Explicit Window Adaptation: A Method to Enhance TCP Performance

Lampros Kalampoukas
Anujan Varma

Computer Engineering Department
University of California
Santa Cruz, CA 95064
E-mail: {lampros, varma}@cse.ucsc.edu

K. K. Ramakrishnan

AT&T Labs - Research
Florham Park, NJ 07932
E-mail: kkrama@research.att.com

This research is supported by the Advanced Research Projects Agency (ARPA) under Contract No. F19628-96-C-0038 and by the NSF Young Investigator Award No. MIP-9257103. The OPNET modeling tool used for simulations was donated to us by MIL-3, Inc.

Explicit Window Adaptation: A Method to Enhance TCP Performance

Abstract

We study the performance of TCP in an internetwork consisting of both rate-controlled and non-rate-controlled segments. A common example of such an environment occurs when the end systems are part of IP datagram networks interconnected by a rate-controlled segment, such as an ATM network using the ABR service. In the absence of congestive losses in either segment, TCP keeps increasing its window to its maximum size. Mismatch between the TCP window and the bandwidth-delay product of the network will result in accumulation of large queues and possibly buffer overflows in the devices at the edges of the rate-controlled segment, causing degraded throughput and unfairness. We develop an explicit feedback scheme, called *Explicit Window Adaptation* based on modifying the receiver's advertised window in TCP acknowledgments returning to the source. The window size indicated to TCP is a function of the free buffer in the edge device. Results from simulations with a wide range of traffic scenarios show that this explicit window adaptation scheme can control the buffer occupancy efficiently at the edge device, and results in significant improvements in packet loss rate, fairness, and throughput over a packet discard policy such as Drop-from-Front or Random Early Detection.

Keywords: TCP, congestion control, buffer management, explicit window adaptation.

1 Introduction

Congestion control in the current Internet is primarily a function of the Transmission Control Protocol (TCP). TCP congestion control is based on controlling the end-to-end window as a function of the congestion state of the network. The TCP congestion control mechanisms continue to evolve as more insight is gained on their behavior, and as new requirements emerge.

An important attribute of TCP congestion control mechanisms is that they do not assume any support from the network for explicit signaling of congestion state. TCP infers the congestion state of the network from implicit signals — arrival of acknowledgements (acks), timeouts, and receipt of the duplicate acks. The evolution of the congestion window in a TCP source consists of two phases: the *slow start* phase and the *congestion avoidance* phase. The slow-start phase occurs during startup, as well as when a packet loss is detected by way of a timeout at the source. During the slow-start phase, the congestion window essentially doubles during each round-trip time (RTT), until a threshold window size known as *slow-start threshold* (`ssthresh`) is reached. At this point the host enters the congestion avoidance phase where TCP is probing for additional bandwidth by increasing the window more slowly, at the rate of one segment per RTT. *

In the original "Tahoe" version of TCP, the only means of detecting a packet loss was by a timer expiration at the source, causing the slow-start to be triggered upon every such loss [1]. This can lead to severe oscillations in buffer occupancy at the routers, as well as in the connection throughput, when losses are random. This deficiency was corrected in the later "Reno" version by adding the *Fast Retransmit* and *Fast Recovery* mechanisms [2, 3]. The Fast Retransmit mechanism retransmits a packet when three duplicate acks are received at the source, without waiting for the timer to expire. The Fast Recovery mechanism avoids slow-start in such cases by setting the congestion window to approximately half its current value and keeping the connection in the congestion avoidance phase. This avoids severe fluctuations in throughput and buffer occupancy when the congestion is not severe enough to warrant a large reduction in offered load.

An important consequence of using the Fast Retransmit and Fast Recovery mechanisms at the TCP source is that it becomes advantageous for routers to signal congestion to the source by discarding a packet *early* at the onset of congestion, without waiting for its buffers to become full, when the router may be forced to discard multiple packets. This avoids the source entering slow-start, and the resulting oscillations in throughput. The *Random Early Detection* (RED) scheme [4] for routers is based on this idea. Multiple losses within the same TCP window, however, can still cause timeouts at the source, causing slow-start to be invoked.

In this paper, our interest is on the behavior of TCP congestion control algorithms in an internetwork consisting of both rate-controlled and non-rate-controlled segments. The most common example of such an environment occurs when the end systems are part of IP datagram networks that are interconnected by a rate-controlled ATM virtual circuit over a wide area. In the absence of congestive losses, the TCP congestion window grows up to the maximum window size permitted by the destination system. However, the use of rate control in the ATM segment enables the queue lengths to be maintained small in the ATM switches. The result is that most of the TCP window is now buffered in the routers at the edges of the rate-controlled channel, causing severe congestion, degraded throughput, and unfairness. Our objective in this paper is to develop a scheme for controlling congestion in the edge routers in such an environment. Note that this problem is not limited to internetworks employing rate-controlled ATM segments, but applies in general to

*The TCP window is actually maintained in terms of bytes. For simplicity, however, we assume that the TCP segments are of fixed size so that the window size can be maintained in segments.

internetworks where the edge routers employ some form of rate-based scheduling at the entry point to a wide-area network.

One approach to control congestion in the edge router is to employ an intelligent packet-discard policy such as Random Early Detection (RED), so that congestion can be signaled to the TCP sources early. However, when the delay in the wide-area segment is large, this policy may still cause timeouts, forcing connections to enter slow start. Another discard policy that has been shown to work well with TCP is *Drop-from-Front*, which drops the packets at the head of the FIFO queue in the router when the buffer becomes full [5]. This approach, however, may still cause back-to-back losses and unfairness. Both RED and Drop-from-Front are generic solutions with the potential of improving the efficiency and fairness of networks with TCP-controlled traffic, without being tied to a specific network technology. Being general does not allow them to take full advantage of information that is available at the entry point of a rate-controlled network segment.

The objective of our scheme is to match the sum of the windows of active TCP connections sharing the buffer in the edge router to the effective network bandwidth-delay product, thus avoiding packet losses whenever possible. This is achieved by explicitly controlling the window size of the connections as a function of the available space in the buffer at the edge router. The window size information is communicated by the edge router to the TCP sources by modifying the window advertisement field in the acks flowing back to them. Our scheme, which we call *Explicit Window Adaptation*, does not require modifications to the TCP implementations in the end systems, and does not need to maintain per-flow state in the router. Results from extensive simulations with a wide range of network configurations show that our scheme is able to provide almost perfect throughput and fairness when the delays in the local segments are small compared to that in the rate-controlled wide-area network. Even when the latter is small compared to the former, Explicit Window Adaptation resulted in close-to-ideal throughput and fairness.

The performance of TCP when operating exclusively over IP datagram networks has been the subject of extensive research in the past [6, 7, 8, 9, 10, 11, 12, 13, 14, 5]. Similarly, TCP performance over ATM networks using the Available Bit-Rate (ABR) service has also been studied [15, 16, 17]. These studies provide valuable insights into TCP dynamics and how TCP congestion control mechanisms interact with ABR congestion control algorithms. Several modifications to TCP have also been proposed with the objective of improving both network utilization and fairness. Floyd [18] proposed to modify TCP to include *Explicit Congestion Notification (ECN)* from routers to the source. By combining explicit notification with RED, the performance of both delay-sensitive (telnet-like) and delay-insensitive (ftp-like) traffic can be improved [18]. Other approaches such as *Tri-S* [19] and *TCP-Vegas* [20] attempt to estimate the bandwidth-delay product for each TCP connection and adjust the window size based on this estimate. However, these schemes introduce complexity in the end-system and require extensive modifications to current TCP implementations.

This paper is organized as follows: In the next section, we provide the motivation for this work by discussing simulation results from an example network configuration. We introduce the Explicit Window Adaptation scheme in Section 3 and discuss its implementation. In Section 4, we present results from extensive simulations of the scheme with a wide range of network parameters and traffic scenarios. We conclude the paper in Section 5 with a summary of the results.

2 Motivation

Window-based protocols control the amount of outstanding data in the network. Even though several extensions to TCP have been proposed with the objective of adapting the window size to the actual bandwidth-delay product of the underlying network [19, 20], current implementations require a packet loss to reduce the congestion window size. In the absence of such losses the congestion window increases up to the maximum socket buffer advertised by the receiver.

When TCP traffic is carried over an ATM network, the window-based congestion control mechanisms of TCP can interact with the rate-based control mechanisms in the ATM network in undesirable ways. These interactions are a result of the mismatch in the dynamics introduced by rate-based and window-based control [21]. TCP controls the total amount of data injected into the network, but does not control its burstiness. However, the service rate the connection receives at the edge of the ATM network is constant. Thus, segments transmitted by the TCP connection arrive typically as a burst at the edge router, and are drained at the available rate on the ATM network. As a result, assuming no losses elsewhere in the network, the TCP window size grows to eventually cause a buffer overflow at the access point into the rate-controlled part of the network. This behavior tends to be periodic, and may result in loss of throughout and overall performance degradation. The effect of such periodic losses on TCP performance was analyzed by Lakshman, et. al. [7].

Although the above problem could occur in more general environments, we focus our attention in this paper on TCP/IP internetworks where the end systems are connected to legacy LANs (such as Ethernet or Token Ring), with the LANs interconnected through a rate-controlled ATM virtual circuit. Figure 1, shows the example network configuration that will be used in our study. We will refer to the routers at the boundary between the two networks as *ATM Access Points* (AAPs) and to the LAN segments as IP networks. For IP over ATM we assume the framework described in [22]. According to this framework, a single VC is set up for carrying ABR traffic between a source and a destination system. TCP connections destined to different IP networks will be carried over separate VCs. Multiple TCP connections set up between the same pair of source and destination IP networks will be multiplexed into a single ATM virtual circuit. In Figure 1, we identify each IP-to-ATM router with the label of the IP network it is connected to, i.e., we will refer to IP-to-ATM Router 1 as AAP-1.

Unless stated otherwise, we assume that the AAPs use *Drop-Tail* packet discard policy. With Drop-Tail, the packet that arrives at a full buffer is dropped. This scheme is widely used today in switches and routers, due mostly to its simplicity. This simplicity, however, comes at the cost of degraded performance: when the buffer becomes full it is likely that many connections will face a loss at the same time. These synchronized losses result in a corresponding synchronized reaction by multiple connections simultaneously reducing their windows. The potential over-correction may eventually lead to reduced throughput [14]. Furthermore, if multiple losses occur from a single connection within one RTT, the connection may be forced into the slow-start phase.

We will now consider two scenarios to illustrate the effects of packet losses occurring at the AAPs on TCP performance. Both scenarios use the network configuration shown in Figure 1. However, a different set of connections is activated in each of these scenarios.

In the first scenario, sources 1, 2b, and 3b are activated and their traffic is destined to IP networks 2, 3, and 4, respectively. The traffic for each of the three connections will be carried by different VCs in the ATM network. We assume the use of per-VC queueing and scheduling in the AAPs to provide isolation between the

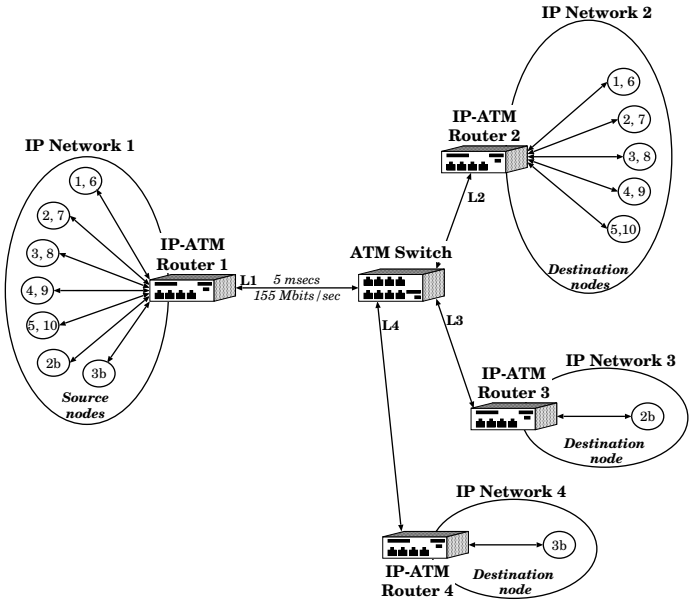

Figure 1: Generic network topology with IP and ATM subnetworks.

different ATM virtual circuits. We also assume the use of an efficient rate allocation algorithm in the ATM network for support of ABR service, providing the three VCs with a fair and loss-free environment [23]. We consider this as a best case scenario. Each of the links in this configuration has a capacity of 155 Mbits/sec. The one-way propagation delays of connections 1 and 2b in IP Network 1 are set to 3 milliseconds and that of connection 3b to 0.8 μsec. The delays in IP Networks 2, 3, and 4 as well as the link delays L1, L2, L3, and L4 are assumed to be insignificant. The TCP segment size is set to 1500 bytes. The buffer size for each ATM connection at the ATM layer in AAP-1 was set to 64 KBytes and the maximum socket buffer size at the receivers for all three connections was set to 200 KBytes.

Figures 2 and 3 show the sequence number growth and the congestion windows for all three active connections. Observe that all three connections periodically go through slow-start because of buffer overflows in AAP-1. As a result, the performance is poor even though there is per-VC queueing and scheduling at the ATM layer. The average throughput for each connection is shown in Figure 4, where the long connections can be seen to receive significantly lower throughput as compared to the short connection. This can be attributed to the long connections losing multiple back-to-back packets and recovering slower from these losses.

The problem becomes even more pronounced when traffic from multiple TCP connections is multiplexed into a single ATM virtual circuit. To study such a configuration, we multiplexed the traffic from ten TCP connections into a single VC. The network configuration for this simulation remains as shown in Figure 1, but the propagation delay of link L1 was set to 5 milliseconds and that of each link in the IP Network to 0.5 milliseconds. A single VC was established between IP Networks 1 and 2, carrying multiplexed traffic from the ten TCP connections $(1-10)$. The segment size for TCP was set to 1500 bytes. All ten connections were opened at time $t = 0$. We studied the way TCP adapts to changing network conditions by varying the number of active TCP connections: At time $t = 10$ secs we closed five of the ten active connections (6–10) and at time $t = 15$ secs we closed another three (3–5). The buffer size at AAP-1 was set to 200 Kbytes

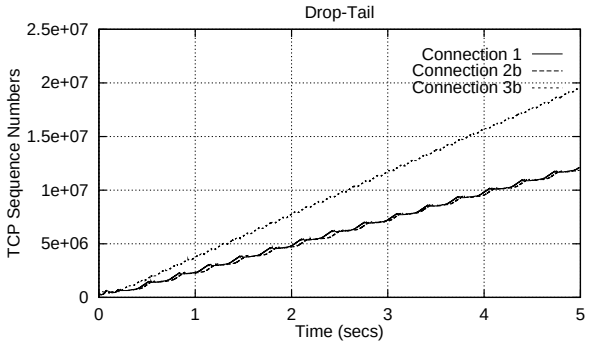

Figure 2: TCP sequence numbers for the three connections (64 Kbytes IP buffer size in the host, packet size = 1500 bytes).

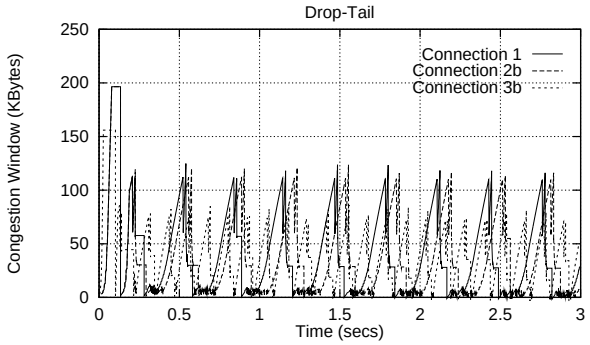

Figure 3: TCP Congestion Windows for one long and one short connection (64 Kbytes IP buffer size in the IP-ATM router, 200 Kbytes maximum socket buffer at receiver, packet size = 1500 bytes).

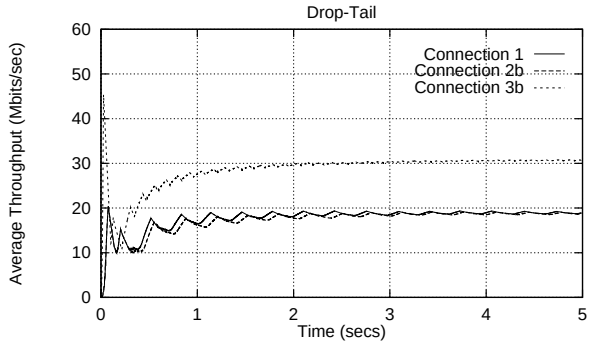

Figure 4: Average effective throughput per TCP connection (64 Kbytes IP buffer size in the host, 200 Kbytes maximum socket buffer at receiver, packet size = 1500 bytes).

which is approximately equal to the bandwidth-delay product of the network. The destination buffer, which effectively sets the maximum TCP window size, was also set to 200 Kbytes. Thus, a single TCP connection may potentially operate at the maximum available link capacity if necessary.

Figure 5 illustrates the sequence number growth of the active TCP connections. We observe that once steady state is reached, all connections make fair progress during the first 10 seconds. The utilization of link L1, measured in intervals of 250 milliseconds, is shown in Figure 6. During the first 10 secs the utilization is approximately 70%. The throughput loss is mainly due to the synchronization of the losses for the connections and their simultaneous recovery. This results in idling the link substantially.

At time $t = 10$ secs, when five of the ten connections close, we would expect the slope of the sequence number growth to almost double since the available bandwidth per connection also doubles. However, we observe that it remains almost unchanged, limiting the utilization of link L1 to approximately 50%. This is because of the synchronization between the connections during periods following a buffer overflow. Finally, during the last 5 seconds of the simulation, when only two TCP connections are active, we observe that the link utilization improves to approximately 80%. This, however, occurs at the cost of degraded fairness. This is because, on retransmitting a lost packet and recovering from the loss, one of the connections (the connection that had the larger window prior to the loss) will increase its window much faster, thus obtaining

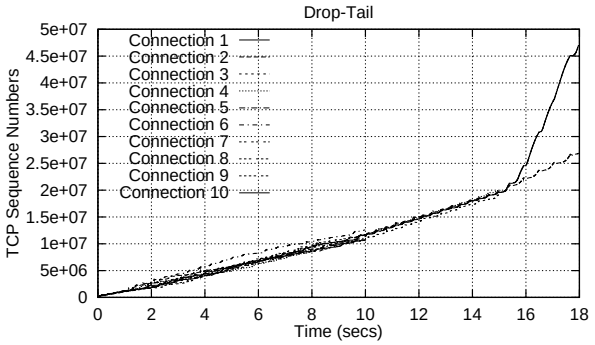

Figure 5: Sequence number growth for TCP connections with Drop-Tail buffer management (IP Network delays = 0.5 msecs, ATM backbone delay = 5 msecs, $ssthresh$ = 64 KBytes initially).

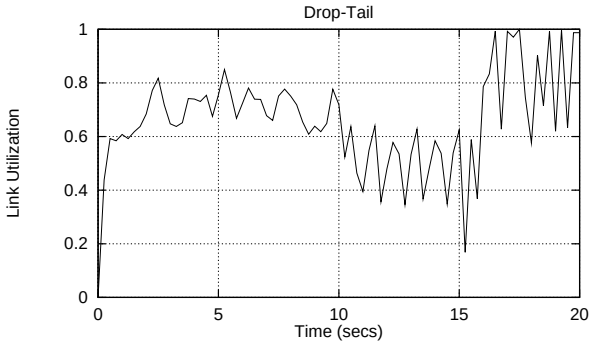

Figure 6: Effective utilization of the ATM link at IP-ATM Router 1 with Drop-Tail buffer management measured every 250 msecs (IP Network delays = 0.5 msecs, ATM backbone delay = 5 msecs, initial value of $ssthresh$ = 64 KBytes).

a larger portion of the available bandwidth.

2.1 TCP Performance with Drop-from-Front AAPs

To reduce the tendency of Drop-Tail routers to synchronize losses of multiple connections, and the possibility of multiple losses to a connection in a round-trip, two alternative policies have been suggested: Drop-from-Front [5] and Random Early Detection [4]. With Drop-from-Front, when a packet arrives at a full queue, the packet stored at the head of the queue is dropped. If service for the packet at the head of the queue has already started, the following packet in the queue is dropped instead. The arriving packet is always accepted, using the space freed by the dropped packet.

The Drop-from-Front strategy has the potential to (i) provide faster feedback to traffic sources regarding congestion, and (ii) break synchronization between competing connections, thus improving fairness. Drop-from-Front provides an indication of congestion about one buffer drain time earlier. This early feedback to TCP sources results in shortening the congestion episode and significantly reduces the subsequent over-correction. Looking at the head of the queue is similar to observing the system state as far in the past as possible. Consequently, the distribution of connections that occupy the head of the queue position is expected to be closer to the actual bandwidth distribution when the congestion epoch started (or as close as possible to that time). Hence, the dropping probability is more likely to be proportional to the inherent bandwidth distribution among the connections.

We investigate now how Drop-from-Front affects TCP performance when applied to AAPs. We use again the network topology of Figure 1 and the traffic scenario described in the previous simulation experiment: during the interval $(0, 10)$ seconds, ten TCP connections are active; at time $t = 10$ seconds five of the ten connections close; and at $t = 15$ seconds three more connections close, leaving only two active. The sequence number growths for the active TCP connections are presented in Figure 7, and the utilization of link L1 in Figure 8.

During the first 10 seconds all the connections make fair progress, with better overall throughput than that with Drop-Tail. From Figure 8, we observe that the link utilization during the first 10 seconds is

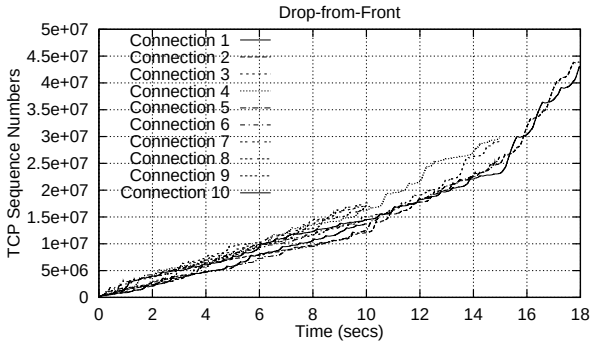

Figure 7: Sequence number growth for TCP connections with Drop-Front buffer management (IP Network delays = 0.5 msecs, ATM backbone delay = 5 msecs, *ssthresh* = 64 KBytes initially).

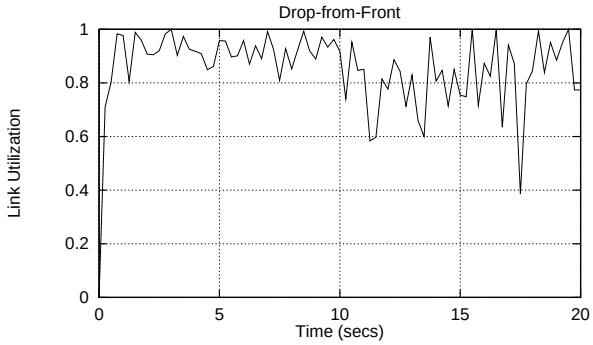

Figure 8: Effective utilization of the ATM link at IP-ATM Router 1 with Drop-Front buffer management measured every 250 msecs (IP Network delays = 0.5 msecs, ATM backbone delay = 5 msecs, *ssthresh* = 64 KBytes initially).

approximately 90% compared to 70% with Drop-Tail AAPs. Even when only five TCP connections are active we can again see that the throughput is significantly better, approximately 75% compared to 50% for Drop-Tail. While better than Drop-Tail, this loss in throughput indicates that the mechanism is still unable to overcome the fundamental problem: there is throughput degradation when packet loss is used as the sole indicator of congestion.

Finally, during the last 5 seconds when only two TCP connections are active, the throughput is again quite high, approximately 90%. In contrast to the Drop-Tail case, the progress of the two active TCP connections is fair.

2.2 TCP performance with Random Early Detection (RED) AAPs

The Random Early Detection (RED) scheme also has the same objective of improving TCP throughput and fairness [4]. Congestion is determined by comparing the average queue size to a predetermined threshold. Congestion can be detected even before the buffer is full, thus controlling the average queue size. In periods of congestion RED *marks* incoming packets in order to indicate congestion to TCP sources. Marking can be performed by setting a bit in the packet header. The current version of TCP protocol, however, does not support such explicit congestion notification. Therefore, congestion must be signaled to the source through a packet loss. RED applies randomization in selecting the connections to notify, with the objective of avoiding global synchronization and to achieve a dropping probability that is proportional to the inherent bandwidth distribution among the connections.

RED maintains two buffer occupancy thresholds in the router: a low threshold min_{th} and a high threshold max_{th}. No packets are dropped when the average queue size is below min_{th}. When the average queue size exceeds max_{th}, all incoming packets are dropped with probability one. When the average queue length is between min_{th} and max_{th}, the incoming packet is dropped with a probability that is a linear function of the average queue length, varying linearly from zero to a pre-determined maximum value max_p.

Using the configuration of Figure 1 and the same traffic scenario used in the previous experiments, we studied the performance when RED was used. We ran two sets of simulations with different sets of parameters

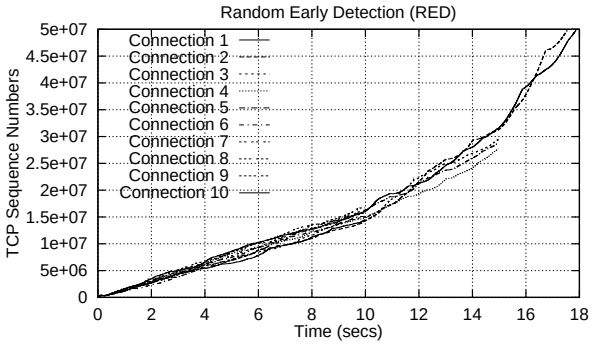

Figure 9: Sequence number growth for TCP connections with RED buffer management ($max_p = 0.02$, $min_{th} = 15$, $max_{th} = 30$, IP Network delays = 0.5 msecs, ATM backbone delay = 5 msecs, $ssthresh$ = 64 KBytes initially).

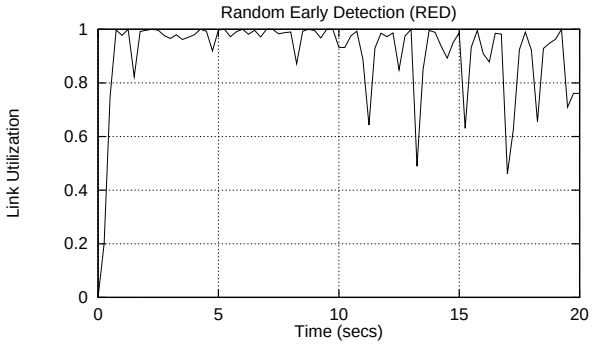

Figure 10: Effective utilization of the ATM link at IP-ATM Router 1 with RED buffer management measured every 250 msecs ($max_p = 0.02$, $min_{th} = 15$, $max_{th} = 30$, IP Network delays = 0.5 msecs, ATM backbone delay = 5 msecs, $ssthresh$ = 64 KBytes initially).

for RED. First, we set min_{th} to 15 packets, max_{th} to 30 packets and max_p to 0.02. The total buffer size in AAP-1 was set to 110 packets, equivalent to the bandwidth-delay product of a single TCP connection. The sequence number growth plots for this first experiment are shown in Figure 9 and the utilization is shown in Figure 10. During the first 10 seconds all connections progress in a fair manner and the overall efficiency is high. When the number of active connections drops from ten to five at $t = 10$ seconds, and subsequently from five to two at $t = 15$ seconds, the active connections manage to claim most of the leftover bandwidth. However, as can be seen in Figure 10, the link utilization is still not ideal, reflecting the penalty of lost throughput in response to packet loss. However, RED succeeds in achieving fairness.

To study the sensitivity of the scheme to the maximum marking probability, we increased max_p from 0.02 to 0.05, while keeping the rest of the parameters the same. This value is within the range suggested for RED [4]. The sequence number plots for this case are given in Figure 11, and the measured link utilization in Figure 12. We observe that the connection progress is again fair but the overall performance is further degraded, and the results are now comparable to those from Drop-from-Front.

The simulation results presented in this section suggest that RED has the potential of improving performance beyond that achieved by Drop-Tail and Drop-from-Front. However, the sensitivity to parameters suggests that further understanding is needed on how to tune the parameters.

3 Explicit Window Adaptation

The packet discard algorithms discussed in the previous section attempt to control the TCP windows implicitly by forcing the connections to respond to packet losses. An alternative approach is to control the window sizes explicitly from the bottleneck point in the network as a function of the effective bandwidth-delay product of the network. Such a scheme requires two key components: (i) A mechanism to signal window updates from the network to the source, and (ii) a scheme at the bottleneck point to estimate the window size based on the congestion state of the network. The former can be accomplished without modifications to the TCP

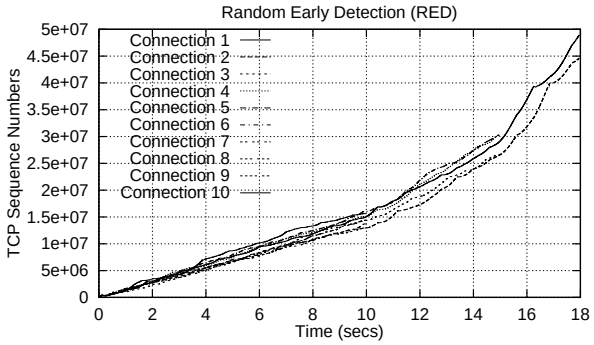

Figure 11: Sequence number growth for TCP connections with RED buffer management ($max_p = 0.05$, $min_{th} = 15$, $max_{th} = 30$, IP Network delays = 0.5 msecs, ATM backbone delay = 5 msecs, $ssthresh$ = 64 KBytes initially).

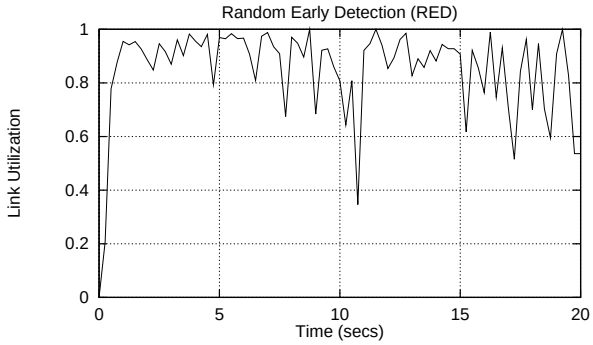

Figure 12: Effective utilization of the ATM link at IP-ATM Router 1 with RED buffer management measured every 250 msecs ($max_p = 0.05$, $min_{th} = 15$, $max_{th} = 30$, IP Network delays = 0.5 msecs, ATM backbone delay = 5 msecs, $ssthresh$ = 64 KBytes initially).

protocol by allowing the network elements to modify the *receiver's advertised window* field carried by TCP acknowledgements from the destination to the source. The latter problem, however, is significantly more difficult: First, when there are multiple bottlenecks on the path of a TCP connection, the window estimation algorithms at these bottlenecks may interact in undesirable ways. Second, estimating the bandwidth-delay product of the link from the bottleneck element is often as difficult as estimating it at the source. Finally, the advantages of the scheme must be compared against those of implicit (packet discard) schemes of comparable complexity. For example, if the scheme requires maintaining per-connection state at the routers, its performance needs to be compared against alternative packet discard schemes that could be designed when the routers provide per-flow queueing and scheduling.

For the specific network environment we consider in this paper, however, the problem is much simpler. First, we assume that the only bottleneck in the path of the TCP connections occurs at the AAP. Second, both the bandwidth available in the ATM segment, and the delay through it, remain relatively steady over short timescales, independent of the number of TCP connections transported over it. This makes it easier for the AAP to estimate the available bandwidth-delay product for each TCP connection sharing the outgoing ATM virtual circuit. Finally, the per-VC queueing and buffering at the ATM layer isolates the TCP connections sharing a single virtual circuit from other traffic. In this section we outline a scheme that takes advantage of these facts, for explicitly controlling the TCP congestion windows from the AAP.

The *Explicit Window Adaptation* (EWA) scheme provides TCP with explicit feedback on the state of the AAP buffer. The objective is to allow TCP connections to grow their window to fill the network pipe (i.e., bandwidth-delay product). Any further increase of the window size contributes only to increased queueing delay, not improved throughput. Optimal setting of the window size, however, requires knowledge of the RTT and the bandwidth-delay product of the network [24]. Such information is usually not available at network elements. Instead, EWA determines when the network pipe is full by monitoring the occupancy of the buffer serving the outgoing ATM virtual circuit at the AAP: non-empty buffer is either an indication of a full pipe or bursty traffic. At that time EWA "marks down" slowly the window size fed back to the

TCP source so that the steady-state buffer occupancy can be maintained well below the buffer capacity, still keeping the network pipe full. Furthermore, controlling the buffer occupancy allows the network elements to accommodate short-term bursts while avoiding buffer overflows or underflows most of the time. When multiple TCP connections share a common VC, the approach matches the aggregate window sizes of all active TCP flows to the bandwidth-delay product of the network while at the same time providing all the connections with similar feedback to achieve fairness.

EWA sends explicit feedback to TCP sources to adjust their window sizes. The feedback is carried by returning TCP acknowledgments in the *receiver's advertised window* field. If the current value in the receiver's advertised window, which is set by the destination system, exceeds the feedback value computed in the AAP, the receiver's advertised window is marked down to the feedback value. The computed feedback is a function of the free buffer space at the AAP. In that sense it is similar to the idea proposed by Choudhury and Hahne [25] for controlling dynamic buffer thresholds in a shared-memory switch. In our case, however, the feedback computed is used to adapt the TCP window maintained at the sources in order to limit packet losses in the AAP's buffer.

Let us denote with $B_e(t) = B - Q(t)$ the empty buffer space at time t when a returning ack arrives at an AAP, where B is the total buffer space and $Q(t)$ is the total buffer occupancy at time t. Let $W_r(t)$ denote the value in the receiver's advertised window field seen in the ack. The algorithm computes a target window size for the TCP connection as a function of the available buffer, that is $f(B_e(t))$. This computed value is then used to mark down the receiver's advertised window field in the acknowledgement. Since setting the window size smaller than the maximum segment size (MSS) negotiated during connection establishment can lead to starvation and deadlocks, a minimum window size of MSS is enforced. Thus, the feedback value, $W_r'(t)$, used to set the receiver's advertised window field, is computed at the AAP as

$$W_r'(t) = \max(\min(W_r(t), f(B_e(t))), MSS). \tag{3.1}$$

The window size computed by Eq.(3.1) is used to modify the returning acknowledgements, regardless of the connections they belong to. That is, all TCP connections are treated equally and receive the same feedback for the same buffer occupancy. This avoids the need to maintain the number of active TCP connections or their states in the AAP. In the case of a connection not making use of its allocated window, the buffer occupancy will start to go down in the AAP, causing an increase in the window size signaled to all connections: This results in the active connections increasing their throughput, sharing the available bandwidth equally.

The difficult task in such an algorithm is to design the feedback function $f(B_e(t))$. The dynamics of the system depend heavily on this feedback function. The goal of the function is to provide all TCP connections with similar feedback, and as a result have them operate with equal windows. We now discuss the requirements and tradeoffs in the choice of the feedback function and describe a function that satisfies these requirements.

3.1 Choosing the feedback function

The window feedback is based solely on the amount of free buffer at the AAPs. This makes the system self-adaptive to the traffic load and the number of active connections. All connections receive similar feedback and as a result the buffer occupancy will reach an equilibrium state. For example, assuming that a known number of N connections are active and that all the connections have equal round-trip times, the system

converges to a state that satisfies the following equation

$$Q_i(t) = f(B - NQ_i(t)), \tag{3.2}$$

where $Q_i(t)$ is the amount of buffer occupied by connection i, which is also equal to the feedback given to that connection. In steady state and under the assumptions made, every connection will occupy the same amount of buffer. It should be noted that the assumption of equal round-trip delays is not necessary for the system to reach an equilibrium state.

In order for such a feedback control system to be able to bring the buffer occupancy in AAPs to equilibrium, it is required that the sources react to feedback messages and increase their current windows rather slowly. TCP already uses such mechanisms to control window increases, imposing a window increase of one packet for every acknowledgement received or for every RTT, depending on whether the connection is in the slow-start or congestion avoidance phase. When a connection receives an advertised window size less than its current window size, the former takes effect immediately.

Note that if all the connections are in the congestion avoidance phase, they increase their window by approximately one segment every round-trip time and therefore, the buffer occupancy can be expected to grow rather slowly. However, it is possible that the window sizes for some of the connections sharing the buffer are below the slow-start threshold *ssthresh*, and are therefore growing exponentially in time. Since the connections in slow-start phase cannot be identified at the AAP without maintaining per-connection state, we must design a feedback function that does not penalize connections in the congestion-avoidance phase over those in slow-start.

One approach is to estimate the target window size as a *linear* function of the instantaneous free space at the AAP buffer. That is, the window for each connection is set to the available buffer space multiplied by a fraction α.

$$f(B_e(t)) = \alpha B_e(t) = \alpha(B - Q(t)). \tag{3.3}$$

The fraction α determines the the buffer occupancy in steady state. From simulations, we observed that such a function performs well and is able to reach a stable state when the TCP sources are physically close to the AAP, that is when the delays in the control loop are small. With larger feedback delays, however, especially when all the connections are in the congestion avoidance phase, such a function may cause substantial over-correction. To illustrate this, consider an example where N TCP sessions with equal round-trip times are active, with all of them in the congestion avoidance phase. Let the parameter α be 1. The behavior of buffer occupancy at the AAP is graphically illustrated in Figure 13. As long as the total buffer occupancy is below a target setpoint, all the connections are allowed to increase their windows by one segment. Thus, the overall buffer occupancy increases by N segments every RTT. In Figure 13 this is the case for intervals (t_0, t_1), (t_1, t_2), and (t_2, t_3). Notice that because of the feedback delay, the new window feedback becomes effective one round-trip time later. At time t_2 the buffer occupancy exceeds the theoretical setpoint and all the connections are requested to decrease their window. The amount of decrease depends on the exact time that the ack from an individual connection is processed at the AAP. In the worst case, each TCP connection will be asked to reduce its window by N segments, since at time t_3 the overall buffer occupancy exceeds the target setpoint by N. In the next RTT, that is during the interval (t_3, t_4), all N connections will decrease their windows by N segments, and the buffer occupancy at the AAP may go down by as many as $O(N^2)$ segments. These large oscillations in the windows can cause frequent underflows in the AAP, resulting in under-utilization of the available capacity in the ATM pipe.

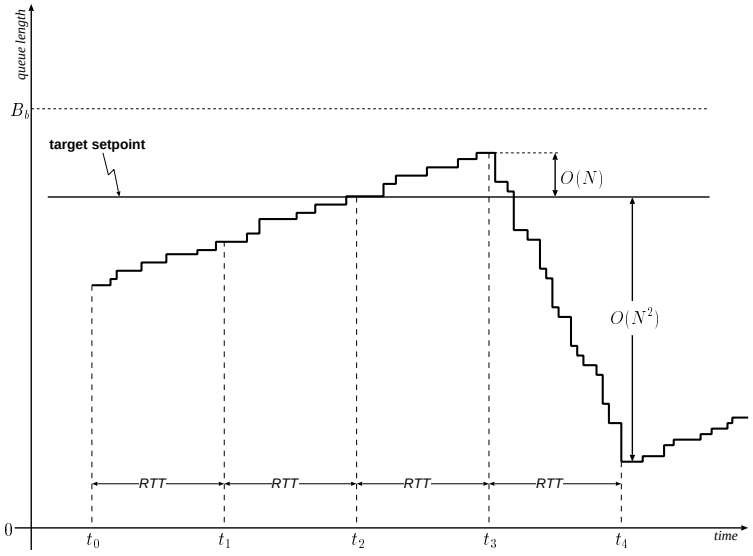

Figure 13: Potential queue behavior when computing feedback using a linear function.

A method of managing the buffer allocation dynamically without resulting in large over-correction is needed. The computed feedback should change rather slowly compared to the actual amount of empty buffer in order to the system to reach a steady state in the general case. We use a *logarithmic* function to compute the feedback sent to individual TCP sessions. That is,

$$f(B_e(t)) = \alpha \log_2 B_e(t) = \alpha \log_2 (B - Q(t)). \tag{3.4}$$

The total buffer size B and the buffer occupancy $Q(t)$ are expressed in terms of number of packets rather than bytes.

The motivation for a logarithmic function for estimation of the target window size is based on an understanding of the difference in behavior of TCP when it is in congestion avoidance phase and in slow start phase. We would like a fair allocation of the buffer to all connections sharing the buffer and, subsequently, the bandwidth in the ATM network. A TCP connection in slow-start phase doubles its window every round-trip time in contrast to a connection in congestion avoidance phase, where it increases only by one every RTT. The slow-start phase is to allow a TCP connection to rapidly ramp up to the point where the round-trip pipe is filled. Any further increases in the window would result in a gradual build-up of the queue at the bottleneck. Once the round-trip pipe is full, we would like the queue occupancy to increase at most linearly with time so that large oscillations in window size can be avoided. By using a logarithmic function to compute the explicit window feedback, we force all TCP connections to behave as if they are in the congestion avoidance phase. To illustrate this point, consider the case where all TCP connections are in the slow-start phase. The buffer occupancy can be expected to increase exponentially in this case, causing the free buffer also to decrease exponentially. The feedback values signaled to each connection will now decrease linearly, since they are the logarithm of an exponentially varying signal. The behavior when the buffer occupancy drops will be similar. Notice that if some of the active TCP connections are in congestion avoidance phase, their window increase process will be unaffected as long as their current window size is smaller than that indicated in the most recently received ack.

This logarithmic feedback function allows us to allocate the free buffer among the TCP connections, incrementally, on encountering an ack from each connection. This does not require maintaining any connection-level state. Although this scheme has similarities with buffer allocation in credit-based flow control [26, 27], our goal is not to achieve loss-free operation, but to reduce losses dramatically, while achieving fairness. Gross over-allocation is avoided because of the incremental allocation of the buffer upon each ack received, so that the feedback returned is a function of the current free buffer.

Using this function, rapid changes in the buffer occupancy will cause relatively small changes to the actual feedback sent to the TCP connections and therefore, the buffer occupancy manages to reach the steady state independent of the phase that TCP connections operate in. In Section 4 we evaluate extensively with simulations the system performance when the feedback is computed using the logarithmic function given by Eq. (3.4).

The feedback computed for each TCP connection when using the logarithmic function can be significantly smaller than that computed with the linear function. This may cause under-utilization when only a small number of TCP connections is active: the sum of the windows of the TCP connections may not be able to fill the network round-trip pipe. In the following section we describe a simple adaptive method that adjusts parameter α to the offered load so that performance is maximized even when a single TCP connection is active.

3.2 Adapting α to the bandwidth-delay product and buffer size

We would like the feedback computed by EWA to affect the source windows only after the round-trip pipe becomes full. However, when α is small, it is possible that the feedback computed by the logarithmic function is small. As a result, if only a small number of TCP connections are active, the bottleneck link will be under-utilized. Also, a side-effect of using the free buffer to compute the feedback is that the buffer occupancy in steady state increases with an increasing number of connections. To be able to bring the buffer occupancy to the desired setpoint, we make the scaling parameter α adaptive to the buffer state.

We use a simple method to adapt α. The method attempts to correct α only over long timescales, so that it will not affect the robustness and stability of the feedback computation process itself. The objective of EWA is to have the average queue length operate within a predefined range. The adaptive method that we propose keeps track of the average queue length. In our experiments the average queue size is computed using a first order low-pass filter. At a sampling point t, if $\overline{Q}(t-)$ is the average queue length and $Q(t)$ the the current queue length sample, then the average queue length is updated as

$$\overline{Q}(t) = (1-g)\overline{Q}(t-) + gQ(t),$$

where in our experiments the gain g is set to $1/128$.

Two thresholds are defined for $\overline{Q}(t)$, with the goal of maintaining the buffer occupancy between these thresholds. If the average buffer occupancy is below the low threshold, then, α is increased by a small constant quantity w_{up} every T seconds, that is

$$\alpha \leftarrow \alpha + w_{up}.$$

If the average buffer occupancy exceeds the high threshold, then α is reduced multiplicatively every T seconds as follows:

$$\alpha \leftarrow \alpha \cdot w_{down}.$$

The additive increases allow the system to slowly increase the computed feedback so that eventually the buffer occupancy will be within the desired operating range. On the other hand, in the event of sudden queue build up (possibly due to a reduction in the available bandwidth in the ATM segment) the multiplicative decreases will enable the buffer occupancy to be brought down rapidly. Thus, the combination of additive increases and multiplicative decreases allows the system to search for a value that will bring the buffer occupancy within the desired range. It is important to note here that, because of the hysteresis introduced by the two buffer thresholds, there is a range of values for α that can bring the buffer occupancy within the specified region. As a result, the robustness and stability of the system can be expected to show little sensitivity to the setting of the parameters used for adapting α.

The high buffer threshold is used to control the maximum buffer occupancy in steady state. Therefore, the setting of this threshold determines the ability of the system to accommodate bursty traffic without causing an overflow. Similarly, the low threshold allows the the system to tolerate short breaks in incoming traffic without causing an underflow. In our experiments we set the high and the low thresholds to 60% and 20%, respectively, which we found to be good choices to achieve these goals. Notice that the dynamic behavior of EWA will be fundamentally the same for other choices as well. We set the parameters w_{up} and w_{down} used to correct α to 1/8 and (1-1/32), respectively. The update interval T was chosen as 10 milliseconds. These choices make the adaptation process sufficiently slow so that the system behavior will be determined mostly by the EWA feedback rather than by the method used to adapt α. In all our simulation experiments, we used an initial value of 1 for α,

Adapting the value of α based on the buffer occupancy solves the problem of under-utilization when only a small number of connections is active and the round-trip delay is long: if the value of α is quite small so that the sum of the feedback sent to the connections sharing the buffer is less than the bandwidth-delay product, the average occupancy will drop below the low threshold. As a result, α will increase until the buffer occupancy is within the desired range. Similarly, when a large number of TCP connections becomes active, the buffer occupancy may increase beyond the high threshold and consequently the value of α will drop, bringing the steady state buffer occupancy within the desired range.

In the following section we use results from extensive simulation experiments to show that the combination of the logarithmic feedback function and adaptive algorithm for adjusting α brings the buffer occupancy within the desired range and achieves high network utilization and fairness.

4 Performance Evaluation

Having described the details of the Explicit Window Adaptation scheme, we now turn to evaluating its performance using simulation. The topology used in all our simulations is the one shown in Figure 1, where all the network links are assigned a bandwidth capacity of 155 Mbits/sec. We test the performance of the scheme for a large range of RTTs — from 11 to 100 milliseconds. We tested the scheme not only for the simple case of constant available rate in the ATM segment, but also when the available rate changes over a wide range. In addition to persistent TCP sources, we also investigated the performance of the scheme under on-off sources where feedback is often less effective. The total buffer size in the AAPs is a parameter in the simulations, but was always set not to exceed the bandwidth-delay product of the network.

Since the feedback loop in our scheme extends only from the AAP to the sources of the TCP connections, the response of the algorithm is determined by the delay in the local IP network rather than the total RTT of the connections. Fortunately, we can expect the delay in the ATM segment to dominate the RTT of the connection in an environment where IP-based local networks are interconnected through a wide-area ATM network segment. The difficult case for EWA, however, occurs when the delay in the IP part dominates the total RTT of the connection. To test the performance of the scheme in such a configuration, we also designed simulation experiments where the IP delay is as large as 49 milliseconds and the delay in the ATM segment is only 1 millisecond. Although such a combination is unlikely in the environment we consider, it serves as a worst-case scenario for testing the effectiveness of the scheme.

We used the OPNET modeling tool for all the simulations. The TCP model is based on the Reno version. It supports the congestion control mechanism described by Jacobson [1], exponential back-off, enhanced round-trip time (RTT) estimation based on both the mean and the variance of the measured RTT, and the fast retransmit and fast recovery mechanisms. However, some adjustments had to be made to the TCP timers; since the RTT values in some of our simulations are of the order of just a few milliseconds, we set the granularity of the timers to 20 milliseconds. We used a constant TCP segment size of 1500 bytes in all the simulations. Unless stated otherwise, the throughput measurements are performed in intervals of 250 milliseconds.

The metrics we use to evaluate the scheme are the link utilization, fairness, and buffer occupancy at AAP-1. In the specific network configuration considered, the utilization of the ATM link L1 (which is the bottleneck link) serves as an indication of the overall network performance. In most of our simulation experiments, the entire capacity of this link is available to the TCP connections under observations, thus simulating a steady rate in the ATM segment. However, we also conducted experiments where the capacity of the link L1 was shared by other on-off traffic during the course of the simulation, forcing the available rate to vary. We demonstrate the ability of the scheme to achieve fairness by plotting the growth of sequence numbers for each simulated TCP connection. Finally, the buffer occupancy plots serve to demonstrate the convergence dynamics of the algorithm.

4.1 Performance with small IP delays

We begin the evaluation of the scheme by first using the traffic scenario used in Section 2. In this scenario, a total of ten connections open simultaneously at time $t = 0$ and remain active for 10 seconds. At time $t = 10$ seconds five of the connections (6–10) close. The remaining connections stay active for another 5 seconds, when three more connections (3–5) close. The last two connections remain active until the end of the simulation at time t=20 seconds.

The one-way network delay in the IP datagram network is set to 0.5 millisecond, and the one-way delay of the ATM backbone to 5 milliseconds, giving rise to 11 milliseconds of round-trip delay. The feedback delay for the EWA scheme is 1 millisecond. The buffer size at AAP-1 is set to 200 Kbytes which is approximately the bandwidth-delay product of the network. Note that these network parameters are identical to those used for comparing the Drop-Tail, Drop-from-Front, and the RED approaches in Section 2. The maximum window size is set to 200 Kbytes, so that a single connection will be able to utilize the available bandwidth. The `ssthresh` variable of TCP is initialized to 64 Kbytes.

The sequence number growths of the TCP connections are plotted in Figure 14. Observe that the progress of all TCP connections is fair. When the number of active connections decreases at time $t = 10$

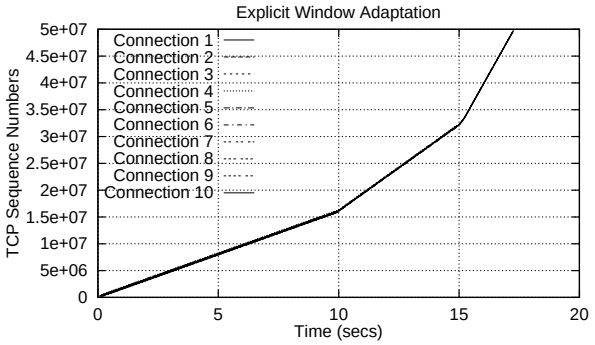

Figure 14: Sequence number growth for TCP connections with EWA (IP Network 1 delays = 0.5 msecs, ATM backbone delay = 5 msecs).

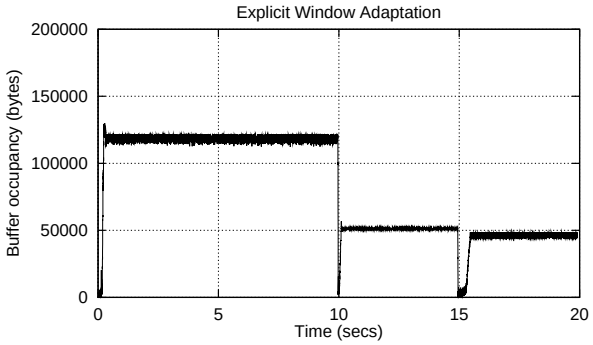

Figure 15: Buffer size at IP-ATM Router 1 with EWA (IP Network 1 delays = 0.5 msecs, ATM backbone delay = 5 msecs).

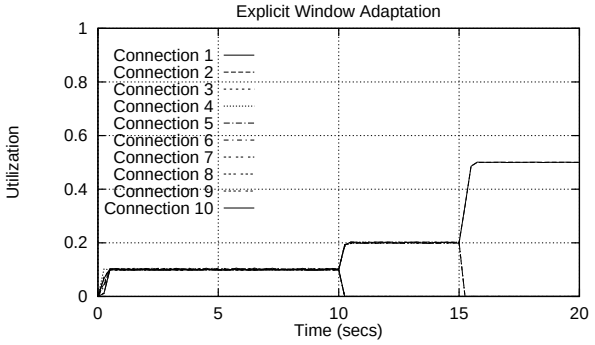

Figure 16: Utilization for each individual connection at link L1 with EWA measured in intervals of 250 msecs (IP Network 1 delays = 0.5 msecs, ATM backbone delay = 5 msecs).

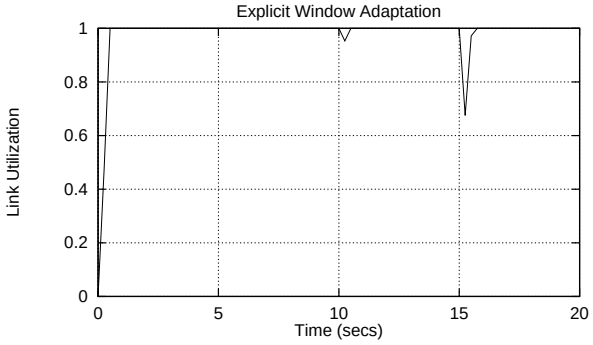

Figure 17: Effective utilization of the ATM link at IP-ATM Router 1 with EWA measured in intervals of 250 msecs (IP Network 1 delays = 0.5 msecs, ATM backbone delay = 5 msecs).

and 15 seconds, the remaining active connections detect and recover the unused bandwidth rapidly while maintaining fairness. Observe in Figure 14 how the slope of the sequence number lines changes at times $t = 10$ and 15 seconds as a result of extra bandwidth becoming available for the active connections.

The amount of bandwidth utilized by each TCP connection is shown in Figure 16, normalized to the maximum throughput that can be achieved (for a 155 Mbits/sec link this is approximately 135 Mbits/sec because of the overhead of cell headers, RM cells, TCP/IP headers, etc.). In this figure, the throughput for each TCP connection is measured in intervals of 250 milliseconds. Observe that the throughput allocation is perfectly fair among the active connections. During the first 10 seconds, when ten TCP connections are active, each of them receives 1/10th of the available bandwidth. From time t=10 secs to t=15 secs, when five TCP connections are active, each receives 1/5th of the available bandwidth. Finally, when only two connections are active, each gets half of the available bandwidth.

Figure 15 illustrates the buffer occupancy at AAP-1. As can be seen in this figure, the buffer never underflows in steady state, leading to perfect link utilization. This is verified by Figure 17 where the measured link utilization is shown. Except for brief periods when the number of active connections changes, the utilization of link L1 remains 100%. The reason for the underflows at time 10 and 15 seconds can be

explained as follows: When the number of active connections decreases, more bandwidth becomes available to each of the remaining active connections. However, the sum of the windows of the active TCP connections is not large enough to fill the round-trip network pipe. As a result, it will take a few RTTs before the windows of the remaining active TCP connections grow to a size that can fill the network links. Note that the rate at which the window of a given TCP connection grows decreases significantly if the connection is operating in the congestion avoidance phase (window size above the `ssthresh` value). This is indeed the case in this simulation experiment at time $t = 15$ when only two connections remain. The window size for the two active connections must increase from approximately 40 Kbytes up to 85 Kbytes to achieve full utilization of the network bandwidth. At the rate of one segment per RTT, this takes approximately 150 milliseconds, as can be verified from Figure 15. Thus, the interval over which the under-utilization occurs in this case is determined by the TCP window increase process rather than by the dynamics of our window adaptation scheme.

4.2 Performance with long IP delays

In this section we test the sensitivity of the explicit window adaptation method to the IP network delays. We study the behavior of the proposed method in the case where most of the network delays are in the IP part. This kind of configurations is the hardest for our scheme since the delays of the feedback loop are determined by the IP delays (assuming that the network pipe in the ATM part is full).

We test the performance of the proposed scheme in the extreme case where the one-way IP network delay is 49 milliseconds and the total round-trip network delay is 100 milliseconds. The one-way delay in the ATM network is only 1 millisecond. The buffer size in AAP-1 is set to be equal to bandwidth-delay product of the network which is approximately 2 Mbytes. Because of the size of the network and in order to isolate the effects of slow window increases, due to the TCP congestion avoidance phase, on the system performance, we have set ssthresh to 1 Mbyte, so that all TCP connections will be able to increase rapidly their windows in the absence of packet losses. Therefore, the dynamics of the network will depend mostly on the explicit window adaptation scheme and more specifically on how fast the parameter α, which is used to compute the feedback sent to TCP sources, adapts to network changes rather than on the TCP window growth rate.

Figure 18 shows the sequence number growth for the active TCP connections. Observe that the progress is perfectly fair, even though the round-trip delay of the network is 100 milliseconds. Figure 20 illustrates exactly what portion of the available bandwidth is used by each TCP connection. The buffer occupancy at AAP-1 is shown in Figure 19. Observe that even in this extreme case where the total delay in the IP network is almost 100 milliseconds, and even though all active TCP connections increase their window exponentially, the explicit window adaptation scheme is able to bring the buffer occupancy to equilibrium very fast. When the number of active connections decreases, at times $t = 10$ secs and $t = 15$ secs, the buffer underflows which unavoidably leads to some link underutilization. Two factors affect the duration of the buffer underflow: the TCP window increase time and the time required for parameter α to bring the system in a state with non-zero buffer occupancy. Faster increase rate for α would help reduce the duration of link underutilization by increasing the risk of packet losses. The observed underutilization, however, is not expected to be a problem in actual networks mainly for two reasons: (i) the number of active connections will be much larger than two, and (ii), the ssthresh variable in TCP will not be set to something even close to 1 Mbyte. In current TCP implementations the initial ssthresh value is in the range of 16 Kbytes-64 Kbytes. Therefore, the duration of link underutilization will be dominated by the time needed by individual TCP connections to increase their

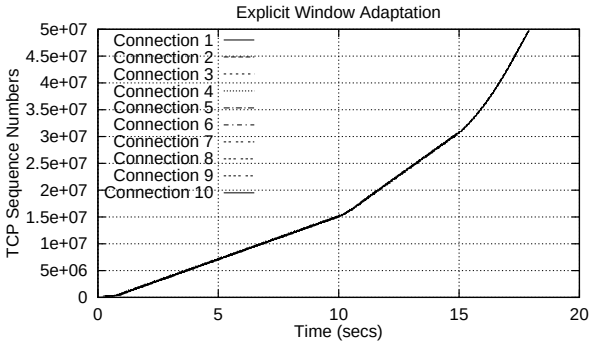

Figure 18: Sequence number growth for TCP connections with EWA (IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

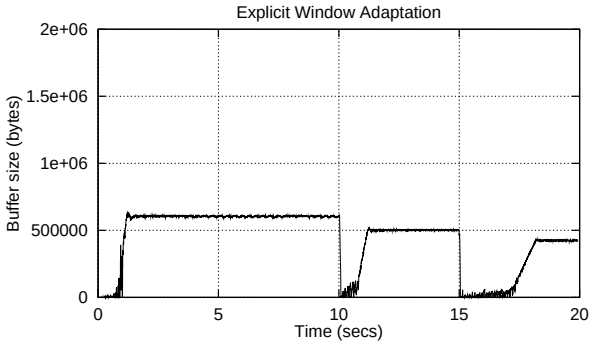

Figure 19: Buffer size at IP-ATM Router 1 with EWA (IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

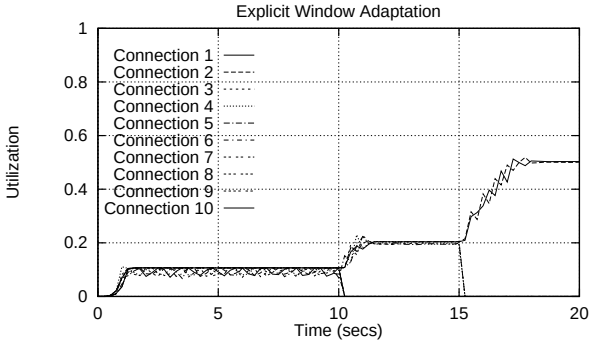

Figure 20: Utilization for each individual connection at link L1 with EWA measured in intervals of 250 msecs (IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

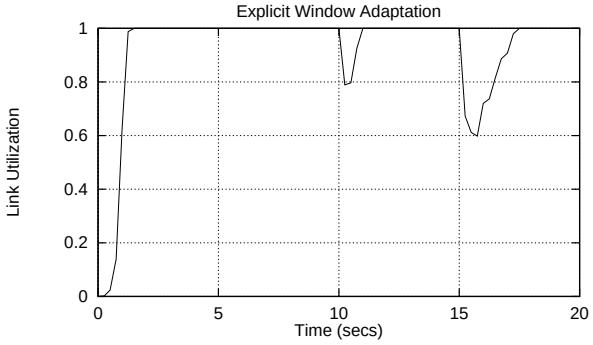

Figure 21: Effective utilization of the ATM link at IP-ATM Router 1 with EWA measured in intervals of 250 msecs (IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

windows. For example, in the scenario considered, if the ssthresh variable was set to 512 Kbytes instead of 1 Mbyte (which is again unrealistically large), TCP would need approximately 20-30 seconds before it was able to achieve 100% utilization after time $t = 15$ seconds when only two TCP sessions remain active compared to approximately 2 seconds needed with the proposed method. In any case, even during the period of underflow and underutilization, the performance of the active connections constantly improves and the progress is fair. Figure 21 shows for the test case considered, that the total link utilization never drops below 60%.

4.3 Performance with small buffers and unequal IP delays

We now show that Explicit Window Adaptation can be equally efficient and stable even when the available buffer in AAP-1 is much smaller than the bandwidth-delay product of the network and when not all TCP connections have equal delays in the local IP network segment.

The fairness and efficiency of packet discard schemes such as Drop-Tail, Drop-from-Front, and RED degrade dramatically when the amount of buffer at the AAP is less than the round-trip bandwidth-delay

product. Lakshman and Madhow [7] showed that the amount of buffer in Drop-Tail switches should be at least two to three times the bandwidth-delay product of the network in order for TCP to achieve decent performance and to avoid losses when operating in the slow-start phase. Also, Lakshman, et. al. [5] demonstrated that the buffering needed by the Drop-from-Front scheme must be of the order of the bandwidth-delay product for the network to achieve good performance.

For this experiment we use the same network topology and traffic scenario from the previous experiment. The one-way delay in the ATM backbone is set to 5 milliseconds. The delays in the IP network vary from 1 to 5 milliseconds for the individual connections. The one-way IP delay is 1 millisecond for connections 1 and 6, 2 milliseconds for 2 and 7, and increasing so on to 5 milliseconds for connections 5 and 10. According to this setup the total round-trip delays varies from 12 to 20 milliseconds for the individual connections. The bandwidth-delay product for the longest connection is approximately 390 Kbytes. To test the ability of our scheme to work with small buffers, we set the buffer capacity to 200 Kbytes, approximately half of this bandwidth-delay product.

The sequence number growth for the TCP connections is shown in Figure 22. The progress for all the active connections is steady and fair. Fairness is achieved in terms of having all TCP connections operate with equal transmission windows, since AAP-1 computes similar feedback for each connection. The throughput achieved by each connection is shown in Figure 24. Note the small variations in the utilization of individual connections because of the differences in round-trip delays. This is because of the well-known effect of TCP favoring connections with smaller round-trip times [28]. Achieving equal throughputs under asymmetric RTTs requires bandwidth allocation and scheduling at the level of individual TCP flows.

The buffer occupancy in AAP-1 is shown in Figure 23. The buffer occupancy is able to reach its equilibrium state rapidly, even though its total capacity is limited to half the bandwidth-delay product of the network and the active TCP sessions have unequal round-trip delays. The duration of the buffer underflow and the resulting link underutilization when the number of active connections decreases are both small to be of concern.

4.4 Performance with increasing number of TCP connections

In the results presented so far, the number of active TCP connections was made to decrease over the course of the simulation, and consequently the available bandwidth for the remaining connections increases. In this section we consider the case where new TCP connections join already active ones during the simulation. An effective buffer management scheme must be able to cause the existing TCP connections to withdraw part of their buffer and bandwidth allocations quickly in order to accommodate traffic sent by the new connections.

We have modified slightly the traffic scenario we used in the preceding sections in order to illustrate the ability of the scheme to accommodate new connections. At time $t = 0$ we activate five TCP sessions (1–5). At time $t = 10$ seconds, we open five more TCP sessions (6–10), thus, changing the total number of active TCP connection from five to ten. Finally, at time $t = 15$ seconds we close eight of the ten active connections (3–10). All connections have equal round-trip delays. Both the IP and the ATM network delays are set to 5 milliseconds, resulting in a total round-trip delay of 20 milliseconds. The buffer size at AAP-1 is set equal to the bandwidth-delay product, which is approximately 390 Kbytes.

The simulation results for this experiment are shown in Figures 26, 27, 28, and 29. The sequence number growth for the TCP connections is shown in Figure 26. During the first 10 seconds, when there are five connections active, the progress for all of them is fair and steady. As shown in Figure 28 each connection

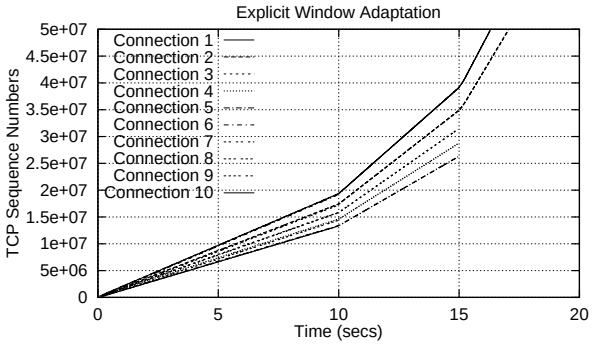

Figure 22: Sequence number growth for TCP connections with EWA (IP Network 1 delays vary from 1 to 5 msecs, ATM backbone delay = 5 msecs).

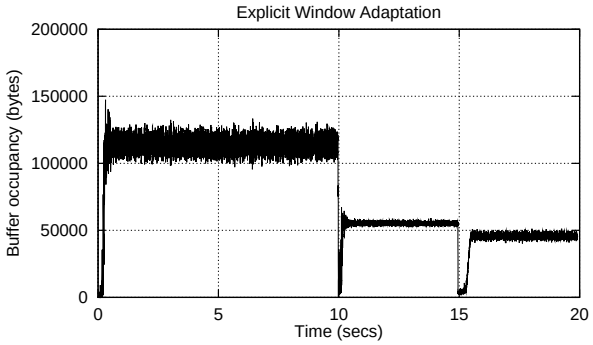

Figure 23: Buffer size at IP-ATM Router 1 with EWA (IP Network 1 delays vary from 1 to 5 msecs, ATM backbone delay = 5 msecs).

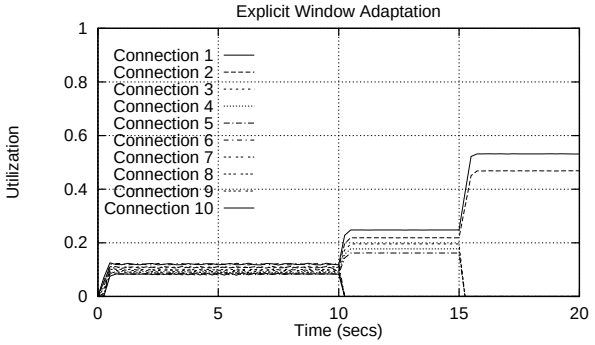

Figure 24: Utilization for each individual connection at link L1 with EWA measured in intervals of 250 msecs (IP Network 1 delays vary from 1 to 5 msecs, ATM backbone delay = 5 msecs).

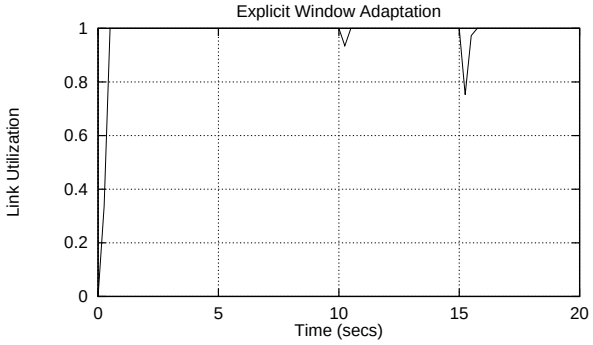

Figure 25: Effective utilization of the ATM link at IP-ATM Router 1 with EWA measured in intervals of 250 msecs (IP Network 1 delays vary from 1 to 5 msecs, ATM backbone delay = 5 msecs).

acquires exactly 20% of the available bandwidth. Figure 27 shows that the buffer occupancy reaches its steady state quickly and remains at that level as long as the number of TCP connections remains the same.

When the five new connections open at time $t = 10$ seconds, the existing connections release quickly some of the available bandwidth. This is evident from Figure 26 by the change in the slope of the sequence number growth lines. Notice that the initial sequence number for the new TCP connections is 2,780,000. The newly opened TCP connections manage to utilize quickly a fair portion of the available bandwidth. This fact is more clearly illustrated in Figures 28 where we can see that right after time $t = 10$ secs, the explicit window adaptation scheme quickly revokes part of the bandwidth, currently in use by the pre-existing TCP connections, to accommodate the new ones. Within approximately half a second all ten connection have reached their steady state and achieved equal thoughputs. It is interesting also to observe in Figure 27 that, after the number of active TCP connections increases from five to ten, the buffer reaches it new equilibrium state without dropping any packets. The behavior of the scheme when the number of active connections decreases is fundamentally identical to that observed and analyzed in the previous experiments.

The results presented in this section suggest that the EWA scheme is capable of accommodating increasing number TCP connection while achieving fairness, high utilization and stable behavior: the existing connec-

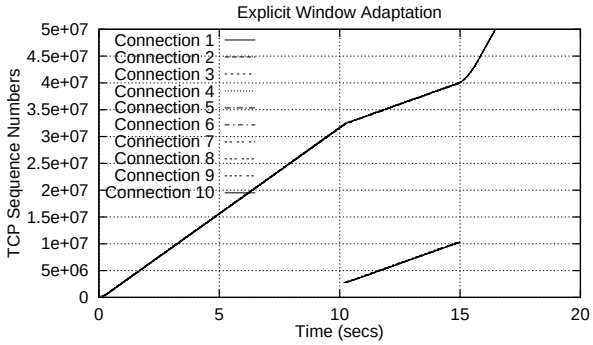

Figure 26: Sequence number growth for TCP connections with EWA (IP Network 1 delays = 5 msecs, ATM backbone delay = 5 msecs).

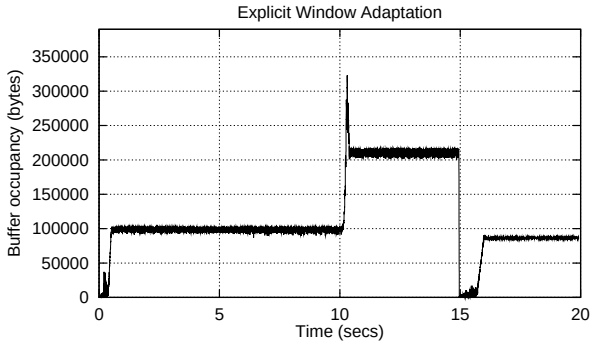

Figure 27: Buffer size at IP-ATM Router 1 with EWA (IP Network 1 delays = 5 msecs, ATM backbone delay = 5 msecs).

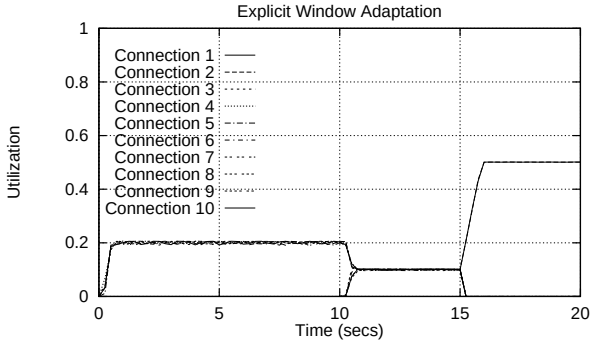

Figure 28: Utilization for each individual connection at link L1 with EWA measured in intervals of 250 msecs (IP Network 1 delays = 5 msecs, ATM backbone delay = 5 msecs).

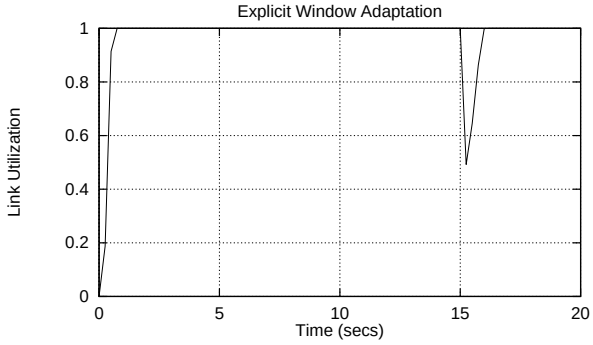

Figure 29: Effective utilization of the ATM link at IP-ATM Router 1 with EWA measured in intervals of 250 msecs (IP Network 1 delays = 5 msecs, ATM backbone delay = 5 msecs).

tions withdraw part of their allocations to allow a quick ramp-up for the new ones. Furthermore, during the transient time the scheme does not show any bias for either the existing or the new TCP connections.

4.5 Performance with varying bandwidth

In all the experiments so far we assumed that the available bandwidth in the ATM segment was constant. We now explore how changes in the bandwidth available to the aggregate TCP traffic affect the behavior and the performance of the TCP connections. We vary the bandwidth available to TCP traffic by opening and closing a high-priority ATM connection that shares the same link L1 with the aggregate TCP traffic. The amount of bandwidth reserved for this high-priority connection is a simulation parameter. Although we have studied the behavior of the scheme for both low- and high-frequency changes in the TCP bandwidth, due to space limitations we will present results only for the case where the available bandwidth changes with high frequency. For this experiment all ten TCP connections open at time $t = 0$ and remain open for the duration of the simulation, lasting 20 seconds.

We model these high-frequency changes in available bandwidth by opening and closing the high priority

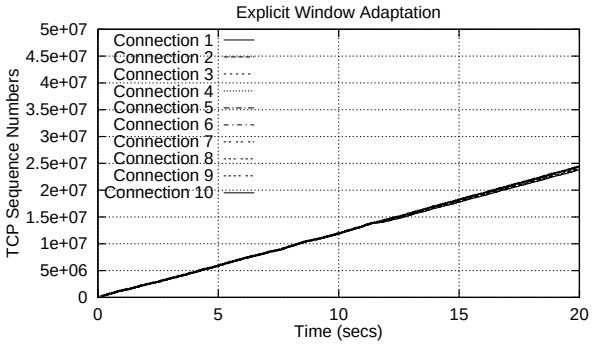

Figure 30: Sequence number growth for TCP connections with EWA when the available bandwidth varies from 80 Mbits/sec to 155 Mbits/sec (IP Network 1 delays = 5 msecs, ATM backbone delay = 5 msecs).

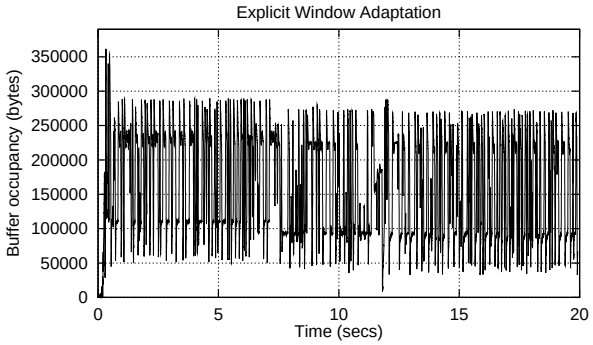

Figure 31: Buffer size at IP-ATM Router 1 with EWA when the available bandwidth varies from 80 Mbits/sec to 155 Mbits/sec (IP Network 1 delays = 5 msecs, ATM backbone delay = 5 msecs).

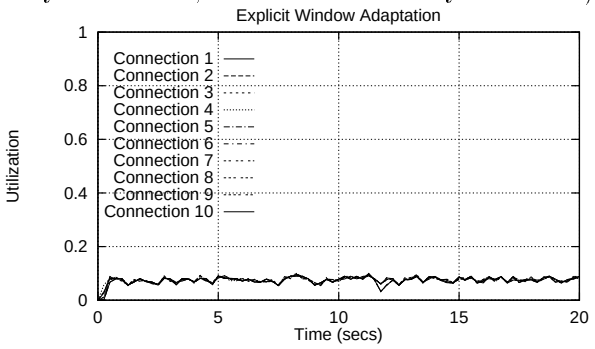

Figure 32: Utilization for each individual connection at link L1 with EWA measured in intervals of 250 msecs when the available bandwidth varies from 80 Mbits/sec to 155 Mbits/sec (IP Network 1 delays = 5 msecs, ATM backbone delay = 5 msecs).

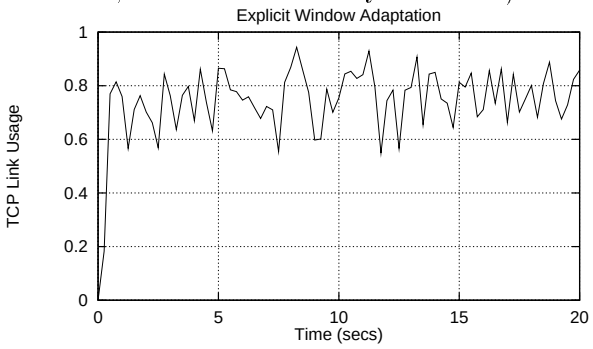

Figure 33: Effective utilization of the ATM link at IP-ATM Router 1 with EWA measured in intervals of 250 msecs when the available bandwidth varies from 80 Mbits/sec to 155 Mbits/sec (IP Network 1 delays = 5 msecs, ATM backbone delay = 5 msecs).

connection using an ON/OFF model where the duration of the ON and OFF periods is exponentially distributed with a mean of 50 milliseconds. When in the ON state, the high priority connection is allocated 75 Mbits/sec. Therefore, the bandwidth variability is of the order of 50% of the link capacity. Both the IP and the ATM delays are set to 5 milliseconds and the buffer size is set to 390 Kbytes, equal to the bandwidth-delay product of the network.

The simulation results for this experiment are presented in Figures 30, 31, 32, and 33. The progress of all ten TCP connections is fair and the EWA mechanism successfully controls the buffer occupancy: there are no buffer overflows or underflows. However, careful examination of the simulation results revealed that the high variability in the queueing delays triggered packet retransmissions for some TCP connections. This is because the TCP timeout estimation algorithm was not able to adapt to large variations in the round-trip delays fast enough.† However, these retransmissions have only a minor effect on the progress of the

†Larger granularities for the TCP timers would have prevented the retransmissions in this particular case.

corresponding TCP connections, as can be seen in Figures 30 and 32. Since the buffer never underflows, we conclude that the total link utilization is 100%. Also, since the high-priority connection is active 50% of the time and the active connections use approximately 48% of total link capacity, we can expect that, on the average, 76% of the link bandwidth will be available for transporting TCP traffic. This expectation is verified by the results in Figure 33.

4.6 Performance with large number of connections and non-persistent sources

So far in our experiments, the TCP connections were allowed to transmit long enough to reach steady state. Next, we consider the case of non-persistent TCP connections where the connections do not have data to transmit at all times. In this experiment, we simulate a mix of greedy and non-persistent connections to study their interaction and the resulting dynamics when the EWA scheme is used to control the TCP windows.

We increase the total number of TCP connections to forty. Ten of them are driven by persistent (greedy) sources. The remaining thirty connections are driven by ON-OFF sources where the ON and OFF periods are exponentially distributed, each with an average of 200 milliseconds. Eight connections originate from each of the five source nodes in Figure 1. Six of them carry traffic from ON-OFF sessions and the remaining two carry traffic from persistent sessions. The one-way delay in the IP is set to 49 milliseconds and that in the ATM backbone to 1 millisecond, giving rise to 100 milliseconds of round-trip delay for each connection. Thus, the ON and OFF periods are on the average small multiples of the RTT. It is important to emphasize here that all the TCP connections remain open for the entire duration of the simulation. It is the applications that feed the corresponding TCP connections that follow the ON-OFF model. We set the buffer size in AAP-1 to be equal to the bandwidth-delay product of the network, which is approximately 2 Mbytes. The start time for each of the forty TCP connections is uniformly distributed from 0 to 5 milliseconds.

TCP-controlled ON-OFF sources can introduce significant amount of burstiness to the network: it is likely that during an OFF period of the connection, all of the outstanding data transmitted during the last ON period have been acknowledged. As a result, in the subsequent ON period the connection may transmit an entire window worth of data as a single burst. In addition, since the connection may not have received any feedback for some time before entering the ON period, it may operate with a larger window size than the current value computed by the window adaptation scheme, until the first feedback arrives from the AAP after one RTT. These two effects pose difficulties for the EWA scheme. Despite these problems, the results to be presented next reveal that EWA remains highly efficient and fair even in the presence of a large number of bursty connections.

Figure 34 shows the sequence number growth for one representative TCP connection from each of the five source nodes carrying ON-OFF traffic, and one TCP connection carrying traffic from a greedy source. Since the ON-OFF sources have a 50% duty cycle, we would expect the sequence number growth rate for ON-OFF sources to be approximately half of that of persistent ones. This is indeed the case, as seen in Figure 34. Deviations from this expected value are due to the statistical nature of the traffic model and because the connections do not open in a synchronized manner. It is important to note that the ON-OFF connections make fair progress and have nearly identical behavior.

The buffer occupancy at AAP-1 for this experiment is shown in Figure 35. Notice that EWA protects the buffer from overflows. In addition, the buffer underflows are brief and do not significantly affect the utilization of link L1. Therefore, we can conclude that EWA manages to control the buffer occupancy well

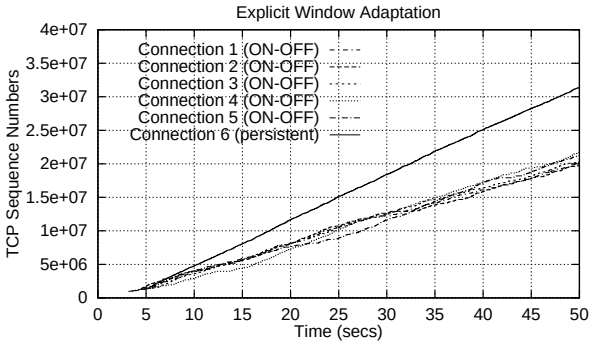

Figure 34: Sequence number growth for 5 ON-OFF TCP connections (one from each source node) and 1 persistent TCP connection with EWA (in total there are 10 persistent TCP sources, 30 ON-OFF TCP sources with mean ON-time = mean OFF-time = 200 ms, IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

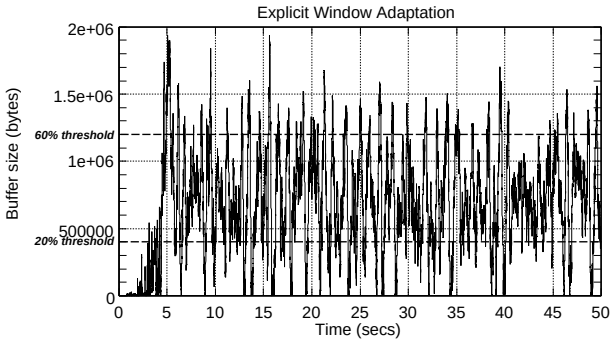

Figure 35: Buffer size at IP-ATM Router 1 with EWA (in total there are 10 persistent TCP sources, 30 ON-OFF TCP sources with mean ON-time = mean OFF-time = 200 ms, IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

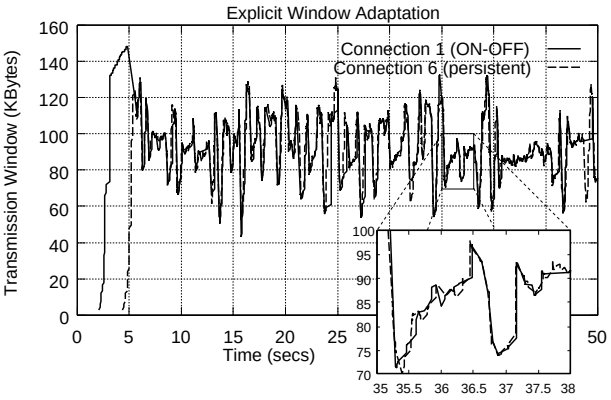

Figure 36: Congestion window size for one ON-OFF and one persistent TCP connection with EWA. (in total there are 10 persistent TCP sources, 30 ON-OFF TCP sources with mean ON-time = mean OFF-time = 200 ms, IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

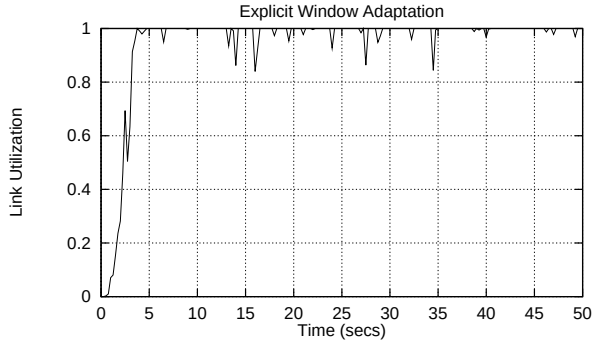

Figure 37: Effective utilization of the ATM link at IP-ATM Router 1 with EWA measured in intervals of 250 msecs (in total there are 10 persistent TCP sources, 30 ON-OFF TCP sources with mean ON-time = mean OFF-time = 200 ms, IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

in this experiment. The utilization of link L1, as can be seen in Figure 37, is perfect most of the time.

The TCP transmission windows for one ON-OFF and one persistent connection are shown in Figure 36 where we can see that both connections operate with equivalent transmission windows. Notice that the ON-OFF connection is not penalized for being idle. Therefore, during the ON period of the ON-OFF connection, the scheme is able to force the greedy source to withdraw rapidly the bandwidth and buffer allocations in order to accommodate traffic from the ON-OFF source.

For comparison, we performed the same experiment with packet discard using the RED algorithm in

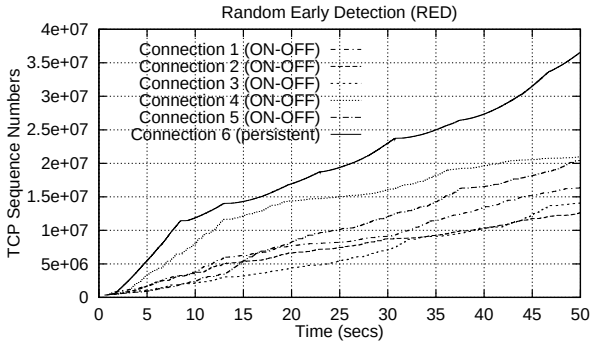

Figure 38: Sequence number growth for 5 ON-OFF TCP connections (one from each source node) and 1 persistent TCP connection with RED (in total there are 10 persistent TCP sources, 30 ON-OFF TCP sources with mean ON-time = mean OFF-time = 200 ms, $max_p = 0.05$, $min_{th} = 150$, $max_{th} = 400$, IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

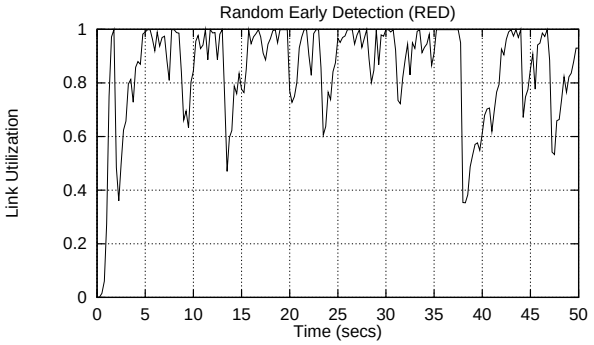

Figure 39: Effective utilization of the ATM link at IP-ATM Router 1 with RED measured in intervals of 250 msecs (in total there are 10 persistent TCP sources, 30 ON-OFF TCP sources with mean ON-time = mean OFF-time = 200 ms, $max_p = 0.05$, $min_{th} = 150$, $max_{th} = 400$, IP Network 1 delays = 49 msecs, ATM backbone delay = 1 msecs).

place of EWA. Figures 38 and 39 summarize the simulation results. As we can see in the sequence number growth graph, the burstiness of the ON-OFF sources affect the performance of the persistent ones. Moreover, the overall progress is not as fair and steady as in the EWA scheme. It can be seen in Figure 38 that all the connections go through a large number of slow-starts, significantly affecting the efficiency and fairness of the network. As can be seen in Figure 39, depending solely on packet losses to detect congestion has an impact on the congested link's utilization which is significantly worse than that achieved by EWA.

5 Conclusion

The Explicit Window Adaptation scheme may be seen as an attempt to reconcile the inherent mismatch between window-based and rate-based congestion control approaches. This mismatch can cause large oscillations in buffer occupancy at the edge device (switch or router) connecting the rate-controlled segment to the rest of the internetwork. The EWA scheme overcomes this problem by explicitly controlling the end-to-end window size to correspond to the round-trip delay-bandwidth product. By careful design of the feedback function to set the window size, the scheme is able to achieve very low packet loss, a high degree of fairness, and almost perfect bandwidth utilization.

An important advantage of the EWA scheme is its simplicity. The scheme is able to adapt automatically to the number of active connections, the traffic load, the buffer size, and the bandwidth-delay product of the network without maintaining any per-connection state. In addition, the TCP implementations in the end systems do not need to be modified, nor does it require modifying the TCP protocol itself. Updating the window size in returning acks is done in a manner transparent to TCP. The processing performed by the AAP on each TCP acknowledgement consists of updating the *receiver's advertised window* and the *checksum* fields. Note that checksum can be updated from knowing only the previous and new values of the advertised

window field, as well as the old checksum value.

Results from our simulation experiments demonstrate that the EWA scheme operates well under a wide range of traffic scenarios. These experiments were not just "toy" simulations, but designed to represent the traffic scenarios in a real network. In addition, we have also compared the results from our scheme with those from using intelligent packet discard policies such as Drop-from-Front and RED. Both the Drop-from-Front and RED policies have exhibited sensitivity to the specific traffic scenario used. For example the dynamics of TCP change when the number of active connections change. The performance of RED can be tuned by adjusting its parameters but such a process requires precise knowledge of the network topology and traffic load. Adjusting RED's parameters to optimize network performance is an open issue and requires further investigation. The EWA scheme achieves fairness in steady state by signaling the same window size to all the TCP connections. It can also operate with buffer sizes smaller than the round-trip delay-bandwidth product of the network. In addition, by adapting the parameter α to control the long-term buffer occupancy, the scheme is able to work well with both small and large number of connections sharing the buffer.

Another important advantage of EWA is that it can accommodate non-responsive or ill-behaved traffic. UDP traffic is an example of such traffic. Our scheme can be extended in ways to isolate the TCP traffic from UDP traffic. One approach is to simply to keep track of the amount of buffer occupied by each UDP connection and drop incoming packets when the connection's share of the buffer exceeds the target value computed by EWA. Although this approach offers fair usage of the available buffer space it requires per-connection accounting for UDP flows. Furthermore, such an approach may not be the best choice for NFS traffic: NFS often uses UDP datagrams as large as 64 Kbytes. If the allowed buffer occupancy for a UDP connection carrying NFS traffic is less than 64 Kbytes, the whole UDP packet will be dropped. A solution to this problem would be to reserve a minimum amount of the available buffer space for the overall UDP traffic. In case the feedback computed by EWA for a UDP flow exceeds its buffer usage, this additional space can be used accommodate large packets.

When multiple ATM virtual circuits share the same buffer at the AAP, the buffer must be partitioned across the VCs, and each partition must be controlled by a separate instance of EWA. A simple static method that partitions the buffer proportionally to the bandwidth allocated to each VC would be adequate. Such an approach is acceptable because the explicit window adaptation method is not sensitive to the exact amount of buffering available in a given partition. Clearly, the scheme can also be applied to the case when each TCP connection is carried over a separate VC.

The operation of the proposed method is not significantly affected by the TCP segment size. Even though it is known that TCP favors connections with large segments when the window is allowed to grow, fairness will be reached in steady state because all active connections will receive similar feedback independent of their current window or segment size.

Although in this paper we studied EWA in the limited context of an edge device, we believe that the scheme is applicable to more general environments to control congestion and improve fairness. In the future, we plan to explore its applications in more general network environments.

References

[1] V. Jacobson, "Congestion avoidance and control," in *Proceedings of ACM SIGCOMM'88*, pp. 314–329, 1988.

[2] V. Jacobson, "Modified TCP congestion avoidance algorithm." message to end2end-interest mailing list, April 1990.

[3] W. R. Stevens, *TCP/IP Illustrated*, vol. 1. Addison-Wesley Publishing Company, 1994.

[4] S. Floyd and V. Jacobson, "Random Early Detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[5] T. V. Lakshman, A. Neidhardt, and T. J. Ott, "The drop from front strategy in TCP and in TCP over ATM," in *Proc. of IEEE INFOCOM'96*, vol. 3, pp. 1242–50, March 1996.

[6] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *Request for Comments: 1323*, May 1992.

[7] T. V. Lakshman and U. Madhow, "Performance analysis of window-base flow control using TCP/IP: The effect of high bandwidth-delay products and random loss," in *Proc. of High Performance Networking, V. IFIP TC6/WG6.4 Fifth International Conference*, vol. C, pp. 135–149, June 1994.

[8] J. C. Mogul, "Observing TCP dynamics in real networks," in *Proceedings of ACM SIGCOMM'92*, pp. 305–317, August 1992.

[9] R. Wilder, K. K. Ramakrishnan, and A. Mankin, "Dynamics of congestion control and avoidance of two-way traffic in an OSI testbed," *ACM Computer Communication Review*, vol. 21, no. 2, pp. 43–58, April 1991.

[10] L. Zhang and D. D. Clark, "Oscillating behavior of network traffic: A case study simulation," *Intenetworking: Research and Experience*, vol. 1, no. 2, pp. 101–112, December 1990.

[11] 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 ACM SIGCOMM'91*, pp. 133–147, September 1991.

[12] A. Mankin, "Random drop congestion control," in *Proceedings of ACM SIGCOMM'90*, pp. 1–7, September 1990.

[13] A. Mankin and K. K. Ramakrishnan, "Gateway congestion control survey," *Request for Comments: 1254*, August 1991.

[14] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, pp. 115–156, September 1992.

[15] A. Romanow and S. Floyd, "Dynamics of TCP traffic over ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 633-41, no. 4, p. 13, May 1995.

[16] L. Kalampoukas and A. Varma, "Performance of TCP over multi-hop ATM networks: A comparative study of ATM layer congestion control schemes," in *Proceedings of ICC'95*, pp. 1472–1477, June 1995.

[17] B. J. Ewy, J. B. Evans, V. S. Frost, and G. J. Minden, "TCP/ATM experiences in the MAGIC testbed," in *Proc. of the Fourth IEEE International Symposium on High Performance Distributed Computing*, pp. 87–93, August 1995.

[18] S. Floyd, "TCP and explicit congestion notification," *Computer Communication Review*, vol. 24, no. 5, pp. 8–23, October 1994.

[19] Z. Wang and J. Crowcroft, "A new congestion control scheme: Slow start and search (Tri-S)," *Computer Communication Review*, vol. 21, no. 1, pp. 32–43, January 1991.

[20] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–80, October 1995.

[21] R. Jain, "Myths about congestion management in high-speed networks," *Internetworking: Research and Experience*, vol. 3, no. 3, pp. 101–113, September 1992.

[22] R. Cole, D. Shur, and C. Villamizar, "IP over ATM: A framework document," *Request for Comments (RFC): 1932*, April 1996.

[23] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Dynamics of an explicit rate allocation algorithm for ATM networks," in *Proc. of International Broadband Communications Conference'96*, IFIP-IEEE, April 1996.

[24] D. Mitra, "Asymptotically optimal design of congestion control for high speed data networks," *IEEE Transactions on Communications*, vol. 40, no. 2, pp. 301–311, February 1992.

[25] A. K. Choudhury and E. L. Hahne, "Dynamic queue length thresholds in a shared memory ATM switch," in *Proc. of IEEE INFOCOM'96*, vol. 2, pp. 679–87, March 1996.

[26] H. T. Kung, T. Blackwell, and A. Chapman, "Credit-Based flow control for ATM networks: Credit update protocol, adaptive credit allocation, and statistical multiplexing," in *Proceedings of ACM SIGCOMM'94*, pp. 101–114, September 1994.

[27] C. M. Ozveren, R. Simcoe, and G. Varghese, "Reliable and efficient hop-by-hop flow control," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 4, pp. 642–650, May 1995.

[28] S. Floyd, "Connections with multiple congested gateways in packet-switched networks, Part I: One-way traffic," *Computer Communication Review*, vol. 21, no. 5, pp. 30–47, October 1991.