP over ATM performance. In *Proceedings of ICC*, 1996.

[18] E. Rathgeb. Modeling and Performance Comparison of Policing Mechanisms for ATM Networks. *IEEE Journal on Selected Areas of Communication*, 9(3), 1991.

[19] J. Rexford, A. Greenberg, and F. Bonomi. Hardware-Efficient Fair Queueing Architecture for High-Speed Networks. In *Proceedings of INFOCOM*, March 1996.

[20] M. Shreedhar and G. Varghese. Efficient Fair Queuing using Deficit Round Robin. *IEEE/ACM Transactions on Networking*, 4(3), June 1996.

[21] J. Wroclawski. Specification of controlled-load network element service. *RFC 2211*, September 1997. MIT.

[22] L. Zhang, S. Shenker, and D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proceedings of ACM SIGCOMM*, pages 133–148, August 1991.