End-to-End TCP Congestion Control over ATM Networks

Mohit Aron Peter Druschel

TR97-273
Department of Computer Science
Rice University

Abstract

It is well documented that the effective throughput of TCP can suffer on plain ATM networks. Several research efforts have aimed at developing additions to ATM networks like Early Packet Discard that avoid TCP throughput degradation. This paper instead investigates improvements to TCP that allow it to perform well on ATM networks without switch-level enhancements, thus avoiding additional complexity and loss of layer transparency in ATM switches.

We present an enhanced version of TCP Vegas and show that it performs nearly as well on plain ATM networks as on packet-oriented networks. Moreover, like the original TCP Vegas, our version achieves a significant increase in throughput over the BSD 4.4-Lite TCP. We also present an analysis of TCP dynamics over ATM networks, and a simulation based study of the various enhancements in our TCP implementation and their impact on TCP performance over ATM and packet-oriented networks.

1 Introduction

ATM (Asynchronous Transfer Mode) technology is expected to gain widespread use both in wide-area networks (including B-ISDN) and in high-speed local area networks (LANs). The performance of the TCP/IP protocol suite over such networks is of great importance, as it is widely used in the Internet and in private internetworks.

Numerous studies have shown that the effective throughput of conventional implementations of TCP suffers when run over plain ATM networks, particularly under conditions of network congestion [22, 10][1]. Partial Packet Discard and Early Packet Discard [22] are techniques that improve the performance of TCP over ATM networks at the cost of increased complexity and the loss of AAL layer transparency in the ATM switches. The ABR (available bit rate) [5] service for ATM has been proposed to address the same problem; its exact form and its impact on TCP performance are still unproven. Both proposals share the common approach of extending ATM networks to enhance TCP throughput.

[1] By "plain" ATM network we refer here to an ATM network with best effort (unspecified bit rate (UBR)) service, and no AAL-specific switch-level enhancements.

This paper takes the approach of considering modifications to TCP, with the goal of achieving good throughput on plain ATM networks without any switch level enhancements. We propose the *adaptive fast retransmit* (AFR) mechanism, which modifies the fast retransmit algorithm of TCP to make it deal effectively with consecutive packet losses frequenting the congestion conditions on ATM (as shown by our analysis). We also show congestion avoidance as a powerful mechanism in improving TCP performance over ATM. Based on our proposed changes, we present an enhanced version of TCP-Vegas [6], a TCP implementation originally designed for packet-oriented networks. Extensive simulations show that our version of TCP-Vegas performs on plain ATM networks nearly as well as or better than both conventional TCP and TCP-Vegas perform on packet-oriented networks. These results hold across a variety of network topologies and switch buffer sizes, and for both long and short TCP transfers.

The remainder of this paper is organized as follows. We begin by giving a brief introduction to TCP and a discussion of the effects of ATM and packet networks on TCP dynamics. Section 3 describes our simulation environment and in Section 4 we investigate the performance degradation of TCP-Lite on ATM. Section 5 presents a detailed simulation-based study of the various enhancements in our version of TCP-Vegas over TCP-Lite, and their impact on TCP performance in ATM and packet-oriented networks. In Section 6 we discuss other related-work and we conclude by summarizing our results in Section 7.

2 Dynamics of TCP over ATM

Since the seminal work of Van Jacobson in [16] the BSD releases of TCP have undergone several important changes from 4.3BSD Tahoe in 1988, to 4.3BSD Reno (TCP Reno) in 1990 and finally 4.4BSD-Lite (TCP Lite) [25] in 1994. The congestion control techniques incorporated in these implementations include the slow-start mechanism and the fast retransmit and the fast recovery algorithms [17, 24]. TCP-Lite[2] is an extension of TCP-Reno and provides support for long fat pipes (high bandwidth-delay paths) amongst other improvements. However, the congestion control algorithms used in TCP Lite are essentially the same as those in TCP Reno.

TCP-Vegas is an implementation of TCP that is described in [6, 8]. TCP-Vegas tries to eliminate the self-induced losses in TCP-Reno. It modifies the slow-start mechanism of TCP-Reno and provides new retransmission and congestion avoidance mechanisms. The performance of TCP-Vegas against TCP-Reno was also evaluated in [1].

The rest of this section discusses key differences between ATM and packet networks, how these differences interact with TCP dynamics, and the manner in which they

[2] Our version of TCP-Lite corresponds to Lite.4 proposed in [7]. This version fixes many bugs in the original code.

affect TCP performance. Many of these observation have been made elsewhere in the literature; we recount them here for completeness.

Discard Unit and Retransmission Unit: A key difference between ATM and packet networks is that in ATM the switches' discard unit (a cell) typically differs from the end host's retransmission unit (e.g., a TCP segment). When TCP/IP is run over ATM, a TCP segment is fragmented at the AAL5 layer into 48 byte ATM cells. A congested ATM switch drops data at the granularity of ATM cells. When even a single cell from a TCP segment is dropped, the segment has to be retransmitted in its entirety. In a packet network, on the other hand, the retransmission unit and the discard unit are identical. This mismatch between ATM and TCP/IP can lead to several undesirable effects on TCP performance:

Bandwidth and resource wastage: The remaining cells from partially dropped segments occupy network resources and waste link bandwidth, only to be discarded at the destination [22]. This is likely to cause more congestion, since dropping n cells leads to the eventual retransmission of m cells, for $n \leq m \leq 2 * n * MTU/48$ (in the worst case, each cell drop could corrupt two segments and each segment has MTU/48 cells).

Multiple segment loss: An ATM switch does not know about segment boundaries. Hence in general, dropping a segments's worth of data at an ATM switch might lead to the loss of cells from several TCP segments. In particular, two consecutive segments from a TCP connection are often lost when a switch drops only one segment's worth of data. [20].

Interleaving of Cells: The cells from different TCP connections (each using a different virtual circuit) may be interleaved in an ATM network. Hence dropping one segment's worth of data at an ATM switch can lead to the loss of segments from several TCP connections. In a packet network, on the other hand, dropping a segment's worth of data results in the loss of just one segment and hence only one connection is affected.

Given these phenomena, one can expect that under otherwise identical conditions, a TCP connection over an ATM network is likely to loose more segments, and it is more likely to suffer more consecutive segment losses than a TCP connection run over packet switched network.

There are some additional characteristics of ATM that have an impact on TCP performance. These characteristics are not specific to ATM; instead, they are related to the fact that ATM networks are typically high-bandwidth networks.

High Bandwidth: ATM is considered a gigabit technology and ATM networks are typically high bandwidth networks. TCP implementations use coarse-grained clocks to estimate round-trip delay (RTT) and schedule retransmission timeouts (RTOs). Consequently, RTOs tend to exceed the actual RTT by a large amount. The throughput loss resulting from the associated period of inactivity preceding a timeout is proportional to the bandwidth of the network.

Large Bandwidth-Delay Product: Wide-area, high-bandwidth ATM networks can have high bandwidth-delay products. During its initial slow-start, TCP doubles its congestion window every RTT until losses are incurred or the congestion window reaches the receiver's advertised window. When losses occur, it can be expected that a substantial fraction of the current congestion window is lost. Since the size of that congestion window is related to the network's bandwidth-delay product, it follows that the amount of data lost—and the degree of congestion caused in the network—during the initial slow-start increases with the network's bandwidth-delay product.

Large MTU: In typical use, IP over ATM uses a packet size of 9180 bytes compared to about 1500 bytes for ethernet networks. The MTU determines TCP's maximal segment size, which in turn determines the length of traffic bursts generated by a TCP sender (TCP can send up to 2 maximal sized segments per receiver acknowledgment during slow-start; under certain conditions it can even send more [7]). A larger MTU therefore requires more buffering at the switches to absorb traffic bursts. In addition, a larger MTU may aggravate the problem of ACK compression [27].

3 Network Configuration

This section describes the simulation environment that was used to produce the results presented in the following sections. The simulations were done using the *x*-sim network simulator [9], which is based on the *x*-kernel [15]. *x*-sim is an execution-driven network simulator, where the actions of network protocols are simulated by executing its actual protocol implementation code, rather than an abstract behavioral model of the protocol. Host and router computation is assumed to have zero overhead. The simulator clock

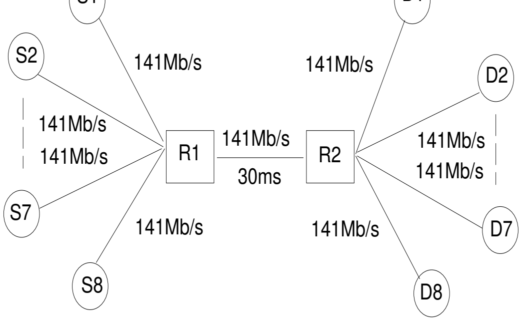

Figure 1 Topology 1

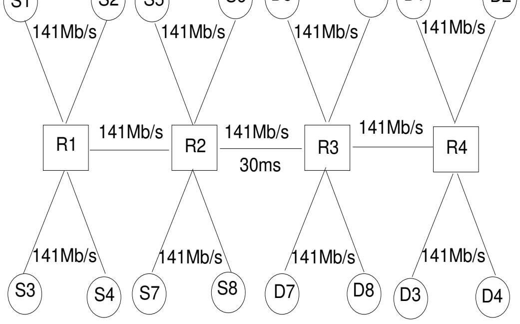

Figure 2 Topology 2

has a granularity of $1\mu s$.

All the simulations use a TCP window size of 512KB. The default MTU for IP over ATM is 9180 bytes [3]. However, the TCP segment size in all the simulations presented in this paper is 9168 bytes. This size was chosen to be an exact multiple of the ATM payload (48 bytes). This minimizes the padding done at the AAL5 layer when a segment is broken into cells, thus enabling a more effective comparison between packet-TCP and TCP over ATM. Padding only occurs when the packets have a size less than 48 bytes (e.g. ACK packets) and possibly for the last segment sent when the total amount of data sent is not an integral multiple of the MTU. The TCP segment size over packet networks used in the simulations is also 9168 bytes so as to make a fair comparison of packet-TCP with TCP over ATM. The switches used in the simulations are FCFS switches which use output buffering and the drop-tail discipline for dropping packets/cells. In simulations involving TCP over ATM, each TCP connection uses a separate ATM virtual circuit (VC).

Two network topologies were used in our simulations. These topologies were used both for ATM as well as packet network. In case of the packet network, the interior nodes are IP routers, while for ATM they are ATM switches. The first and simpler network topology used is shown in Figure 1. Eight senders (S1, S2, ... S8) use TCP to send data to eight destinations (D1, D2, ... D8) across a bottleneck link. The bottleneck link connects two switches R1 and R2 where R1 is the bottleneck switch. The link bandwidth was chosen to be 141.33 Mbps so as to make the transmission of an ATM cell (53 bytes) an integral number of microseconds ($3\mu s$). The round-trip propagation delay is 60ms. This topology resembles the ones used in other studies on TCP dynamics [22, 21]. The second topology is shown in Figure 2. This has two more switches (R3 and R4) than the first topology. Four senders each are connected to switches R1 and R2. Four destinations each are connected to R3 and R4. This topology differs from the former in providing two bottleneck points (switches R1 and R2). It more closely resembles a real internet where different hosts on a LAN (local area network) transfer data to different destinations across a WAN (wide area network).

When all senders start simultaneously sending data to the respective destinations, we noticed segregation effects [13, 22] for low switch buffer sizes. This caused some of the connections to get most of the bandwidth causing the other connections to starve. We staggered the startup times of the senders by 10ms to allow their congestion windows to grow for some time before experiencing congestion. This eliminates most of the segregation effects and all results reported in this paper have a fairness of at least 0.79 for topology 1 and 0.70 for topology 2, as measured by Jain's Fairness index [18]. Most of the results reported have a fairness of above 0.95.

4 TCP-Lite over ATM

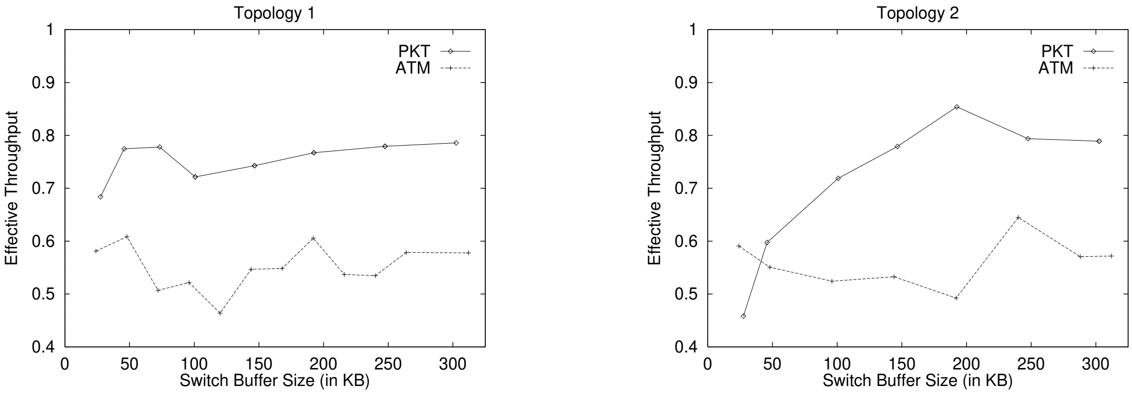

Figure 3 TCP-Lite over ATM and packet network

In this section we compare the performance of TCP-Lite over ATM and packet networks. All senders were made to transfer 60MB of data to the corresponding destinations and the simulation time was limited to 10s (enough to transfer about 175MB of data across the bottleneck link). The simulation results reported are medians from 5 runs. The results showed very low variance with different seeds for the simulator. Figure 3 compares the effective throughput of a typical sender as a function of switch buffer size on both network topologies. The effective throughput is shown as a ratio of the total amount of useful data that was transferred, to the maximum amount of useful data that could have been transferred over the bottleneck link for the time of the simulation. For the ATM networks, the x-axis refers to the *effective* switch buffer size. The effective switch buffer size for ATM does not include the space needed for the cell headers. It is clear from the plots that the effective throughput of TCP-Lite over ATM can be 10-30% lower than that over packet networks.

Earlier studies have suggested that the primary cause of TCP throughput loss over ATM are the transmission of useless cells from partially dropped segments. Our results show that this effect can only account for a fraction of the observed throughput loss.

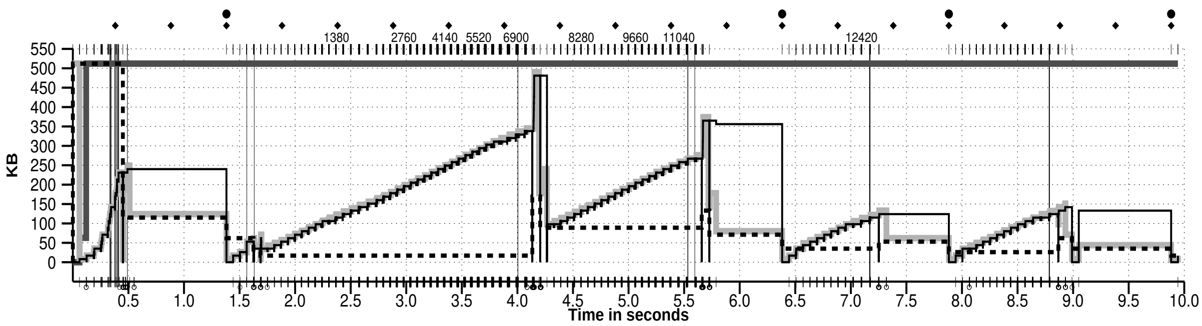

Figure 4 TCP-Lite over ATM

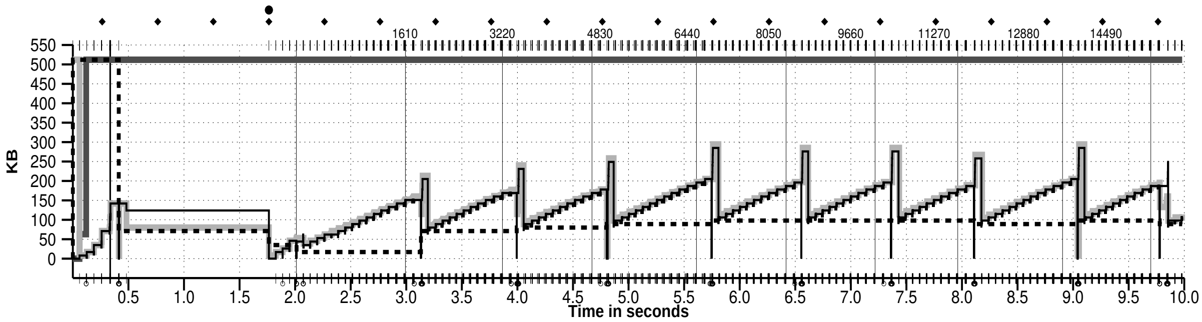

Figure 5 TCP-Lite over packet network

Consider the following. In the simulations shown in Figure 3, the average amount of retransmitted data never exceeds 560KB. This gives an upper bound on the number of segments that were lost due to cell loss. In the worst case, only one cell from each of these segments is dropped, in which case almost all of the 560KB will occupy link bandwidth and waste resources. As the simulation time was limited to 10s, the bandwidth occupied by these dead cells does not exceed 56KB/s per sender. However, a bandwidth loss of 56KB/s only accounts for about 3% loss in throughput of each sender, whereas we observe a 10-30% loss in throughput. In the following sections, we will analyze all of the causes of TCP throughput loss over ATM.

Figure 4 shows the window traces of a typical TCP-Lite transfer over ATM. Figure 5 shows a corresponding trace over the packet network. The traces[3] depict a transfer with the standard TCP coarse-grained timers.

On ATM, a large number of retransmission timeouts (near 1.5, 6.5, 8 and 10 seconds) can be seen as compared to the non-ATM network. These timeouts, and the long periods of inactivity associated with them, is the primary cause of the observed throughput loss. Moreover, the effective throughput for ATM is very sensitive to small variations in switch buffer size. This also results from the large number of retransmission timeouts and the coarse-grained clock used for scheduling these timeouts. A change in switch buffer size can affect traffic conditions so as to cause longer or shorter waits for retransmission timeouts, hence affecting throughput. It is also to be noted that congestion mostly causes single segment loss on the packet network, while multiple segment drops tend to occur during periods of congestion on the ATM network. These facts will be analyzed in more detail in the following sections.

[3] A description of the traces shown in these figures is given in Appendix A. [7] gives a more detailed description of the same.

5 Analysis of TCP Enhancements

In this section, we present and discuss the results of a detailed simulation study to assess the performance of various TCP enhancements over ATM. We propose a new technique called "adaptive fast retransmit" (AFR) and show that a version of TCP-Vegas, augmented with this technique, performs on plain ATM networks almost equally well as on packet networks for modest to large switch buffer sizes. Like the original TCP-Vegas, our version also has a throughput advantage on packet networks over TCP-Lite.

Figure 6 shows the effective throughput of our enhanced version of Vegas (Vegas-AFR) as a function of switch buffer size for ATM and packet networks, and on the two topologies shown in Figures 1 and 2. The results clearly show that Vegas-AFR performs almost equally well on ATM and on packet network across a wide range of switch buffer sizes, and on both topologies. In a later section we will show that Vegas-AFR also performs better over ATM for short transfer sizes. Performance with short transfer sizes is important for HTTP traffic, which accounts for a larger fraction of Internet traffic.

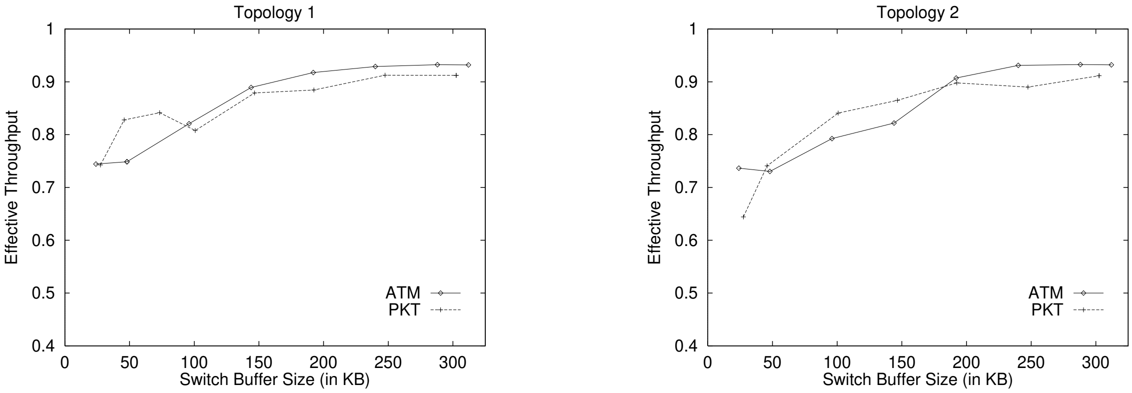

Figure 6 Vegas-AFR over ATM and Packet Network

In the remainder of this section, we present a detailed study of all enhancements over TCP Lite that are embodied in Vegas-AFR. In particular, we discuss how these techniques address the issues raised by ATM networks and quantify their impact on TCP throughput over ATM and packet networks.

The original TCP Vegas implementation, as distributed by the University of Arizona, incorporates three distinct enhancements[4]. It features a modified slow-start procedure (Vegas-SS) that allows a doubling of the congestion window only every other RTT. Vegas-SS makes a decision to move into the congestion avoidance state based on a comparison of the actual sending rate and a predicted sending rate.

[4]The Vegas distribution also contains a bug that prevented the congestion window from increasing beyond 64K on receiving duplicate ACKs while doing fast recovery. All simulations presented in this paper had this bug fixed.

Secondly, TCP-Vegas incorporates a new retransmission mechanism (Vegas-RM) that detects packet losses earlier than Reno and Lite. On receiving the first and second non-duplicate ACK after a retransmission, it also retransmits any segment that hasn't been acknowledged for a time larger than the timeout period. This is likely to catch any other segments that are lost prior to the first retransmission without having to wait for three duplicate ACKs again. Incidentally, this mechanism also catches the loss of consecutive segments in ATM networks. Finally, TCP-Vegas does not decrease its congestion window multiple times for losses that occur in the same RTT, and it decreases it congestion window by 3/4 and not by 1/2 as in TCP-Lite and Reno.

Thirdly, TCP Vegas uses a congestion avoidance procedure (Vegas-CA) that is proactive, and tries to detect the incipient stages of congestion before losses occur. It does so by additively increasing and decreasing its congestion window according to the difference between the actual sending rate and the predicted sending rate. This is in sharp contrast to TCP-Lite and Reno, which periodically create losses to probe the available capacity of the network.

Our own modification of TCP enhances the fast retransmit algorithm and is called adaptive fast retransmit (AFR). It observes the frequency of consecutive segment losses. When consecutive losses occur, indicating that an ATM network is used to support the connection, TCP enters a state where upon detection of a segment loss, it immediately retransmits the lost segment *and its successor*. All of the TCP enhancements considered in this paper are discussed in detail in the following subsections.

Observe that Vegas-RM and AFR deal with the situation when congestion occurs and segments are lost. In particular, they aim at avoiding timeouts and stalls in the transmission pipeline as a result of segment losses. Vegas-SS and Vegas-CA, on the other hand, aim at avoiding congestion in the first place. Thus, they reduce the number of situations where losses occur, and they reduce the amount of data lost in situations when losses cannot be avoided.

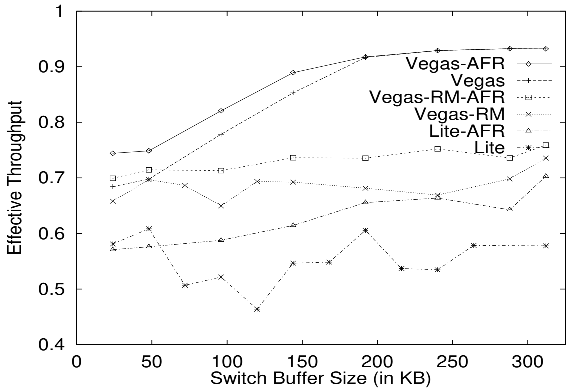

Figure 7 TCP over ATM

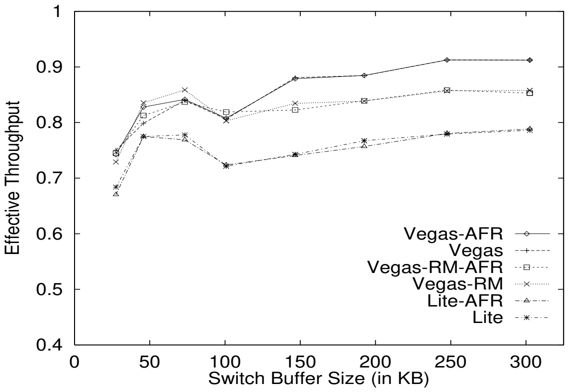

Figure 8 PKT-TCP

Figures 7 and 8 show the throughput results obtained in simulations involving various versions of TCP, both on ATM and packet networks. The network topology used is the one depicted in Figure 1. The plots corresponding to "Lite" and "Vegas-AFR" have already been alluded to earlier. Observe that the effective throughput achieved with Vegas-AFR surpasses or matches that of all other versions of TCP on ATM networks, across nearly all switch buffer sizes. Moreover, Vegas-AFR also matches the throughput of Vegas on packet networks, which in turn achieves between 5-15% higher throughput than TCP Lite on packet networks.

The remaining results shown in Figures 7 and 8 are discussed in the following subsections, along with a detailed discussion of the techniques that were used in the corresponding TCP versions, and their interaction with ATM networks. The purpose of this study is to analyze the function of the various enhancements embodied in Vegas-AFR, their relative importance, and their individual and combined impact on the performance of TCP over ATM and packet networks.

5.1 Handling Multiple Segment Losses

We begin with the issue of consecutive TCP segment losses. Consecutive losses commonly arise from congestion conditions in plain ATM networks, and they have a profound impact on the performance of TCP-Lite. This section motivates, presents and evaluates the adaptive fast retransmit (AFR) mechanism, a technique that effectively deals with consecutive packet losses.

As mentioned earlier, cell losses in plain ATM switches may span segment boundaries, resulting in the loss of two TCP segments as opposed to one in a packet network [20]. While TCP-Lite can effectively recover from the loss of a single segment in one RTT using its fast retransmit mechanism [24], it is not well equipped to handle the loss of two or more segments in a single RTT [14, 12]. First, the fast retransmit mechanism can only detect lost segments if the congestion window is large enough to supply duplicate ACKs for each lost segment. If the congestion window is too small, a costly timeout is necessary to recover from the losses. A second reason is that Lite (and Reno) halve the congestion window for each lost segment detected by duplicate ACKs. As a result, after the fast retransmission of the 2nd lost segment, no further segments can be sent until an ACK for the lost segments arrives (also noted in [11]). This causes a stall in TCP's transmission pipeline, temporarily underutilizing the available network capacity.

Finally, consider the case where a timeout occurs after some of the losses were detected by fast retransmit, causing the congestion window to be halved multiple times. As a result of the multiple congestion window reductions, TCP will terminate the slow-start following the timeout potentially too early, preventing it from quickly reaching the congestion window size necessary to fully utilize the network.

Figure 4 illustrates the effect of losses in an ATM network on TCP-Lite's windows. At around 4.2s, two segments were lost (only one vertical line can be seen as the segments were transmitted within a few microseconds of each other). The congestion window however was large enough to recover from both the losses. However, as a result of the recovery, the congestion window is halved twice, once for each segment that was lost. Near 5.6s, 3 segments were lost, which resulted in a retransmission timeout. In this case, the congestion window was halved twice when the first two losses were detected. Then Lite's slow-start threshold (ssthresh) was set to half of this value. As a result, slow-start ends at 1/8th the value of the congestion window when losses were first detected. The timeout near 7.9s was caused by two consecutive segment losses near times 7.3s. The congestion window was not large enough to recover from both losses in this case. The timeout near 9.8s was caused by 3 segment losses near 8.8s.

Similar problems can arise during the slow-start mode of TCP Lite. Here, the problem is even more acute as TCP-Lite doubles its congestion window every RTT. This increases the possibility of more than two consecutive segment losses when congestion occurs, rendering the fast retransmit mechanism ineffective. Hence in this case there is an even greater chance that TCP-Lite suffers a retransmission timeout. The timeout in Figure 4 near 1.5s is an example of this phenomenon.

The above discussion suggests that a TCP implementation that can recover more effectively from multiple segment losses during a single RTT interval can have improved performance over ATM networks. TCP Vegas incorporates a retransmission mechanism (Vegas-RM) that is more effective in handling multiple segment losses within one RTT. Upon arrival of the first or second non-duplicate ACKs after a fast retransmission, it uses timestamps to determine if more segments have been unacknowledged for a period exceeding the calculated timeout interval. If so, it retransmits the segments in question immediately. Furthermore, TCP-Vegas decreases its congestion window at most once during one RTT. The Vegas-RM mechanism proves to be effective in handling increased rates of multiple segment losses characteristic of an ATM network.

To evaluate the impact of Vegas-RM on the throughput over ATM networks, we performed simulations with a version of TCP-Vegas that has Vegas-SS and Vegas-CA disabled, leaving only the Vegas-RM enhancement active. The plots for Vegas-RM in Figures 7 and 8 show the effective throughput with this mechanism[5]. Note that the Vegas-RM mechanism alone does indeed afford a significant performance advantage over Lite on *both* ATM and packet networks. On ATM networks, Vegas-RM alone achieves approximately two thirds of the throughput gains that Vegas-AFR yields. Note also that Vegas-RM alone accounts for a large fraction of the throughput gains that Vegas obtains over Lite on packet networks.

[5] This Vegas implementation was derived from TCP-Reno but it does support the big-windows extension. Brakmo has shown in [7] that the code for TCP-Reno remains 5% behind TCP-Lite after his suggested fixes have been applied. We applied the RTO enhancements suggested in [7] to the Vegas code so as to make the codes compatible in performance when similar algorithms are being used.

5.1.1 Adaptive Fast Retransmit

We have shown that the Vegas-RM retransmission mechanism can more effectively handle multiple segment losses in one RTT. In particular, Vegas-RM handles such losses without incurring costly timeouts. However, consecutive losses stall the transmission pipeline for one RTT, causing a temporary underutilization of the network. Figure 4 and the previous analysis show that consecutive segment losses are very common in plain ATM networks. Hence, one would expect that the effective throughput of TCP over a plain ATM network can be improved by retransmitting *two* segments instead of just one during a fast retransmit. However, a TCP implementation cannot assume that it is using an ATM network. Retransmitting two segments over a packet network would cause unnecessary retransmissions, as on packet networks congestion commonly causes a single segment loss.

Our mechanism, called *adaptive fast retransmit*, is based on the idea of making TCP "learn" from its previous losses. If the earlier congestion periods caused a loss of consecutive packets, then it could be expected that in the next congestion period there would also be a consecutive-packet loss. Instead of retransmitting one packet and then waiting for three duplicate acks to detect more losses, TCP can then immediately retransmit two segments. A consecutive segment loss can then be recovered in almost the same time as a single segment loss.

Adaptive fast retransmit incorporates a 2-bit predictor that allows TCP to "learn" about consecutive segment losses. After encountering two back-to-back situations when consecutive segments were lost, on the next fast retransmit, TCP transmits two segments instead of just one. We dub this state the "double fast retransmit" state and the modified fast retransmit mechanism of TCP as the "adaptive fast retransmit" mechanism. Similarly when in "double fast retransmit" state, on encountering two occasions when a single-packet was lost, TCP reverts back to the original fast retransmit mechanism (the "single fast retransmit" state). Figure 9 shows the transitions between these states. We expect that TCP would mostly remain in state 0 on packet networks, while it

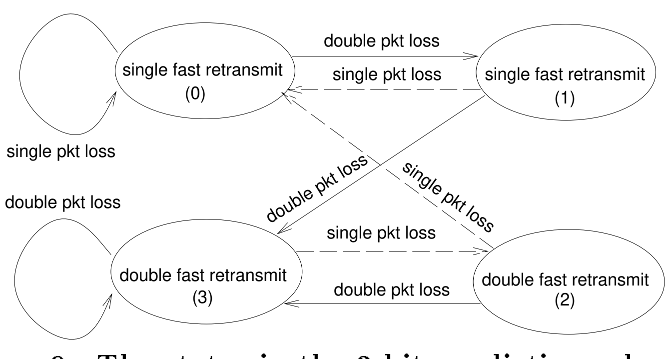

Figure 9 The states in the 2-bit prediction scheme

would mostly be in state 3 on ATM networks[6]. The retransmission state is maintained on a per-connection basis. This initial state is always 0 (single fast retransmit).

We have added AFR to both Lite and Vegas. Figures 7 and 8 show the effective throughput of TCP-Lite with AFR (Lite-AFR) and a version of TCP that incorporates both Vegas-RM and AFR (Vegas-RM-AFR) over ATM and packet networks. Figure 8 shows little or no change in the plots for the corresponding versions of TCP with and without AFR. This confirms that the throughput of TCP over packet switched networks remains unaffected by AFR. However over ATM, there is a noticeable throughput increase when AFR is used either in conjunction with Vegas-RM, or when it is added to Lite. In the case of Vegas-RM-AFR, the improvement directly reflects the avoidance of pipeline stalls associated with consecutive segment losses. With Lite-AFR, an additional factor in the throughput gain is the elimination of timeouts associated with consecutive losses.

An important issue is the increase in retransmissions caused due to the addition of the AFR mechanism. Our results show that these extra retransmissions never occupy more than 0.62% of the link bandwidth. On the average, the additional occupation is about 0.1%. It is to be noted that only a part of this is due to wrong decisions made by the AFR mechanism (transmitting 2 segments when only 1 was lost). The remaining is due to the normal operation of TCP algorithms when AFR eliminates long periods of inactivity associated with waits for retransmission timeouts.

5.2 Congestion Avoidance

We have shown that Vegas-RM and AFR can effectively deal with consecutive segment losses in ATM networks, and that the addition of these mechanisms to TCP can significantly improve the performance of TCP over ATM networks. In fact, the combination of Vegas-RM and AFR can bring the throughput on ATM within 10% of TCP-Lite's throughput over a packet network. In this section, we analyze the effect on performance by using Vegas-styled congestion avoidance.

TCP-Lite's congestion control scheme reduces the size of the congestion window only in response to segment losses. Furthermore, to probe the available capacity of the network, TCP-Lite periodically increases its congestion window until congestion occurs, as indicated by losses. As a result, the variation of TCP-Lite's congestion window exhibits a typical saw-tooth pattern as can be seen in Figures 4 and 5. TCP-Vegas tries to eliminate these self-induced losses by using congestion avoidance mechanisms. The plots for Vegas in Figures 7 and 8 show the performance of these mechanisms over ATM and packet network. Vegas differs from Vegas-RM only in the addition of its congestion avoidance mechanisms (Vegas-SS and Vegas-CA).

[6] Further experimentation may be required to tune this scheme for effective operation over heterogeneous networks.

The Vegas congestion avoidance mechanisms try to avoid losses by monitoring the difference between the expected sending rate and actual sending rate [6]. This difference is maintained so as to occupy only a fixed amount of buffer space in the network. These mechanisms do not perform well with very small buffer sizes. Thus Vegas finds itself depending more on its retransmission mechanism (Vegas-RM) with lower switch buffer sizes (less than 180K in Figure 7). For these sizes, the performance stays about 5-10% below packet-TCP. Fortunately, most of this difference can be eliminated by using adaptive fast retransmit with TCP-Vegas as shown in the plots for Vegas-AFR.

The Vegas-styled congestion avoidance mechanisms afford another advantage over the congestion control schemes of Lite. By periodically causing congestion, TCP-Lite will always tend to keep the buffers in the bottleneck switch nearly filled, independent of the size of these buffers. On the other hand, the Vegas mechanisms try to occupy only a fixed amount of buffer space in the network. Increasing the switch buffer size under otherwise identical conditions therefore increases the available buffer space in switches, improving the network's ability to absorb traffic bursts and reducing losses due to congestion.

5.3 Short Transfers

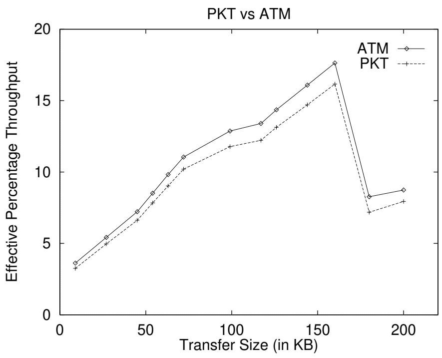

Figure 10 Vegas-AFR

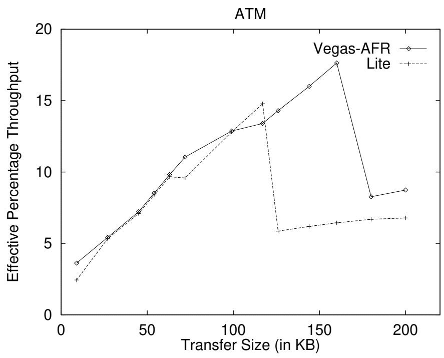

Figure 11 Vegas-AFR vs Lite

This section presents simulation results showing the performance of Vegas-AFR with short transfer sizes. The work in [2] shows that the mean size of HTTP transfers is currently about 13K. With the widespread use of the Web, the importance of these relatively short transfers cannot be underestimated.

Slow-start is known to slow down the time for short transfers because of the many RTTs taken for filling up the pipe [19]. TCP-Vegas starts slow-start with an initial congestion window of 2 segments instead of 1, as in Lite. This allows the sending of two segments in the first RTT (in addition to the connection setup overhead) and 4 segments in the first two RTTs (as Vegas does exponential increase every other RTT).

With an MTU of about 9K on ATM, most HTTP transfers will be completed in one RTT and almost all would be completed in two RTTs. It can be shown by an informal argument that starting the initial slow-start with an initial congestion window of 2 will mostly be better that starting with a congestion window of 1:

Case 1: If the network has space for 2 segments, then starting with an initial congestion window of 2 is guaranteed to be better than starting with 1. This is because 2 segments will otherwise be sent in two round trip times.

Case 2: If the network has space for only 1 segment, then Vegas would lose the other segment and will have to incur a retransmission timeout to recover from it. Lite on the other hand would lose the segment one RTT later and would also have to recover from a retransmission timeout. Lite would have transferred 2 segments however, while Vegas-AFR would have transferred 1 segment successfully. We expect this problem to be rare. The costly waits for the retransmission timeout would also shadow any gains that Lite has over Vegas-AFR in this case.

Case 3: If the network doesn't have space for even a single segment, both Lite and Vegas-AFR would have to depend on a retransmission timeout.

It is to be noted that the above argument cannot be extended to starting slow-start with more than 2 segments. This is because TCP normally never transmits more than 2 segments back to back in slow-start.

Figure 10 shows the variation of effective throughput with transfer size. The simulations were done using the topology shown in Figure 1. The switch buffer size is kept constant at 192.5K. It is evident from Figure 10 that our enhanced version of Vegas (Vegas-AFR) performs better over ATM than packet networks over the whole range of transfer sizes shown. Figure 11 shows a comparison of Vegas-AFR versus Lite for short transfers over ATM. It can be seen that the throughput of Vegas-AFR remains better than Lite for all transfer sizes up to 100K. This includes most transfer sizes relevant to Web traffic today. It is to be noted that Vegas-AFR has this advantage on account of its starting the initial slow-start with a congestion window value of 2. The same optimization applied to TCP-Lite would make it perform better than Vegas-AFR for transfer sizes shorter than about 100K. With larger sizes, Vegas-AFR would show better performance on account of the congestion avoidance mechanism built into its slow-start.

5.4 Fine-grained Clock for Timeouts

This section considers the issue of timer granularity in TCP. TCP-Lite performs round-trip time (RTT) estimation and schedules retransmission timeouts (RTOs) at a granularity of 500 msecs. It is widely recognized that these coarse-grained timers pose problems for TCP on high-bandwidth networks due to the large discrepancy between the actual RTT and the scheduled RTO [7, 14].

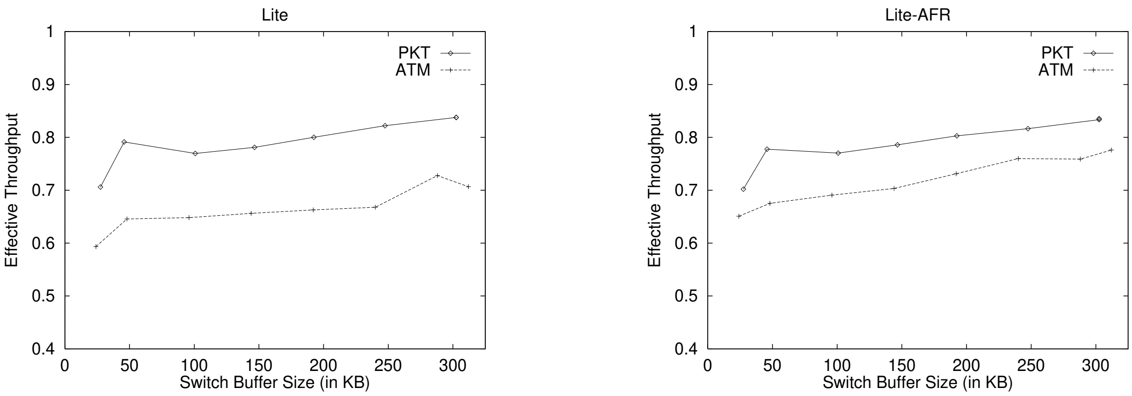

Figure 12 TCP-Lite with 10ms timer

We have performed simulations with TCP implementations that schedule timeouts at a granularity of 10ms. The performance of Vegas and Vegas-AFR on both ATM and packet networks are not significantly affected by the addition of fine-grained timeouts. This can be explained by the decreased reliance on timeouts to handle segment losses. We refrain from showing the data here, and conclude that with TCP-Vegas, fine-grained timeout do not appear to be necessary to achieve good throughput on ATM networks.

The addition of fine-grained timers does however improve the throughput of TCP-Lite over ATM. Figure 12 shows the effect of increasing the granularity of TCP-Lite's timeouts to 10ms. Also shown is the throughput achieved with a combination of adaptive fast retransmit and fine-grained timeouts (Lite-AFR). Without the adaptive fast retransmit, Lite's throughput over ATM remains about 15% below that achieved on packet networks. This shows that improving timer granularity alone is not sufficient to make TCP-Lite perform as well over ATM as on packet networks. With AFR, the difference in performance narrows down to below 10%.

6 Related Work

Romanow and Floyd [22] have proposed switch level enhancements to improve the performance of TCP over ATM. Partial Packet Discard involves dropping all subsequent cells from a TCP segment once one cell is dropped. Early Packet discard involves dropping entire TCP segments prior to switch buffer overflow. These techniques prevent the wastage of link bandwidth and network resources by useless cells from corrupted segments. Also by dropping more cells than are necessary, they prevent consecutive segment losses from occurring. So in effect, they cause TCP to experience single segment losses, and TCP is well specialized in recovering from those. A disadvantage of this approach is additional switch complexity and loss of layer transparency.

Bestavros and Kim [4] suggest making changes to TCP so that it uses the partial packets that it receives on an ATM network. They provide a few controlling parameters in their TCP implementation so that it can be used differently on an ATM network than on a packet switched network. The difficulty with this approach is that it requires an application to know a priori what type of network its traffic will traverse. This approach also causes loss of layer transparency.

Hoe [14] addresses the problem of improving the startup behavior of the TCP congestion control algorithms. During slow-start, TCP-Lite doubles its congestion window every RTT until either losses occur or the congestion window's size reaches a value specified by the slow-start threshold (ssthresh). During initial connection startup, the ssthresh is normally set to the maximum possible advertised window size as nothing is known about the underlying network. The uninhibited growth of congestion window can cause severe congestion and is confirmed by our simulation results with TCP-Lite.

The work in [14] proposes to set the initial value of ssthresh by measuring the bandwidth-delay product. The round-trip delay is measured by timing a distinguished segment. The bandwidth is estimated using the least-squares estimation on the time of receipt of three closely-spaced ACK's received at the sender. A similar attempt was also made in the context of TCP-Vegas [8]. There are problems associated with the measurement of both the bandwidth as well as the round-trip delay. ACK compression [27] and ACK clustering [23] can affect the estimation of the available bandwidth. Furthermore, the measurement of the round-trip delay by timing a segment would include the queuing delays. Even if the bandwidth was estimated correctly, the value calculated using this value of round-trip delay would be larger than the actual bandwidth-delay product. In particular, if the congestion window was allowed to grow to this value, it would cause a doubling of the queues at the bottleneck router/switch.

Vegas solves the problem of terminating the initial slow-start in a different way. As described earlier, Vegas-SS incorporates congestion avoidance into its slow-start mode. This allows Vegas to detect when its congestion window has exceeded the available capacity of the network, potentially before losses have occurred. As a result, Vegas is not completely dependent on the initial ssthresh value for controlling congestion.

Floyd [12] discusses the problem of invoking fast retransmit mechanism multiple times for the same window of data. The work in [14] attempts to make TCP deal effectively with multiple segment losses in the same congestion window. Although it does not incorporate Vegas-styled congestion avoidance, we expect the proposed TCP implementation to perform considerably better over ATM than TCP-Lite. Fall and Floyd [11] also investigates the effect of multiple packet losses on the congestion control algorithms of TCP-Reno. They point out that the absence of selective acknowledgments imposes limits on TCP's performance. Their work also shows that TCP with selective acknowledgments can effectively recover from multiple packet losses. In essence, SACK-TCP would also solve the problem that AFR tries to solve. While AFR requires changes only at the sender side, SACK-TCP requires making changes to existing implementations both at the sender as well as the receiver side.

The work in [8] attempts to control the “instantaneous sending rate” during slow-start. In slow-start, TCP-Lite sends segments at twice the rate at which the ACKs arrive (as two segments are sent for every ACK). Even if the value of ssthresh reflects the correct bandwidth-delay product segments might still be dropped if the available switch buffer space is not sufficient. An experimental version of Vegas called Vegas* was proposed in [8]. Instead of sending two segments back to back for each ACK during slow-start, an event is scheduled in the future to send the second segment. This scheduling of the event is such that the “instantaneous sending rate” remains the same as the bandwidth of the bottleneck link. The problem with this approach is that very fine-grained timers are required which can schedule events at the granularity of the inter-arrival times of ACKS. Also this method can also fail in the presence of ACK compression [27] when the ACK spacing does not give a reliable estimate of the available bandwidth. The work in [26] discusses the intrinsic limitations of timers in improving performance.

7 Conclusions

This paper makes four contributions. First, we show that the inefficiency of TCP-Lite's retransmission algorithms in recovering from multiple-packet losses is a major factor responsible for throughput degradation over ATM. Second, we propose a new technique called *adaptive fast retransmit* (AFR) that allows TCP to recover from consecutive segment losses without delay. Thirdly, we show that it is possible to avoid TCP throughput degradation over plain ATM networks without any switch level enhancements. We demonstrate a version of TCP-Vegas with adaptive fast retransmit that performs equally well on plain ATM and packet networks, in addition to yielding better performance over TCP-Lite on packet networks. These simulation results hold for both long and short TCP transfers. Finally, we present a study of individual TCP enhancements, their dynamics over ATM and packet networks, and their impact on performance over ATM. We have also shown that by making limited modifications to TCP-Lite in the form of AFR and a fine-grained clock for scheduling timeouts, it can be made to perform reasonably well over ATM.

8 Acknowledgments

We thank Lawrence Brakmo for his many insightful comments. We also thank Dejan Mircevski and Zhiying Chen for their suggestions and help with this paper.

References

[1] J. Ahn, P. Danzig, Z. Liu, and E. Yan. Evaluation of TCP Vegas: Emulation and Experiment. In *Proceedings of the SIGCOMM '95 Symposium*, 1995.

[2] M. F. Arlitt and C. L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proceedings of the ACM SIGMETRICS '96 Conference*, Philadelphia, PA, Apr. 1996.

[3] R. Atkinson. Default IP MTU for use over ATM AAL5. *IETF Internet Draft*, Feb. 1994. Available by anonymous ftp from ds.internic.net:internet-drafts/internet-drafts/draft-ietf-atm-mtu-07.txt.

[4] A. Bestavros and G. Kim. TCP Boston: A Fragmentation-tolerant TCP Protocol for ATM Networks. In *Proceedings of INFOCOM*, Apr. 1997.

[5] F. Bonomi, K. Fendick, and N. Giroux. The Available Bit Rate Service. http://palgong.kyungpook.ac.kr/~bckim/abr.html.

[6] L. Brakmo, S. W. O' Malley, and L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of the SIGCOMM '94 Symposium*, pages 24–35, 1994.

[7] L. Brakmo and L. Peterson. Performance Problems in 4.4BSD TCP. *ACM Computer Communication Review*, 25(5):69–86, Oct. 1995.

[8] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, Oct. 1995.

[9] L. S. Brakmo and L. L. Peterson. Experiences with Network Simulation. In *SIGMETRICS*, 1996.

[10] B. J. Ewy, J. B. Evans, V. S. Frost, and G. J. Minden. TCP/ATM Experiences in the MAGIC Testbed. ftp://ftp.tisl.ukans.edu/pub/papers/TCP-Perform.ps.

[11] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z, Dec. 1995.

[12] S. Floyd. TCP and Successive Fast Retransmits. ftp://ftp.ee.lbl.gov/papers/fastretrans.ps, May 1995.

[13] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, Sept. 1992.

[14] J. C. Hoe. Improving the Start-up Behaviour of a Congestion Control Scheme for TCP. In *Proceedings of the SIGCOMM '96 Symposium*, 1996.

[15] N. C. Hutchinson and L. L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[16] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of the SIGCOMM '88 Symposium*, pages 314–32, Aug. 1988.

[17] V. Jacobson. Berkeley TCP evolution from 4.3-tahoe to 4.3-reno. In *Proceedings of the Eighteenth Internet Engineering Task Force*, Aug. 1990.

[18] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for ExperimentalDesign, Measurement, Simulation and Modelling* . John Wiley & Sons, New York, 1991.

[19] J. C. Mogul. The Case for Persistent-Connection HTTP. In *Proceedings of the SIGCOMM '95 Symposium*, 1995.

[20] R. Morris and H. T. Kung. Impact of ATM Switching and Flow Control on TCP Performance: Measurements on an Experimental Switch. *GLOBECOM*, 1995. ftp://virtual.harvard.edu:/pub/htk/atm/globecom995.ps.gz.

[21] M. Perloff and K. Reiss. Improvements to TCP Performance in High-Speed ATM Networks. *Communications of the ACM*, 38(2):90–100, Feb. 1995.

[22] A. Romanow and S. Floyd. The Dynamics of TCP Traffic over ATM Networks. In *Proceedings of the SIGCOMM '94 Symposium*, pages 79–88, 1994.

[23] S. Shenker, L. Zhang, and D. Clark. Some Observations on the Dynamics of a Congestion Control Algorithm. *ACM Computer Communication Review*, 20(4):30–39, Oct. 1990.

[24] W. Stevens. *TCP/IP Illustrated Volume 1 : The Protocols.* Addison-Wesley, Reading, MA, 1994.

[25] G. Wright and W. Stevens. *TCP/IP Illustrated Volume 2 : The Implementation.* Addison-Wesley, Reading, MA, 1995.

[26] L. Zhang. Why TCP Timers Don't Work Well. In *Proceedings of the SIGCOMM '86 Symposium*, Aug. 1986.

[27] L. Zhang, S. Shenker, and D. D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proceedings of the SIGCOMM '91 Symposium*, pages 133–148, 1991.

A Graph Description

Figures 13 and 14 explain the trace graphs used in this paper. Figure 13 shows some general information shown in the graphs:
(1) Hash marks on the x-axis indicate when an ACK was received. (2) Hash marks at the top of the graph indicate when a segment was sent. (3) The numbers on the top of the graph indicate when the n^{th} kilobyte (KB) was sent. (4) Diamonds on top of the graph indicate when TCP checked whether a coarse-grained timeout should happen. (5) Black circles on top of the graph indicate that a coarse-grained timeout actually

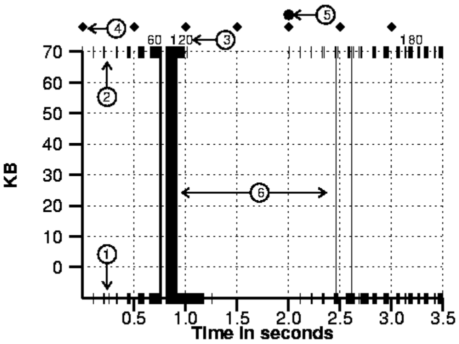

Figure 13 General Elements

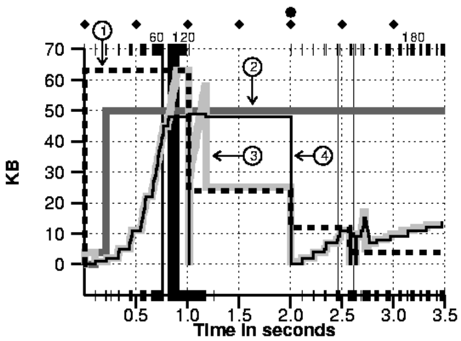

Figure 14 TCP Windows

occurred. (6) Solid vertical lines running the whole height of the graph indicate when a segment that is retransmitted was actually sent.

Figure 14 shows traces of the TCP windows:
(1) The dashed line gives the slow-start threshold (ssthresh). (2) The dark gray line gives the send window (minimum of the sender's buffer size and receiver's advertised window) and gives the upper limit on the amount of unacked data. (3) The light gray line gives the congestion window. (4) The thin-line gives the actual amount of unacked data.