TCP Enhancements for an Integrated Services Internet

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

†Real-Time Computing Laboratory
Department of Electrical Engg. & Computer Sc.
University of Michigan, Ann Arbor, MI 48109
{wuchang,kgshin}@eecs.umich.edu

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

Abstract

Flow and congestion control mechanisms in TCP have been designed with the premise that network capacity should be shared equally among competing connections. With the evolution of the Internet from a class-less best-effort network to an integrated services network with different classes of service, the premise of equal fair sharing is no longer true. In an integrated services Internet, network bandwidth should be shared in accordance with the reservation made for each session. In this paper, we consider a specific class of service, namely *controlled-load*, which is currently under consideration for deployment in the Internet. We propose a simple scheme to realize this service and study its interaction with the control mechanisms used in TCP. In light of this study, we propose a number of enhancements to the control mechanisms used in popular implementations of TCP. Using deatiled simulation experiments, we show that with the proposed enhancements in place, TCP is indeed able to provide the desired end-to-end behavior.

1 Introduction

The Internet is on a fast path of evolution from a best-effort data network to an integrated services network carrying voice, video, image, and data on the same fabric. With applications such as Web TV, NetRadio, and Internet Phone pushing the frontier, the need to provide better than best-effort service on the Internet is becoming a necessity. In order to support these emerging applications better, the Internet Engineering Task Force (IETF) is developing a set of protocols and standards [2, 20, 23] with the goal of providing a number of new classes of service in the Internet. The service classes currently under consideration are *controlled-load* [23] and *guaranteed service* [20]. Controlled-load service is an enhancement to best-effort service that guarantees a minimum level of network performance at all times. Each controlled-load connection has a traffic envelope associated with it. Traffic that conforms to this envelope should observe an end-to-end network behavior that is similar to that in an unloaded network. Traffic not conforming to this envelope is treated as best-effort traffic. This is in contrast to guaranteed service which provides firm guarantees for lossless on-time delivery of packets. A consequence of the less stringent nature of the performance guarantees provided by controlled-load service is that it provides more latitude for implementors and is potentially easier to realize and simpler to deploy. Since the existing Internet infrastructure is engineered to provide best-effort services, and since upgrades to this infrastructure will most likely be evolutionary, it is anticipated that controlled-load service will be the first non-best effort service to be supported on the Internet.

The controlled-load service paradigm opens up a new dimension in the design space of applications which have been developed for use in today's Internet, but are sensitive to overload conditions. For example, increasingly popular applications such as PointCast [15], Real Audio [16], VDOnet [22], and Bamba [10] stream text/image, audio, and video data over the Internet. Depending on the network load, the quality of the transmission can vary from very good to intolerable. With the availability of controlled-load service, these applications can be tuned to provide a minimal quality of video and audio at all times with a possibility of improved service during periods of light network load. Similar enhancements are possible for a large number of other networked multimedia applications, such as `vic`, `vat`, `nevot`, and `wb`. Controlled-load service is also useful to some of the more traditional applications such as `ftp` and `telnet`. For example, booting a diskless workstation over the network, backing up remote files, or synchronizing web proxies can be performed with more predictability and within a bounded time by guaranteeing a minimal bandwidth to the underlying sessions. Controlled-load connections 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.

Realization of controlled-load service requires enhancements to the existing Internet infrastructure. The network needs to distinguish between conformant datagrams belonging to connections with reservations and treat them differently from best-effort and nonconformant controlled-load traffic. Our goal in this paper has been to limit these enhancements to a minimal level. We propose a simple extension to the queuing mechanisms used in today's routers coupled with a policing and marking scheme at the source. In our scheme, controlled-load traffic is policed at the source and conformant packets are marked. Non-conformant controlled-load traffic and best-effort traffic is injected into the network unmarked. At the routers we use an enhanced random early detection (RED) [5] algorithm. 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 in the queue have the same drop probability, in the enhanced RED gateway, marked packets have a lower drop probability than the unmarked packets. There are other ways of realizing controlled-load service, but we show

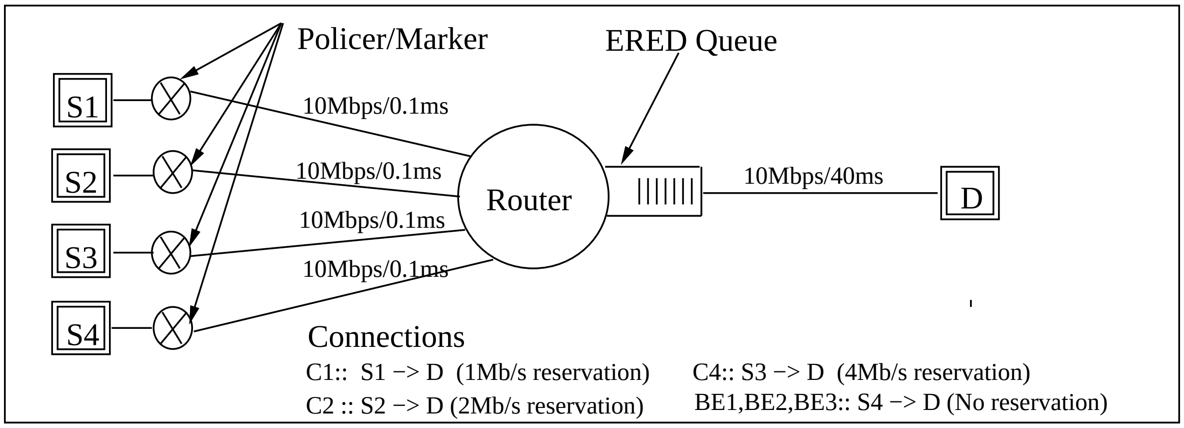

Policer/Marker
ERED Queue
S1
S2
S3
S4
10Mbps/0.1ms
10Mbps/0.1ms
10Mbps/0.1ms
10Mbps/0.1ms
Router
10Mbps/40ms
D
Connections
C1:: S1 -> D (1Mb/s reservation)
C2 :: S2 -> D (2Mb/s reservation)
C4:: S3 -> D (4Mb/s reservation)
BE1,BE2,BE3:: S4 -> D (No reservation)

Figure 1: An example network configuration.

that even with these simple enhancements we can effectively provide this service in the Internet.

The objective of this paper is to study the impact of controlled-load service on the end-to-end performance seen at the transport layer. Specifically, our focus is on TCP primarily because (1) the overwhelming number of applications that use TCP as the transport protocol including most web applications using HTTP, and (2) TCP has a well-developed congestion control mechanism which is able to evenly share bandwidth among best-effort connections. While even bandwidth sharing isn't necessarily a goal of controlled-load service, some form of fair sharing of network bandwidth is required. In the absence of resource reservation, TCP strives to share the network capacity equally among competing connections. In an integrated services Internet that supports both controlled-load and best-effort services the bandwidth sharing paradigm is slightly different. A TCP connection with a controlled-load reservation should receive its reserved bandwidth and be eligible to receive a fair share of any residual capacity in the network. Consider the scenario in Figure 1. In this example, three connections with reservations of 1Mb/s, 2Mb/s, and 4Mb/s share a 10Mb/s link with three other connections with no reservations. Classical TCP strives to share the link bandwidth equally among the six connections leading to an ideal throughput limit of 1.67Mb/s for each connection. In the new paradigm, the connections with 1Mb/s, 2Mb/s, and 4Mb/s reservations should receive shares of 1.5Mb/s, 2.5Mb/s, and 4.5Mb/s, respectively, which represent their reserved shares and a fair share of the residual capacity. The ideal limit for the best-effort connections is 0.5Mb/s.

In the rest of the paper, we discuss enhancements to TCP's control algorithms in tune with this new paradigm. We propose two extensions to TCP's congestion and flow control mechanisms – (1) an adaptive windowing mechanism that takes into account the reservation associated with the connection, and (2) a timed/delayed send mechanism that reduces the dependency on acknowledgment triggered transmissions to keep the reserved pipe between the sender and receiver full. These enhancements have been applied to both Reno and SACK [13] TCP senders. No additional changes are required at the TCP receiver. Using detailed simulation experiments, we have shown that with the proposed enhancements in place, TCP provides the desired end-to-end behavior. Our initial investigations into incorporating these extensions in TCP implementations in BSD-variant operating systems indicate that the overheads of these enhancements are minimal and are far outweighed by their potential benefits.

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 [8] address similar issues. However, due to significant differences between the service architectures of ATM and the Internet, the proposed solutions are quite different. In [8], the authors propose using a modified switch buffer allocation policy in order to obtain peak throughputs and fairness between TCP connections over ATM. In particular, each connection is effectively given 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 in the TCP sender. 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.

The rest of the paper is organized as follows. In Section 2 we briefly review the different classes of service that are being considered for deployment in the Internet. Performance of TCP in an integrated services Internet is investigated in Section 3. In Section 4 we present enhancements to TCP's windowing mechanisms to make it adapt to the rate-based reservation paradigm. Section 5 is devoted to a discussion on timed and delayed transmissions. Section 6 considers the impact of multiple losses and mechanisms to recover from such losses. We conclude in Section 7.

2 Controlled-Load Service and Its Realization

The Integrated Services working group in the Internet Engineering Task Force (IETF) is responsible for defining and standardizing the service specifications that will transform today's best-effort Internet into an integrated services network providing different classes of service. The two classes of service that are currently under consideration for standardization are guaranteed service and controlled-load service. In this paper our focus is on the controlled-load service primarily because (1) it is likely to be the first non-best-effort service to be supported on the Internet, and (2) the potential beneficiaries of this service include many of today's Internet applications.

According to the IETF specification, the end-to-end behavior provided to an application by controlled-load service closely approximates the behavior visible to applications receiving best-effort service under `unloaded` network conditions. That is to say, a very high percentage of transmitted packets will be successfully delivered by the network to the receiving end-nodes, and the transit delay experienced by a very high percentage of delivered packets will not greatly exceed the minimum transit delay experienced by any successfully delivered packet. Clearly, there is room for different interpretations of this service specification. Controlled-load service is not intended for applications that require firm guarntees on end-to-end delay, but is instead designed to provide congestion relief to a broad class of existing Internet applications. In other words, the class of applications that are likely to benefit from controlled-load service neither requires nor expects hard guarantees on network performance and a precise defintion of the service specification.

To avail controlled-load service, a connection has to specify a traffic envelope, called *Tspec*. Tspec includes a long-term average rate t_m, a short-term peak rate t_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, peak rate could be the link bandwidth at the source, and the burst size could be the maximum size of a frame. Tspec also specifies the maximum and minimum packet sizes to be used by the application. Compliant traffic belonging to a controlled-load session can expect to see a service equivalent to that observed in an "unloaded network," while traffic that violates the session's Tspec is treated as best-effort traffic. As mentioned earlier, this service specification, especially the term "unloaded network", is open to various interpretations. Any real network is never unloaded in the true sense of the term, and hence no packet ever sees a service equivalent to that in an unloaded network. A more pragmatic interpretation of this specification is a service architecture where sufficient network resources are reserved to provide a priority service to a traffic stream defined by the Tspec. Traffic in excess of the Tspec is treated as best-effort. In the following, we discuss a possible realization of controlled-load service based on this interpretation including the one used in the experiments reported in this paper.

Two rather well-researched means of realizing different classes of service in an integrated services network is to use class-based queuing [1,6] and weighted fair queuing [4,7,19,21]. In a class-based queuing scheme, datagrams belonging to different classes are put in different queues in the routers. The queues are serviced in different priority order based on the associated traffic class. A possible realization of controlled-load service using class-based queuing is by maintaining two queues – one for the conformant controlled-load traffic, and another queue for both non-conformant controlled-load and best-effort traffic. The apparent problem with this implementation is that it separates the conformant and nonconformant traffic from controlled-load connections into two queues. Since the queues are served in different priority order, this may lead to a large number of datagrams being delivered out of sequence. Although out of sequence delivery of datagrams do not violate either the controlled-load or IP specifications, it is often detrimental to the performance of transport protocols such as TCP. Per-session fair queuing is another well-studied mechanism to realize different classes of service in an integrated services network. One possible way to realize controlled-load service is to treat each controlled-load connection as a separate session and reserve a rate of service commensurate with its Tspec. Any excess traffic will automatically receive a fair share of the residual capacity. While per-session fair queuing provides an effective way to implement controlled-load service, the underlying scheduling mechanism may be expensive, especially in the overloaded and under-powered routers in today's Internet.

Ideally, we would like to realize contolled load service with minimal changes to the end-hosts and routers. While implementations may have varying degrees of network and end-host modifications, it is expected that each scheme will have the ability to (1) classify controlled-load from best-effort datagrams and (2) be able to treat these datagrams differently. We achieve these two characteristics by employing policing and marking at the source as well as simple enhancements to the FIFO queuing structure at the routers.

In this scheme, connections with reservations 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. We use token buckets [18] for policing and marking. The token generation process follows the Tspec advertised by the source. That is to say, 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 number of tokens are available, it is sent as a marked packet and an equivalent number of tokens are considered consumed. If sufficient tokens are not present at the time of transmission of the packet, it is sent as an unmarked packet. All best-effort traffic is sent as unmarked datagrams. 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. Marking of packets facilitates the realization of controlled-load service, however, it is not a requirement. 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 source and not at the routers. In the rest of the paper, we assume that a marking facility is available.

In order to handle marked and unmarked packets differently, we propose using enhanced RED (Random Early Detection) at the routers. In classical RED routers a single FIFO queue is maintained for all packets. Packets are dropped randomly with a given probability when the queue length exceeds a certain threshold. The random drops continue until the queue length drops below the threshold. The drop probability itself depends on the queue length and the time elapsed since the last packet was dropped. Because these RED gateways control the size of the queues, they are ideal for supporting the controlled-load service using a single queue. Queue lengths can be kept low by dropping packets before the queue fills up. Studies have shown that RED gateways are better than traditional head-drop or tail-drop routers. Enhanced Random Early Detection (ERED) is a minor modification to the original RED algorithm. ERED routers handle both marked and unmarked packets. They both share the same FIFO queue as in the original case, but the drop probabilities of the marked packets are lower than unmarked packets.

This scheme can also be embedded into more sophisticated scheduling disciplines such as class-based queuing (CBQ) and weighted fair queuing (WFQ). These scheduling disciplines provide mechanisms for bandwidth sharing between different classes/sessions. The ERED queue may be operated as a class in a CBQ system or as an aggregated session in a WFQ system. For example, all reserved sessions that use TCP as the transport protocol could be aggregated into a class with the ERED discipline. With the RSVP reservation protocol such a classification can be accomplished by examining the protocol type for the RSVP session [2]. In this environment, it would provide a mechanism for buffer sharing and loss-priority enforcement amongst a set of reserved sessions.

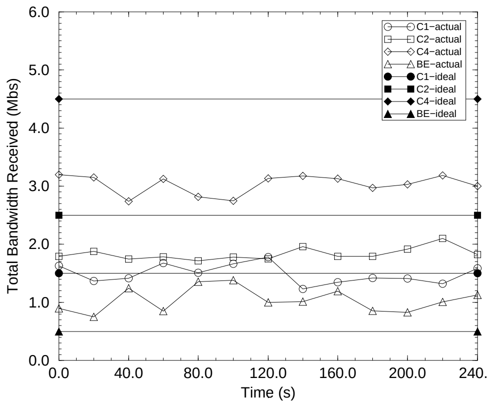
The scheme described above is consistent with the service definition of the controlled-load service class. By keeping the queue length within a configurable limit, it is possible to control the delay seen by different packets. By controlling drop probabilities, it is possible to keep the loss rate of conformant controlled-load datagrams as low as possible. It will be shown later that this simple mechanism is effective in providing the desired service.

3 TCP in an Integrated Services Internet

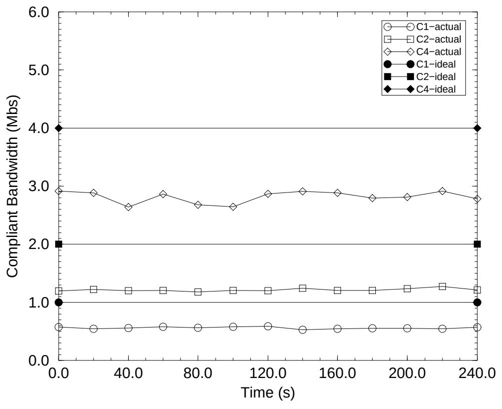
In order to understand the dynamics of TCP over an Integrated Services Internet we simulated a very simple scenario as shown in Figure 1. Our objective here is to study the interaction of controlled-load service with the flow and congestion control mechanisms used in TCP on a single bottleneck link. Later in the paper, we consider more complex network topologies. In Figure 1, the circles indicate finite-buffer ERED routers, and the squares indicate transmitting or receiving hosts. In this example we have six TCP connections going to a common destination through a router and sharing a bottleneck link of bandwidth 10Mb/s. Three connections, C_1, C_2, and C_4 have reservations of 1Mb/s, 2Mb/s, and 4Mb/s, respectively. The other three are best-effort connections. In all the experiments reported in this paper, the links in the reverse direction are uncongested. We assume that the segment [1]size used by TCP is 1KB and that the router maintains an 80 segment (80KB) queue and employs ERED as described in the last section. The links are labeled with their respective bandwidths and transmission delays. As described in the last section, data transmitted by a TCP sender is passed through a policer. The policer is implemented using a token bucket with token generation rate equal to the reserved rate of the connection and a token bucket depth of 50ms. To observe the effects of congestion control exercised by TCP, we set the receiver's advertised window to 100 segments which is equal to the delay bandwidth product of the unloaded network.

We simulated the scenario described above using a modified NS [14] simulator. The NS simulator

[1]We use segment and packet interchangeably in this paper.



(a) Total throughput.



(b) Compliant throughput.

Figure 2: Throughput seen by New-Reno TCP sessions.

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 the experiments reported here, we use the Reno version of TCP, which is the most widely used implementation of TCP. We extended the simulator by adding policing and extending the RED queuing discipline. In our simulation, all senders and receivers use TCP Reno and sources are greedy. TCP segments belonging to reserved connections are sent 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 the best-effort connections are sent as unmarked datagrams.

Figure 2 shows the plot of the throughput (goodput) seen by the receiver for different connections. Throughput is measured by computing data received at the receiver over 20s periods and dividing that by the observation interval. In the ideal case, each connection should receive its reserved bandwidth and an equal fair share of the residual bandwidth. Hence, the expected ideal throughput seen by the connections C_1, C_2, and C_4 should be 1.5Mb/s, 2.5Mb/s, and 4.5Mb/s, respectively. The best-effort connections should see a throughput of 0.5Mb/s. We observe from the graphs that this is not quite the case. While connections with higher reservations receive a higher share of the bandwidth, it is not close to their respective ideals. On the other hand, the best-effort connections receive more than their expected share of residual bandwidth. We have also plotted the compliant bandwidth received by different reserved connections. This is computed by counting the marked segments received by the receiver over 20s periods and then dividing that by the observation interval. As observed from the figure, compliant bandwidths received by different connections are far less than their respective reservations.

The observations from Figure 2 can be explained if we take a careful look at TCP's windowing mechanisms. The TCP sessions with reservations exercise their congestion window the same way as the best-effort connections. However, they have a lower probability of losing a packet at the gateway since their marked packets have lower (in this case close to zero) probability of getting dropped. As a matter of fact, the probability of packet drop decreases for higher levels of reservation. That is precisely the reason why C_4 receives more throughput than C_2, why C_2 receives more bandwidth

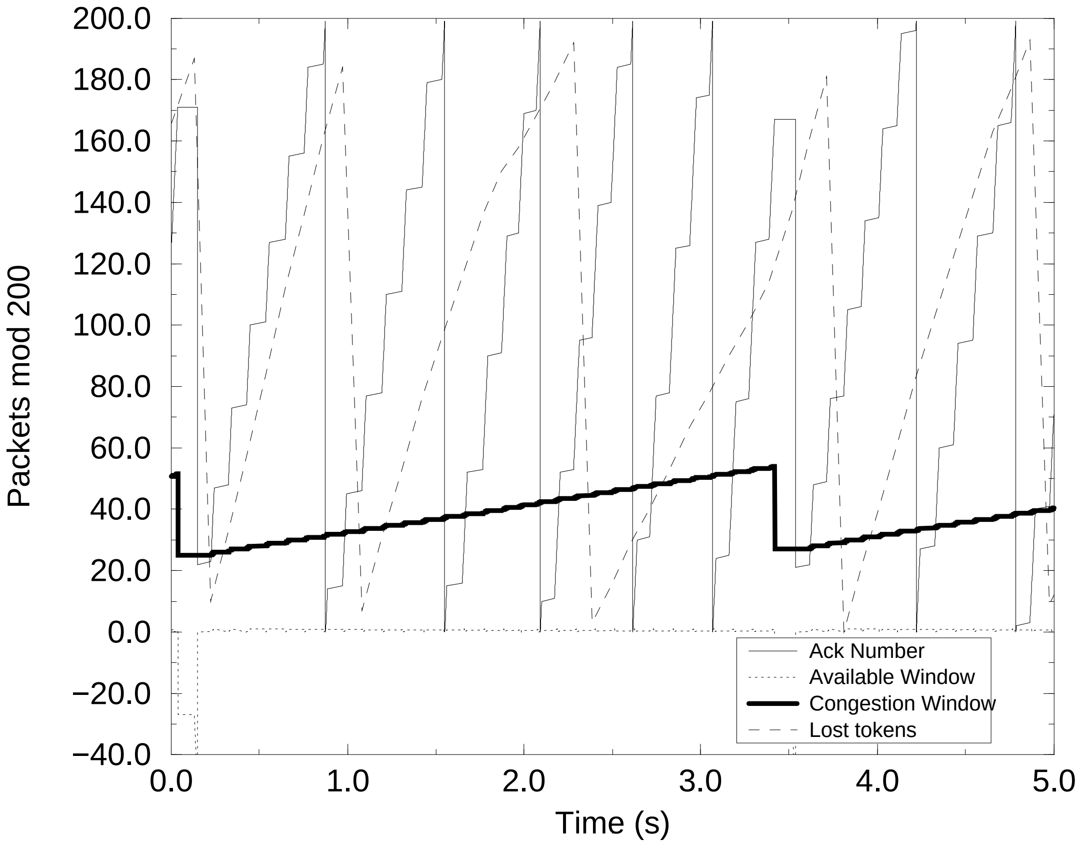


Figure 3: Packet trace of C_4 running New-Reno TCP.

than C_1, and why C_1 sees higher throughput than the best-effort connections. However, as observed from Figure 2, the windowing mechanism used by TCP Reno is not able to fully exploit the benefits of reservation. Since marked packets are not dropped by the network, this suggests that not all tokens generated are consumed to mark packets at the source, and there are token losses due to token bucket overflow. That is, although the sender is a greedy source, the TCP windowing mechanism throttles the source to an extent that makes it impossible for it to send data to fill up the reserved portion of the communication pipe. To confirm this speculation, we plotted the packet trace from connection C_4.

Figure 3 shows the packet trace of C_4 over a five-second interval. We have plotted the cumulative acknowledgements received by the sender, the total numbers of tokens lost, the sender's congestion window, and the sender's available transmission window against time [2]. A close look at the packet

[2]The available transmission window measures the difference between the total number of unacknowledged packets the sender has outstanding and the congestion window. On a loss, when the congestion window is reduced, this window may be negative.

```
(1) After every new acknowledgment
        if (CWND < SSTHRESH) CWND ← CWND+1
        else CWND ← CWND+1/CWND
        endif

(2) When NDUP exceeds a threshold
        CWND ← RWND+(CWND-RWND)/2+NDUP
        SSTHRESH ← RWND+(CWND-RWND)/2

(3) Upon timer expiry
        CWND ← RWND+1
        SSTHRESH ← RWND+(CWND-RWND)/2
```

Figure 4: Rate adaptive windowing algorithm.

trace reveals a steady rate of token loss while the source waits for the congestion window to open up. In this trace 830 tokens are lost in 5 seconds. The rate of token loss increases during the recovery phase when the source stays idle after a retransmission.

There are two ways to alleviate this problem – (1) using a deeper token bucket that can store more tokens during the periods of inactivity and use them later, and (2) changing TCP's congestion control mechanisms to maintain a steady stream of packet flow at least at the rate of token generation. The first approach of using a deeper token bucket necessitates a higher level of reservation at the network. Moreover, adjusting the depth of the token bucket to reduce/eliminate token loss depends upon the round-trip delay and other dynamics in the network. In any case, a more comprehensive solution to this problem is to enhance TCP's windowing mechanism to better adapt to the rate based reservation oriented service architecture. In the next section, we explore this option in greater detail.

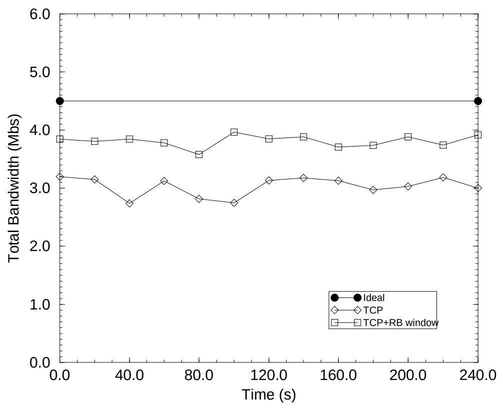
4 Rate-Adaptive Window Control

TCP relies on a window-based protocol for congestion and flow control. In this section we present enhancements to TCP's windowing mechanisms in order to exploit reservations in the network. Before presenting the enhancements, we first discuss windowing mechanisms used in classical implementations of TCP.

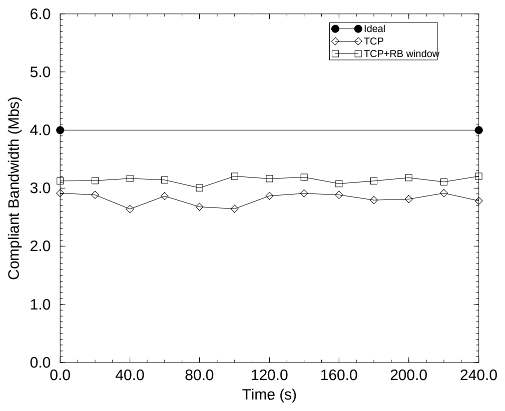
4.1 Classical Window Control

Congestion and flow control in TCP is enforced by two windows. 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 cannot send more than the minimum of AWND and CWND worth of data at a time.

The TCP congestion control scheme consists of two phases: slow start and congestion avoidance.



(a) Total throughput seen by C_4.



(b) Compliant throughput seen by C_4.

Figure 5: Throughput seen by C_4 with rate-based windowing.

TCP maintains a variable SSTHRESH at the source to distinguish between the two phases. At start, CWND is initialized to 1, and SSTHRESH is set to AWND. The source starts transmission in the slow start phase by sending one segment (1KB in our example) of data. When the source receives acknowledgment for a new segment, it increments the congestion window by 1. Since the time between sending a segment and the receipt of its acknowledgment is one round-trip time (RTT), CWND is doubled every RTT. In the absence of any packet loss, the slow start phase continues until CWND reaches SSTHRESH, at which point the congestion avoidance phase is entered. In the congestion avoidance phase, the source increases CWND by 1/CWND every time a segment is acknowledged. Consequently, in the congestion avoidance phase CWND increases by one each round-trip time. The slow start and the congestion avoidance phases correspond to an exponential increase and a linear increase of the congestion window [17] every round-trip time.

The sender can detect a lost packet from either a retransmission timeout or from the receipt of multiple duplicate acknowledgments. The latter mechanism is popularly known as fast retransmit. The acknowledgments sent by the receiver are cumulative and include the sequence number of the next segment expected. When a segment is lost [3], the receipt of subsequent segments trigger acknowledgments with the segment number of the lost segment. If several duplicate acknowledgments are received, the sender infers a loss of a segment. The retransmission timer is set based on a conservative estimate of the RTT and its variance to protect against random variation in the RTT and false firing of the retransmission timer. Consequently, timeout based loss detection is quite slow in terms of responsiveness. Furthermore, TCP uses a coarse-grained timer (typically 500ms) to implement the timeout mechanism in order to minimize the system load due to timer interrupts. As a result, loss detection and subsequent retransmission in response to the receipt of duplicate acknowledgments has been called "fast retransmit".

In the Tahoe [11] implementation of TCP the detection of a lost packet results in CWND being set to 1 and SSTHRESH being set to the minimum of AWND and one-half of CWND. Consequently, the source enters the slow start phase. In the Reno implementation of TCP, the fast retransmit operation has been modified to include a fast recovery mechanism. The new algorithm prevents

[3]This may also happen when a segment is received out of sequence.

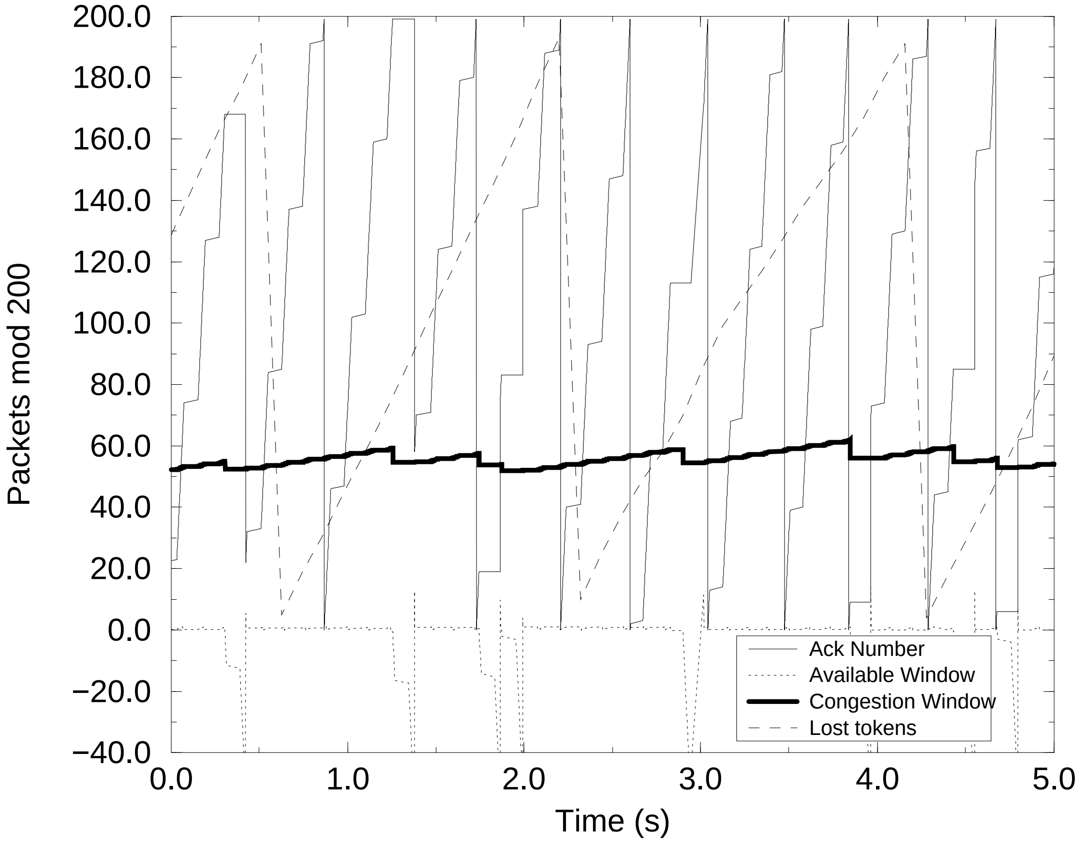


Figure 6: Packet trace of C_4 after windowing modifications.

the communication pipe from going empty after the fast retransmit, thereby avoiding the need to slow start to re-fill the pipe. The fast recovery phase is entered by a TCP sender after receiving an initial threshold (typically three) of duplicate acknowledgments and retransmitting the lost packet. In the fast recovery mode, the sender sets CWND to one-half of its current size. Duplicate acknowledgments are used to trigger subsequent transmissions. Consequently, the sender's usable window becomes the minimum of AWND and CWND + NDUP [4]. NDUP is maintained at zero until the number of duplicate acknowledgments reaches the initial threshold, and thereafter tracks the number of duplicate acknowledgments. After entering the fast recovery phase and retransmitting a single packet, the sender effectively waits until half a window of duplicate acknowledgments are received, and then sends a new segment for each additional duplicate acknowledgment that is received. Upon the receipt of an acknowledgment for a new segment, the sender exits the fast recovery mode by setting NDUP to zero.

An improvement over Reno, referred to as *New-Reno* in this paper, eliminates Reno's wait for

[4]NDUP is the number of duplicate acknowledgments received by the sender.

```
(1) After every new acknowledgment
        if (room under congestion and advertised windows)
                if (DELAY-TIMER set)
                        if (enough tokens in the bucket) send packet as marked
                        else send packet as unmarked
                        endif
                else
                        if (tokens available > packet size) send packet as marked
                        else set DELAY-TIMER
                        endif
                endif
        endif

(2) After every DELAY-TIMER expiry
        if (room under congestion and advertised windows)
                if (tokens available > packet size) send packet as marked
                else send packet as unmarked
                endif
        endif
```

Figure 7: Delayed send algorithm.

a retransmission timeout when multiple packets are lost from a window [9]. During fast recovery, when an acknowledgment acknowledges some, but not all, of the segments that were outstanding at the beginning of the fast recovery period, a Reno sender exits the fast recovery phase by deflating the usable window to CWND. In New-Reno, partial acknowledgments do not take TCP out of the fast recovery mode. Instead, partial acknowledgments received during fast recovery are treated as an indication that the segment immediately following the acknowledged segment in the sequence space has been lost and should be retransmitted. Thus, when multiple segments are lost from a single window, a New-Reno sender can recover without a retransmission timeout. It retransmits one lost packet each round-trip time until all the lost packets from the window have been retransmitted. A New-Reno sender remains in fast recovery until all outstanding segments when fast recovery was initiated have been acknowledged.

4.2 Adaptive Window Control

Each time 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. For these 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 mechanism. The key idea behind this modification is that for reserved connections CWND consists of two parts: a fixed part equal to the product of the reserved rate and the estimated round-trip time, and a variable part that tries to estimate the

```
(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
                endif
        endif

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

Figure 8: Timed send algorithm.

residual capacity on the link and share it equally with other active connections. Strictly speaking, the fixed part is not really fixed, but varies with the variation in the round-trip delay. Let us assume that the size of the fixed or reservation window is RWND. Hence, the size of the variable window is CWND−RWND. In the modified scheme, we treat 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 to 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 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. The control algorithm can be summarized in Figure 4.

With these changes in place, we repeated the experiments described in the last section. The total and compliant throughput seen by connection C_4 with and without the windowing changes are shown in Figure 5. We observe that enhancements to the windowing mechanisms significantly improves the throughput seen by connection C_4. However, it is still not quite close to the expected ideal. The compliant bandwidth is still less than the reserved bandwidth for the connection.

Figure 6 plots a five-second packet trace of connection C_4 with the changes in the windowing mechanism. The packet trace shows that the swings in the congestion window are much smaller than those in Figure 3. However, there are more packet losses over this five-second interval as compared to that over a comparable period in Figure 3, indicating that the operating point is closer to the saturation point of the network in this experiment. We also see a lower, but still significant rate of token loss (560 packets worth of tokens as opposed to 830 in Figure 3).

The steady rate of token loss is a result of the fundamental limitation of TCP's acknowledgment triggered transmissions. TCP relies on acknowledgments to trigger transmission of new segments. When there are significant gaps in the acknowledgment stream, the sender cannot send and the token bucket for the connection overflows. This phenomenon explains why even a greedy TCP source cannot consume all the tokens that are generated. In comparing the two packet traces, we see that the connection using the windowing changes sees less gaps in the acknowledgement stream than the connection without the changes. This is because the connection using the windowing changes is throttled less on a packet loss than its couterpart. The gaps in acknowledgments, also known as acknowledgment compression, is a well-known myth [24]. This may happen due to a number of reasons including congestion on the forward and/or reverse path as well as additional queueing delays and jitter caused by new connections coming online. Unfortunately, these gaps tend to be regenerative since the inability to send during these gaps leads to gaps in the acknowledement stream during the next round-trip. This results in a periodic and persistent loss of tokens as indicated in the packet trace. To alleviate this fundamental incompatibility between TCP's acknowledgment triggered transmissions and constraints of a rate-based reservation paradigm, we experimented with timed and delayed send mechanisms. In the next section, we explore them in greater detail.

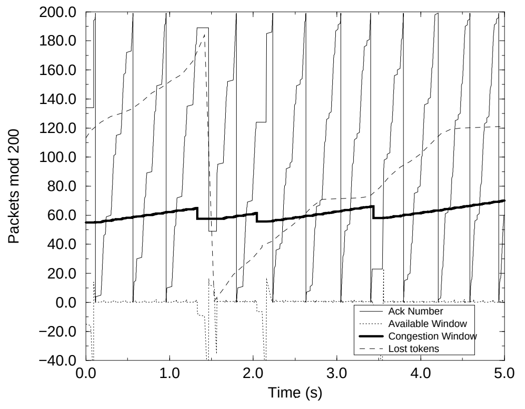
5 Timed and Delayed Sends

The results from the experiments in the previous section show how persistent gaps in the acknowledgments can cause token loss at the source and thus, lower than expected throughput. This is an artifact of TCP's acknowledgment triggered transmit mechanism. In this section we explore two different schemes, delayed and timed transmissions, to better adapt acknowledgment-based transmit triggers to the rate-based reservation paradigm. In the delayed mechanism a segment is held back for a random amount of time when there aren't enough tokens in the token bucket to transmit it as a marked segment. This, in effect, adds randomization to the data stream of the connection which can eliminate any persistent gaps in the acknowlegdements. 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. The second mechanism we examine involves the use of a periodic timer. In this scheme, we augment TCP's acknowledgement-triggered sends with a timer-triggered send mechanism. This timer-based triggering ensures that transmission opportunities are not lost while the connection is waiting for an acknowledgment.

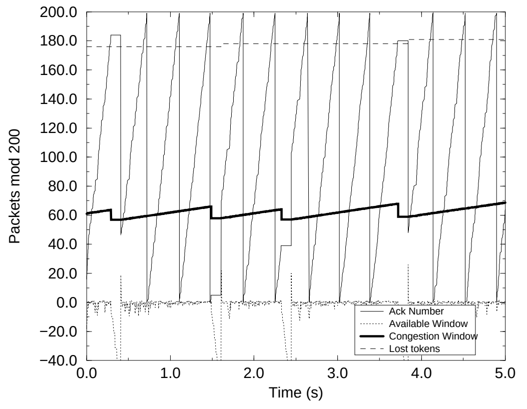
For the delayed send mechanism, we maintain at most one timer per connection. If a segment is ready to be transmitted and there aren't enough tokens in the token bucket, the sender sends this packet without delay and marks it as compliant. If there are not sufficient number of tokens in the bucket and the delayed send timer is not currently set, the transmission of the segment is delayed by a random amount of time by setting the delayed send timer and putting the connection to sleep. If an acknowledgment comes while the timer is active, the next packet is sent immediately, marked or unmarked. Finally, when a delayed send timer expires, the sender checks to see if there is enough room under the congestion window to send the packet. If there is room, the sender sends a packet regardless of whether or not there are tokens available. Figure 7 explains the algorithm more formally.

The delayed send algorithm adds randomization to the connection without directly sacrificing its ability to send compliant packets. The connection only delays its transmission when there aren't enough tokens in the token bucket to send the next packet as marked. The randomization in the data stream induces similar randomization into the returning acknowledgment stream. This reduces, but does not necessarily eliminate, the problem of persistent cycles of gaps in the acknowledgment stream.

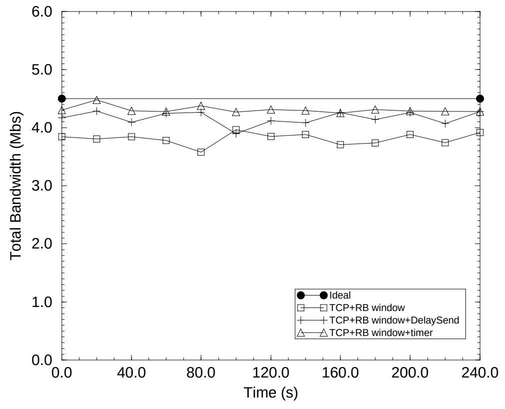
In the timed send mechanism, acknowledgment-based triggering of transmissions is augmented with timer-based triggering. Each reserved connection uses at most one timer; each can have potentially different timer intervals. Connections can also share a single timer depending on the overhead on the end-host. In the timed send scheme, the acknowledgment-clocked transmission algorithm is left unmodified. However, whenever a periodic timer expires, the connection examines



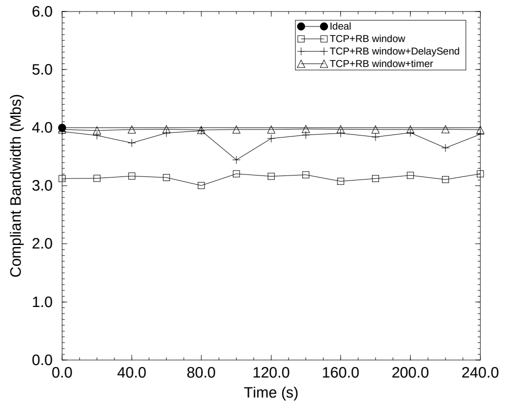
(a) Packet trace with delayed send.



(b) Packet trace with timed send.



(c) Total throughput.

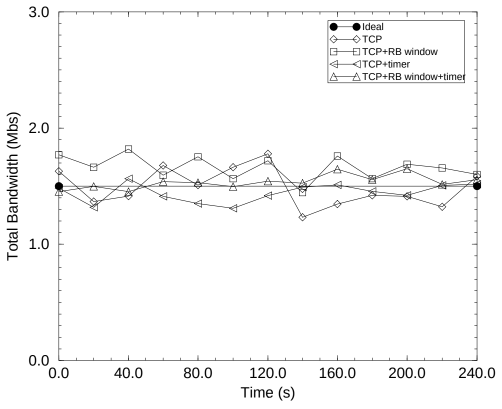


(d) Compliant throughput.

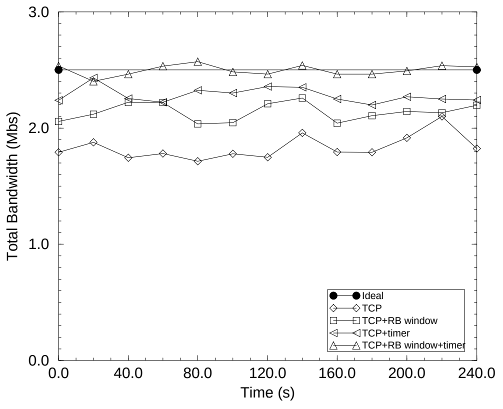
Figure 9: Packet trace and throughput of connection C_4 with windowing and timer modifications.

the tokens in the token bucket. If there are sufficient number of tokens 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 and the connection goes to sleep. Figure 8 presents the algorithm formally.

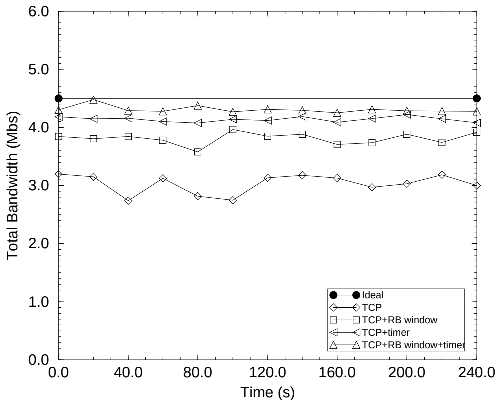
The intuition behind timed send is very simple. If there are enough tokens in the bucket, as per the contract with the network, the sender is eligible to inject new data in the network. Hence, we disregard the congestion window under such circumstances. Note that the connection still adheres to the advertised window constraint to avoid overflowing the receivers buffers. 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.



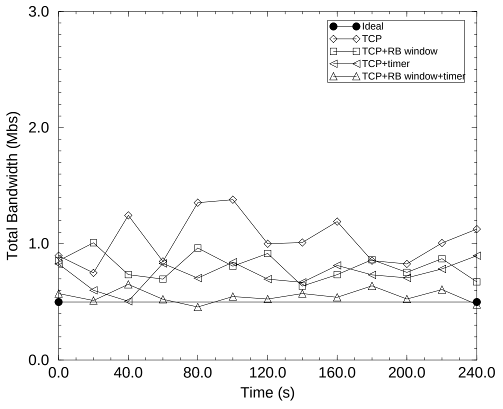
(a) Total bandwidth seen by C_1



(b) Total bandwidth seen by C_2



(c) Throughput seen by C_4.



(d) Throughput seen by BE connection.

Figure 10: Relative performance of TCP schemes.

5.1 Performance Impact

Figure 9 shows the packet traces of C_4 for both the delayed send and timed send algorithms. The setup for both of these experiments is the same as the experiments described in the last section. For the delayed send experiment, the timeout interval is a uniformly distributed random variable in the range of 0–50ms. For the timed send experiment, the timer interval is set to 20ms.

While the delayed send scheme allows us to better synchronize the transmissions of TCP packets with token generation process, the trace shows that tokens are still lost. In particular, when packet losses occur, as indicated by the reduction of the congestion window, gaps in the acknowledgments appear. These gaps, as shown in the acknowledgment number plot, cause an increased number of tokens to be lost. The gaps and subsequent token loss continues for a period of time before the delayed send mechanism is able to correct them. In contrast, the timed send scheme, shows almost no lost tokens. Tokens are only lost when packet losses cause the advertised window to fill up.

Figures 9(c) and 9(d) show the throughput seen by connection C_4 when the modified windowing

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).

mechanism is used in conjunction with delayed and timed sends. The graphs show the total throughput and compliant throughput seen by the high bandwidth connection, C_4. As shown in the Figure 9(c), the TCP senders using delayed and timed transmissions get closer to the ideal bandwidth than those using just the windowing changes. Both timer schemes get significantly better compliant throughput as shown in Figure 9(d).

Figure 10 shows the throughput for C_1, C_2, C_4, and best-effort connections using normal TCP, TCP with windowing changes alone, and TCP with both windowing and timed sends. We also plot the performance of TCP with just the timed sends in order to see if we can obtain ideal performance without the windowing changes. As shown in the figure, TCP senders using both the enhanced windowing and timed send mechanism perform the best in all cases. TCP senders using just the timed send perform fairly well, but, since their windowing is not sensitive to the reserved rate of the sender, high bandwidth connections receive slightly less than their respective ideal throughput. This allows the best-effort connections to grab more than their ideal fair share of the bandwidth. As explained earlier, TCP with just windowing enhancements does not perform well since a large number of tokens are lost, causing the reserved connections to lose more than their share of packets.

5.2 Timer Overheads

In our description of delayed and timed transmission algorithms 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. Also, note that the timers expire only occasionally, during periods when the gaps in the acknowledgments become large enough to keep a connection in the idle state for a long time. Hence, in our judgment, the proposed timed and delayed send mechanisms do not add significant overhead on the system. Nonetheless, 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 has advanced considerably since the first design and implementation of TCP. Processors today are much faster, and consequently the overheads of handling timer interrupts are much lower. In the following we present 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 from the results presented in Table 1 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

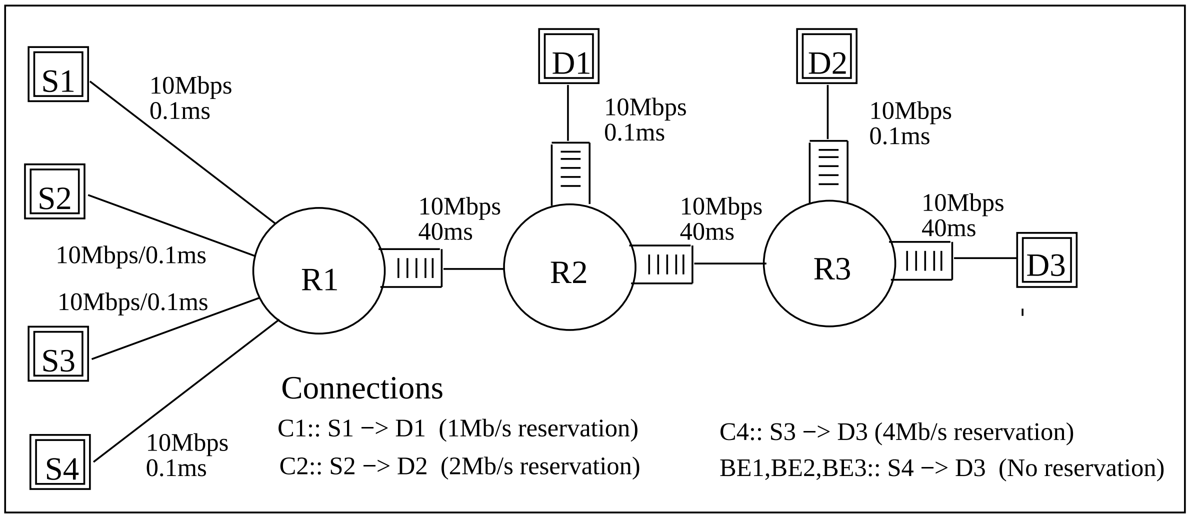

S1
S2
S3
S4
D1
D2
D3
R1
R2
R3
10Mbps 0.1ms
10Mbps/0.1ms
10Mbps/0.1ms
10Mbps 0.1ms
10Mbps 40ms
10Mbps 0.1ms
10Mbps 40ms
10Mbps 0.1ms
10Mbps 40ms
Connections
C1:: S1 -> D1 (1Mb/s reservation)
C2:: S2 -> D2 (2Mb/s reservation)
C4:: S3 -> D3 (4Mb/s reservation)
BE1,BE2,BE3:: S4 -> D3 (No reservation)

Figure 11: Network configuration of multi-hop experiments.

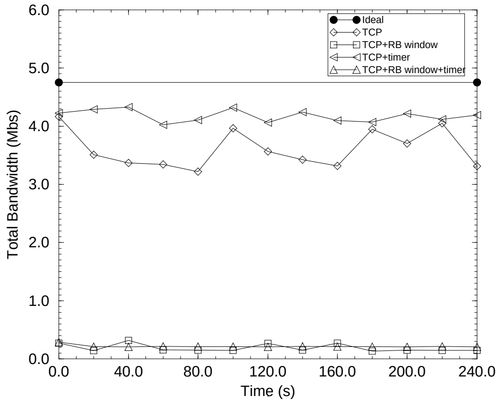
such an optimization is not deemed necessary.

6 Recovering from Multiple Losses

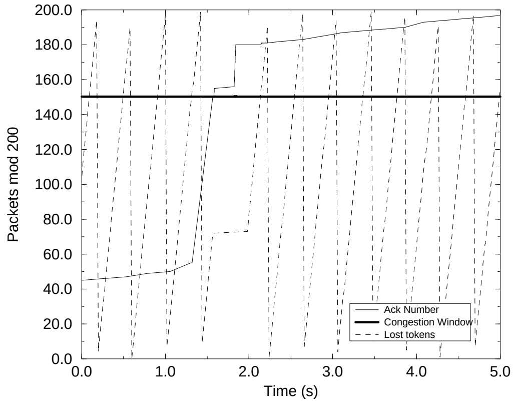
The experiments reported in previous sections show that the enhancements to the windowing mechanisms and timed transmissions perform well on a simple single-hop network. We now examine the impact and effectiveness of these enhancements in a larger multi-hop network. One of the aspects of particular importance in a large network is the delay-bandwidth product of the network, and consequently, the number of in-flight segments for a particular TCP connection. It has been observed [13] that the goodput of TCP connections suffers when there are multiple segment losses within the window. To investigate the impact of multiple packet losses on the proposed schemes, we performed a series of experiments on a multi-hop network. While most experiments produced results similar to those presented in the last section, some network configurations gave different results.

Figure 11 shows one such network configuration, where the links are labelled with their respective bandwidth capacities and transmission delays. As before, we have considered six TCP connections in this experiment. Connection C_1 traverses from $S1$ to $D1$ and has a reservation of 1Mb/s. Connection C_2 has a reservation of 2Mb/s and traverses from $S2$ to $D2$. Similarly, connection C_4 has a reservation of 4Mb/s and traverses from $S3$ to $D3$. All three best-effort connections traverse from $S4$ to $D3$.

Figure 12(a) shows the throughput seen by connection C_4. In this case, both schemes employing the modified window mechanisms perform poorly, achieving a throughput less than 0.5Mb/s. Figure 12(b) shows the packet trace of the TCP sender with windowing and timer changes. The first part of the plot shows the connection recovering from a large burst of losses. Since the underlying TCP takes one full round-trip time to send each lost packet, the rate at which the sender transmits segments is equivalent to one packet per round-trip time (as seen by the near horizontal plot for the



(a) Throughput seen by C_4.



(b) Packet trace of C_4 with windowing and timer changes.

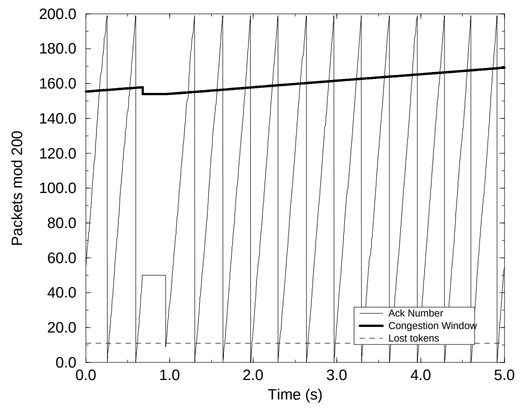
Figure 12: Throughput and packet trace of C_4 (multihop).

acknowledgment number). That is, TCP degenerates into a stop-and-wait protocol. The connection is thus frozen during this period by the congestion window (in the TCP with windowing changes scheme) and by the receiver's window (in the TCP with windowing and timer changes). In this situation, tokens are lost at a rate equal to ReservedRate − (SegmentSize/RoundTripTime). This period ends when the acknowledgement for the final lost packet is received, which opens up the window. At this point, a large burst of packets is sent on the network, of which some packets are compliant (as seen by the flattening of the lost tokens plot), but most are not. This large burst drives up the queue sizes of intermediate routers and places a large number of noncompliant (unmarked) packets in these queues. Once again this inflicts a large number of losses to the connection, which slows down the transmit process for a period equal to the product of burst losses and round-trip time. This cycle of burst and idle periods results in poor goodput for the connection. Note that this phenomenon also occurs in classical TCP [3,12], but since TCP halves the window on every loss, the cycle only repeats a small number of times before the window shrinks down sufficiently to limit the size of the bursts and the number of burst losses.

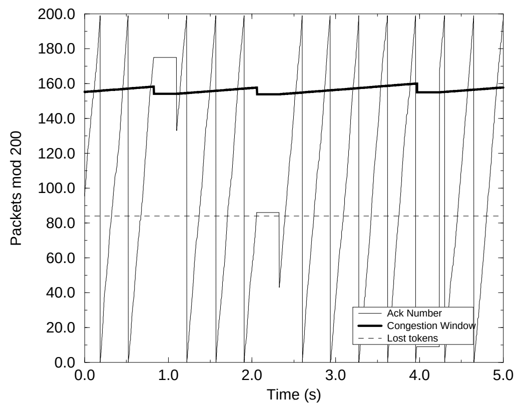
To alleviate this problem, we examine two possible solutions: using TCP with selective acknowledgments (TCP SACK) and limiting bursts during the fast recovery phase of Reno TCP variants. By using TCP SACK, the sender can recover much more quickly from multiple burst losses especially when the round-trip times are large. On the other hand, limiting bursts during the fast recovery phase of Reno TCP variants can prevent burst losses from occurring in the first place.

Figure 13 shows the packet traces using both the TCP SACK as well as the Reno variant with burst control. We control bursty transmits by using a *MaxBurst* parameter at the sender that determines the maximum number of segments transmitted in response to a single acknowledgment. By setting this parameter to 2, we effectively force a slow start after recovering from a loss. As the traces show, both schemes perform comparably well with a low number of lost tokens.

Figure 14 shows the total bandwidth received for each connection using the various Reno schemes. For the schemes incorporating the windowing changes, the MaxBurst parameter is set to 2. As the figure shows, these schemes approach the ideal performance. Connection C_1 receives



(a) New-Reno with MaxBurst set to 2.



(b) SACK.

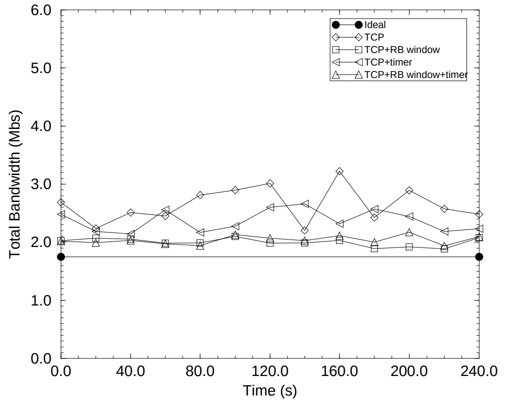
Figure 13: Packet traces of C_4 with windowing and timer changes.

slightly more than its fair share due to its relatively smaller round-trip time. In contrast, the best-effort connections get less than their fair share since their round-trip times are the largest. This is consistent with observations by other researchers which have shown that connections with longer round-trip times are slower in opening up their congestion windows [12] and thus end up getting less bandwidth. Interestingly, connection C_4 overcomes the negative impact of the long round-trip time with its timer triggered transmissions.

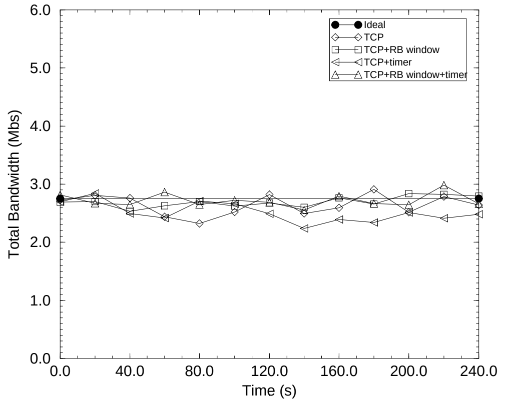
7 Conclusions

In this paper we have considered transport level support for a class of applications that require (a) reliable packet delivery, and (b) a guaranteed minimum service rate for the connection. Although TCP provides reliable packet delivery, it is unable to provide applications with a guaranteed minimum service in today's best-effort Internet. However, with the advent of new classes of service in the Internet, we believe that TCP may be employed to support this class of applications. This new environment is not entirely in tune with the original philosophy of TCP, which was designed with the premise that network capacity should be shared equally amongst competing connections.

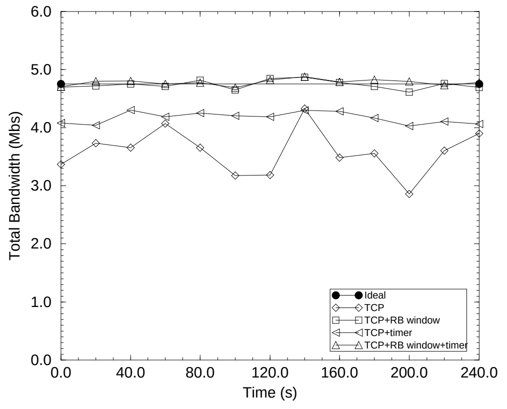
We have studied the behavior of TCP with the controlled-load using simulation. We anticipated an end-to-end throughput for the connection that included the bandwidth reserved for the connection and a fair share of the residual bandwidth in the bottleneck link. However, we observed that the congestion control mechanisms in TCP interfere with the rate based reservation mechanisms defined for the controlled-load service. The resulting end-to-end throughput delivered across the TCP connection is, consequently, less that the reserved rate for the connection. To rectify these shortcomings, we have proposed and evaluated several enhancements to the congestion and flow control algorithms used in TCP. Among these enhancements are a windowing mechanism that is cognizant of the transmission rate reserved for the session. This is complemented by delayed and timed transmissions mechanisms that supplement the basic ACK-clocked transmissions in TCP. We have shown that, with these enhancements in place, TCP is able to operate effectively for sessions with reserved bandwidth and provide the desired end-to-end behavior. We have also shown that our



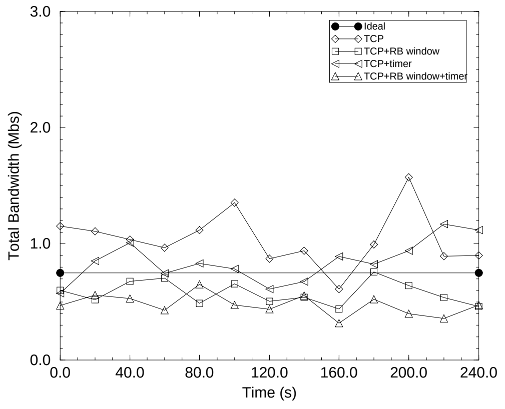
(a) Throughput seen by C_1.



(b) Throughput seen by C_2.



(c) Throughput seen by C_4.



(d) Throuput seen by a best-effort connection.

Figure 14: Relative performance of different TCP schemes.

enhancements can be combined easily with other proposed mechanisms such as SACK that enhance the error control algorithms of TCP.

This work can be extended in several ways. We are in the process of studying the effectiveness of the proposed enhancements in a heterogeneous network environment where the capabilities of different routers are different. Since upgrades to the Internet infrastructure is expected to be evolutionary, it is very likely that routers with enhancements for class-based service will co-exists with legacy best-effort routers. Hence, it is important to study, and possibly enhance, the proposed changes to TCP in an environment that reflects the evolution of the Internet from a best-effort network to a network with different types and classes of services.

Another important study would be to evaluate the impact of different scheduling mechanisms on the control algorithms used by TCP. In this paper we only consider routers that implement a variant of early random detection and discard algorithm. Repeating the same experiments with routers employing class based queuing or weighted fair queueing will be an interesting exercise.

We are also considering the possibility of using a rate-adaptive flow control algorithm in the context of RTP sessions running over UDP. Many of today's multimedia applications use uncontrolled UDP streaming to send data across the Internet with very high losses. This is not only wasteful, but threatens the very future of the Internet. It may be useful to implement a socially acceptable flow control mechanism in RTP/UDP to share the network capacity among active sessions in a manner commensurate with reservations associated with different sessions.

References

[1] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne. Real-Time Communication in Packet-Switched Networks. *Proceedings of IEEE*, 82(1), January 1994.

[2] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification. *Internet Draft draft-ietf-rsvp-spec-13.txt*, May 1996. ISI/PARC/USC.

[3] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of ACM SIGCOMM*, pages 24–35, October 1994.

[4] A. Demers, S. Keshav, and S. Shenkar. Anslysis and Simulation of Fair Queuing Algorithm. In *Proceedings of SIGCOMM*, 1989.

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

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

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

[8] 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.

[9] 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.

[10] IBM Corporation-AlphaWorks: http://www.alphaworks.ibm.com/. Bamba from AlphaWorks, 1996.

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

[12] V. Jacobson and R. Braden. TCP Extensions for Long-Delay Paths. RFC 1072, October 1988.

[13] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018, 1996.

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

[15] PointCast, Inc.: http://www.pointcast.com/. PointCast Inc - Front Door., 1996.

[16] Progressive Networks: http://www.realaudio.com/. Progressive Networks, The Home of RealAudio, 1996.

[17] K. K. Ramakrishan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transaction on Computer Systems*, 8(2):158–181, May 1990. **Review:** *Computing Reviews*, December 1990.

[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] S. Shenker, C. Partridge, and R. Guerin. Specification of Guaranteed Quality of Service. *Internet Draft draft-ietf-intserv-guaranteed-svc-05.txt*, June 1996. Xerox/BBN/IBM.

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

[22] VDOnet Corp.: http://www.vdo.net/. VDOnet - Real-Time Video & Audio over the Internet, 1996.

[23] J. Wroclawski. Specification of Controlled-Load Network Element Service. *Internet Draft draft-ietf-intserv-ctrl-load-svc-03.txt*, June 1996. MIT.

[24] 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.