Improving TCP Throughput over Two-Way Asymmetric Links: Analysis and Solutions

Lampros Kalampoukas*, Anujan Varma*
and
K. K. Ramakrishnan†

UCSC-CRL-97-20
August 21, 1997

* Board of Studies in Computer Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064

†AT&T Labs-Research
Florham Park, NJ 07932

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.

Abstract

We study several schemes for improving the performance of two-way TCP traffic over asymmetric links where the bandwidths in the two directions may differ substantially, possibly by many orders of magnitude. The sharing of a common buffer by data segments and acknowledgments in such an environment produces the effect of *ack compression*, often causing dramatic reductions in throughput. We first demonstrate the significance of the problem by means of measurements on an experimental network and then proceed to study approaches to improve the throughput of the connections. These approaches reduce the effect of ack compression by carefully controlling the flow of data packets and acknowledgments. We first examine a scheme where acknowledgments are transmitted at a higher priority than data. By analysis and simulation, we show that prioritizing acks can lead to starvation of the low-bandwidth connection.

The second approach makes use of a connection-level backpressure mechanism to limit the maximum amount of data buffered in the outgoing IP queue of the source of the low-bandwidth connection. This approach, while minimizing the queueing delay for acks, is shown to result in unfair bandwidth allocation on the slow link. In addition, the connection throughputs are highly sensitive to parameters such as packet and ack sizes in either direction. Finally, our preferred solution makes use of a connection-level bandwidth allocation mechanism. We show that this scheme overcomes the limitations of the previous approaches, provides isolation, and enables precise control of the connection throughputs. We present analytical models of the dynamic behavior of each of these approaches, derive closed-form expressions for the expected connection efficiencies in each case, and validate them with simulation results.

Keywords: TCP, two-way traffic, asymmetric links, backpressure

1 Introduction

The Transmission Control Protocol (TCP) has become the most widely used transport-layer protocol today, due largely to the explosive growth of the TCP/IP Internet in recent years. An important component of TCP is the collection of algorithms used to perform congestion control and recovery [1, 2]. These algorithms give rise to a variety of interesting dynamics, some of which have been studied extensively [3, 4, 5, 6]. In this paper, our interest is in analyzing the dynamics of TCP connections over asymmetric access links in the presence of two-way traffic.

We define *two-way* or *bidirectional* traffic as the traffic pattern resulting from two or more TCP connections transferring data in opposite directions between the same pair of end nodes over a network path. The TCP segments transmitted by the connections in one direction share the same physical path with the acknowledgments (acks) of connections in the opposite direction. These data segments and acknowledgments may share a common buffer in the end systems as well as network switches/routers. This sharing has been shown to result in an effect called *ack compression*, where acks of a connection arrive at the source bunched together [6, 7]. The result of ack-compression is a marked unfairness in the throughput received with competing connections, and reduced overall throughput compared to what could be expected without this effect [7]. Ack compression may occur either at the end system or in a switch/router. In either case, the smooth flow of acknowledgments to the source is disturbed, potentially resulting in reduced overall throughput.

While previous studies provide a qualitative treatment of the ack compression problem [6, 7], our objective in this paper is to analyze the dynamic behavior and quantify the throughput degradation of TCP connections in a two-way environment when the bandwidths of the links in the two directions differ significantly, possibly by several orders of magnitude.

Some of the reduced throughput in datagram networks under two-way traffic could be attributed to the bunching of acks at the bottleneck link, behind data packets. Since the acks typically take less time to process in the routers as compared to data packets, the former tend to become bunched as they travel through the network. However, even when routers and their links have adequate bandwidth, undesirable interaction between bidirectional connections can still occur in the end systems, leading to ack compression and throughput loss [8]. This is due to the sharing of a common queue by data packets and acknowledgments. With asymmetric link speeds, the effect of ack compression is more pronounced at the end system with the lower-speed upstream channel. With a non-preemptive scheduling policy in the end system, acknowledgments of the fast connection may be queued for a long time behind a data packet being transmitted on the slow link, causing a large number of acknowledgments to be bunched.

The genesis of ack compression can be traced to the slow-start phase of the TCP connection that increases the window progressively at startup [1]. The slow-start algorithm sets the initial window size to one and increases it by one with every acknowledgment received. This effectively doubles the window every round-trip time. Thus, during slow start, the receipt of every ack causes the end system to add two segments to its outgoing queue. Since the outgoing queue is usually maintained in FIFO order, these two segments must be transmitted before an ack to the connection in the opposite direction can be sent. In addition, when the acks to the two transmitted segments arrive after a round-trip delay, with no data segments in between, four data segments are transmitted in response, which also appear back-to-back. Meanwhile, the acks of the reverse connection are queued behind the data segments, causing them to be bunched. This behavior can persist in steady state when the windows reach their final values [8].

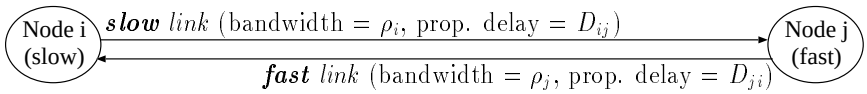


Figure 2.1: Network configuration.

This paper extends the analysis approach in [8] by considering the effect of asymmetric access links on the performance of two-way TCP traffic. Asymmetric link speeds are likely to be common in the future, with the widespread deployment of cable and high-speed DSL (Digital Subscriber Line) [9] access networks to homes and businesses. These access networks have substantially higher speed in one direction than the other. We demonstrate that the asymmetry in link bandwidth has a dramatic effect on the performance of the connection going through the faster down-link. We consider several solutions to the problem and study their effects on TCP performance. These solutions attempt to control the queueing delays of acks in the end system with the slow outgoing link. For each of the solutions, we develop detailed analytical models that capture the dynamics of the system and compute the expected throughput efficiency for each connection. The solutions include providing priority to acks over data packets queued at the end node; applying back-pressure to limit the number of data packets transmitted by an end node prior to transmitting an ack; and finally, a policy that provides connection-level bandwidth allocation and exploits knowledge of the asymmetric link speeds.

The remainder of the paper is organized as follows: In the next section we define our models of the network and the end nodes, briefly review the dynamics of two-way TCP connections in an asymmetric network, and quantify the resulting throughput degradation. To lend credence to our analytical results, in Section 3 we show evidence of the severe throughput degradation that can result in the two-way asymmetric environment by providing measurement data from an experimental cable access network. The next three sections evaluate various solutions to the ack compression problem in the asymmetric environment. Section 4 analyzes the effect of prioritizing acks, and Section 5 examines the backpressure scheme. In Section 6 we examine the benefits of the connection-level bandwidth allocation scheme. Finally, we conclude in Section 7 with some observations.

2 Network Models and Effects of Two-Way TCP Traffic

In this section, we describe our models of the network and the end nodes and review the results presented in [8] on the effects of two-way TCP traffic.

The basic configuration we consider consists of two nodes i and j communicating over a network that guarantees in-order packet delivery and provides a fixed-bandwidth pipe for the TCP connections between the end-nodes in each direction. Since our primary focus is on the performance of two-way TCP traffic over asymmetric access links (especially lower speed uplinks), we can model the path connecting the end-nodes (consisting of links and routers) in each direction by single point-to-point links.

The data packets transmitted by the TCP connection in one direction share a common FIFO queue with the acks of the opposite connection within the IP layer in the outbound direction. We refer to the TCP connection transferring data from node i to j as *connection* i and the connection in the opposite direction as *connection* j. We denote by D_{ij} the constant delay seen by TCP data packets transmitted from i to j, and by D_{ji} the delay seen by packets from j to i. We assume that

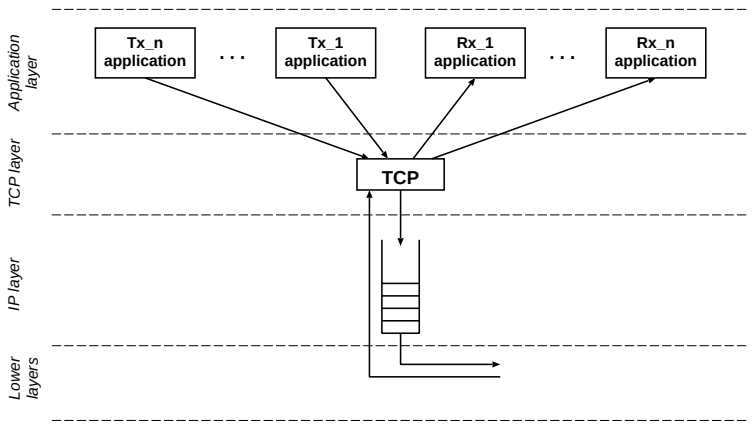


Figure 2.2: Model of an end-node used in the analysis.

the data packets transmitted by each connection are of constant size. The bandwidth capacities of the links originating at nodes i and j will be denoted by ρ_i and ρ_j, respectively, in units of TCP data packets per second. We assume that the link from j to i has larger capacity compared to that in the opposite direction, that is, $\rho_j > \rho_i$. For convenience, we often refer to connection i, transmitting data on the slow link, as the *slow connection* and its originating node i as the *slow node*. Likewise, we call the opposite connection j the *fast connection* and its originating node j as the *fast node*. We use t_i^d and t_i^a, respectively, to denote the transmission time of data packets and acks over the slow link; similarly, we use t_j^d and t_j^a to denote the transmission time of data packets and acks, respectively, over the fast link. Let α denote t_i^d/t_j^d, the ratio between the packet transmission times in the two directions. Similarly, let β denote the ratio of the packet transmission time to the ack transmission time on the *slow* link, that is, t_i^d/t_i^a.

The model of each end node is shown in Figure 2.2. A number of receiving and transmitting applications may reside within each node; sending applications generate data for TCP connections originating at that node and receiving ones act as the sinks for data arriving from the opposite node. Transmitting applications are assumed to be greedy and receiving ones are considered to be able to absorb all received data immediately.

A common process handles all TCP processing within the node. The TCP process receives data segments from the network and delivers them to the receiving application. In addition, an acknowledgment is generated and added to the outgoing queue for each segment received. The same TCP process also handles transmission of data to the opposite node by queueing one or more data segments from the transmitting application into the outgoing queue each time an ack is received from the opposite node. Finally, the TCP process is responsible for controlling the window growth of the sending TCP during the slow-start phase by incrementing it each time an ack is received, until the window reaches its maximum size. Since our analysis pertains to steady-state operation of the connections over an interval where we can assume the network is loss-free, we can ignore some of the details of the congestion control functions at the TCP layer. Differences between the slow start and the congestion avoidance phase and recovery on packet losses are not expected to dramatically change the effect of ack compression and the basic insights we gain here. This is because the initiation of ack compression is the important phase, which occurs at the slow start phase. The subsequent congestion avoidance phase does not break the ack compression phenomenon, once begun.

Segments of the forward connection and acks to the backward connection share a common FIFO

queue at the IP layer that is serviced at the transmission rate of the outgoing link. This common queue is key to the occurrence of ack compression where acks to the backward connection get bunched while waiting in the queue behind data packets of the forward connection (and vice versa), causing the TCP dynamics we study in this paper. We assume that the IP service rate is equal to transmission rate on the link. However, this is not critical if the acks of the backward connection and data packets of the forward connection share common outgoing queues all the way down the protocol stack. A higher service rate out of the IP queue reduces queueing at the IP layer, but the same bunching effect will now occur at the device driver or at the FIFO queue residing in the transmitter's data-link layer.

For the purpose of our analysis, the action of the TCP process can be summarized as follows: The process is invoked each time an ack arrives at a node, say node i. Processing of the ack results in a new data packet to be added to the IP queue for transmission. In addition, if connection i is in its slow-start phase, its window is increased by one and an additional packet is queued in the IP queue; this latter packet is queued immediately behind the first in the IP queue, resulting in the two packets being transmitted without an intervening ack for the opposite connection, j. Furthermore, the data packets added to the IP queue in response to a burst of acks arriving from the opposite node are transmitted as a bunch with no intervening acks. This behavior causes the entire window of each connection to be transmitted always as a single bunch [8], giving rise to the effects we study in this paper.

The functionality assumed for the IP layer is simple. For incoming traffic the IP layer is responsible for forwarding data from the lower layer to the local TCP process. Since TCP processing time is assumed to be small, no queueing is required for incoming traffic at the IP layer. For the outgoing traffic, on the other hand, a queue may be built up at the IP layer awaiting transmission on the link, as a result of ack bunching. Thus, in the worst case, an entire window's worth of packets may be added to the outgoing queue in quick succession due to a bunch of acks received from the opposite end. Therefore, to avoid packet losses at the source node, we assume that the IP queue has a size equal to the maximum window size of the sending TCP. The interaction between TCP and IP described above is consistent with the 4.4 BSD-Lite Unix Release [10].

The analytical models in this paper ignore the TCP processing time in end systems. In [8] we have shown how the TCP processing time, denoted as t_{pr}, may be incorporated into the analytical models. We also demonstrated that in configurations where this processing time is significantly smaller than the transmission time of a data packet over the physical link, the effect of TCP processing time on connection throughput is negligible. This is particularly true for the asymmetric case we consider in this paper: the transmission time of a data packet over the slow link typically dominates delays in any other system component. For example, the transmission time of a 1500 byte packet over a 100 Kbits/sec link is about 120 msecs, while the TCP protocol processing time in modern workstations does not exceed a few hundred microseconds [6, 11, 12]. Thus, we can safely assume that t_{pr} is zero.

We now review the analytical results on connection throughputs from [8] for the asymmetric case. First, we note that the sum of the windows of the two connections must be large enough to fill the round-trip pipe for ack compression to occur. If this condition is not satisfied, it is easy to see that no persistent queueing will occur at the IP queues at both nodes in steady state. Thus, if W_i and W_j are the window sizes of the two connections in packets, we must have

$$\frac{W_i}{\rho_i} + \frac{W_j}{\rho_j} > D_{ij} + D_{ji}, \tag{2.1}$$

for ack compression to occur. We will therefore assume that this condition always holds.

Under this condition the dynamics of two-way traffic may be summarized as follows: as the congestion window for a TCP connection increases, so does the number of data packets that are transmitted back-to-back as a batch. Acks for a window worth of data transmitted from the connection flowing in the opposite direction are queued in the end node's queue behind a window worth of data sent by the forward connection. The increased queueing delay for acks at the destination node increases the round-trip delay for that connection. As a result the window size for connection is never large enough to fill the *effective* round-trip pipe. Consequently, the network links remain idle while the TCP connection at the source node awaits acks that are queued behind a large amount of data at the destination node.

We call the ratio of the throughput of a connection to the corresponding link capacity as its *throughput efficiency*, or simply *efficiency*. The following results from [8] estimate the connection efficiencies and maximum queue sizes for two-way TCP traffic under ack compression. Detailed derivations of these results can be found in [8, 13].

Theorem 1: *The throughput efficiency F_i of connection i in a two-way traffic configuration with asymmetric link rates is given by*

$$F_i = \begin{cases} 1, & \text{if } W_i/\rho_i > W_j/\rho_j + (D_{ij} + D_{ji}); \\ \dfrac{2(W_i/\rho_i)}{(W_i/\rho_i + W_j/\rho_j) + (D_{ij} + D_{ji})}, & \text{if } W_j/\rho_j - (D_{ij} + D_{ji}) \le W_i/\rho_i \le W_j/\rho_j + (D_{ij} + D_{ji}); \\ \dfrac{(W_i/\rho_i)}{(W_j/\rho_j)}; & \text{otherwise.} \end{cases} \tag{2.2}$$

Theorem 2: *In steady state, the maximum occupancy $Q_{i,max}$ of the IP queue of node i under two-way traffic with asymmetric link rates is given by*

$$Q_{i,max} \le \begin{cases} W_i - (D_{ij} + D_{ji})\rho_i, & \text{if } W_i/\rho_i > W_j/\rho_j + (D_{ij} + D_{ji}); \\ \left(\frac{W_i}{\rho_i} + \frac{W_j}{\rho_j} - D_{ij} - D_{ji}\right)\rho_i, & \text{if } W_j/\rho_j - (D_{ij} + D_{ji}) \le W_i/\rho_i \le W_j/\rho_j + (D_{ij} + D_{ji}); \\ W_i, & \text{otherwise.} \end{cases} \tag{2.3}$$

Figure 2.3 illustrates the effect of asymmetric link capacities on the efficiencies of the TCP connections. These results were obtained from simulation of two TCP connections, one in each direction, over asymmetric point-to-point links. The efficiencies were measured in intervals of 3 seconds. The following parameter values were used in the simulation: The maximum window size for both TCP connections was set to 64 Kbytes. The capacities of the fast and slow links were taken as 5 Mbits/sec and 100 Kbits/sec, respectively. A fixed packet size of 1500 bytes was used for TCP data on each connection, and the ack size was taken as 28 bytes. The link propagation delays were assumed as zero. As predicted by Theorem 1, the slow connection achieved almost 100% utilization. However, the efficiency of the fast connection was under 2% (throughput of 100 Kbits/sec.). This outcome can be explained as follows: the slow connection takes 5.1 seconds to transmit a maximum window of packets (64 Kbytes) at 100 Kbits/sec. Due to ack compression, all the acks of the fast connection queue up for access to the slow link at node i, behind this full window of packets, encountering a queueing delay of 5.1 seconds. When these acks arrive at node j, the fast connection transmits a full window of packets in response. These packets are transmitted in

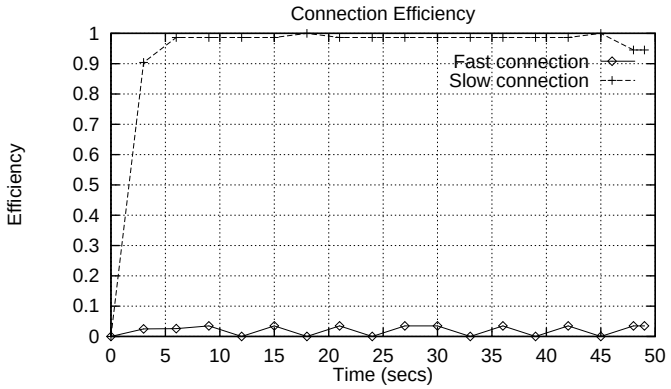


Figure 2.3: Connection efficiencies from simulation of two TCP connections in an asymmetric network (Maximum window size = 64 Kbytes, packet size = 1500 bytes, ack size = 28 bytes).

about 90 milliseconds down the fast link. Thus, the fast connection is able to transmit only for 90 milliseconds within an effective round-trip delay of 5.1 seconds, an efficiency of approximately 2%.

3 Measurements Demonstrating Effect of Ack Compression over Asymmetric Links

To verify the effect of ack compression in real networks with asymmetric links, we performed measurements on an experimental network with cable modems. The experimental setup consists of a Silicon Graphics Indy workstation on an isolated Ethernet connected through a router and a cable modem to a remote workstation, as illustrated in Figure 3.1. The downstream channel through the cable network has a nominal speed of 10 Mbits/sec, while the upstream link is via a 28.8 Kbits/sec modem connected to a telephone line. With link-layer data compression, the effective bandwidth of the upstream channel was found to be approximately 50 Kbits/sec. With IP header compression [14] enabled, we observed an ack size of 9 bytes on the serial link. We used "ttcp" to measure throughput of TCP connections between the workstations. Through separate measurements, we ensured that the workstations were not a bottleneck. Each measurement point in Figure 3.2 involved transmitting 10,000 user messages to ensure reasonable measurement accuracy.

Figure 3.2 shows the throughput of the TCP connection transferring data through the high-bandwidth path in both one-way and two-way TCP-traffic scenarios. In the one-way case, the socket buffer size was set to 16 KBytes and the observed throughput was approximately 1.2 Mbits/sec. This throughput is in fact what should be expected for the 16 KBytes window, since the effective round-trip delay in this case was about 80 msecs. We then initiated connections in both directions. The socket buffer size (maximum window size) for the upstream connection with respect to workstation B was fixed at 4 KBytes to avoid packet losses due to buffer overflow in the modem. The throughput for the slow upstream connection in all the experiments with two-way traffic was found to be approximately 50 Kbits/sec. The throughput for the fast connection, however, dropped dramatically under two-way traffic. For 4 KByte socket buffer size, the fast connection achieves a throughput of only about 50 Kbits/sec. This is consistent with that predicted by Eq. (2.2). It is important to note that increasing the socket buffer size of both connections by the same amount does not alter the results considerably, assuming that no packet losses occur in the network. With large socket buffer

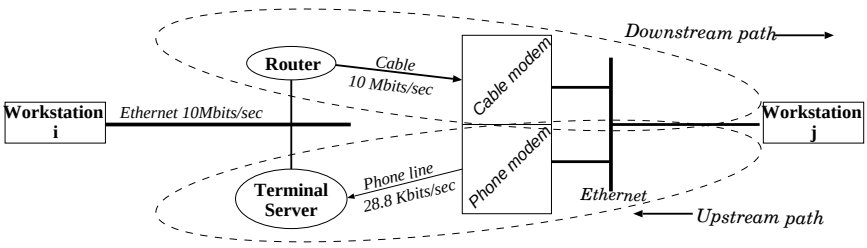


Figure 3.1: Configuration used for the experimental measurements.

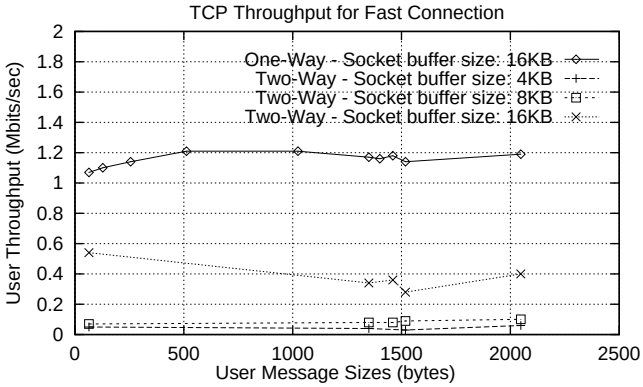


Figure 3.2: Measured throughput of the fast connection with varying socket buffer sizes. Socket buffer for the slow connection is fixed at 4 Kbytes.

sizes, however, there is a potential for packet loss in the cable modem. This, in turn, changes the dynamics of the connections and reduces the severity of ack compression.

Theorem 1 asserts that increasing the ratio of the socket buffer size of the fast connection to that of the slow connection improves the throughput of the former by a factor proportional to this ratio. This behavior is validated by the measurement data in Figure 3.2. Doubling the socket buffer size for the fast connection to 8 Kbytes improved its throughput by a factor of two. However, a further doubling of the buffer size to 16 Kbytes improved the throughput by more than a factor of four. This disproportionate increase is likely due to packet losses. A loss of acks on the fast link would force the slow connection to back off for significant amounts of time during which the fast connection is able to operate at full bandwidth. However, even in this last case the throughput of the fast connection is less than 25% of that observed under one-way traffic.

Finally, we must note in closing that the measurements were carried out in a controlled environment, and therefore the results should not be construed as representative of the performance of production networks.

4 Reducing Ack Compression by Prioritizing Acknowledgments

In the previous section, we showed that the throughput of the fast connection in an asymmetric environment can be degraded significantly under two-way traffic. This is mainly due to the queueing of acknowledgments of an entire window of packets of the fast connection behind data packets in the slow node. An obvious solution to reduce this effect is to service data and ack packets selectively

from the IP queue. Perhaps the simplest means of achieving this is to assign strict priority to acknowledgments over data packets while servicing the outgoing IP queue. When service is non-preemptive, this results in the lowest queueing delay for the acknowledgments. In this section, we analyze this solution and determine the connection efficiencies under such a scheme.

It is easy to show that prioritizing acks over data packets for transmission maximizes the throughput of the fast connection. The waiting time for a group of acks of the fast connection in the slow node is now within the transmission time of a single data packet on the slow link, as compared to an entire window of packets in the original system. This improvement, however, is achieved at the expense of degrading the slow connection's throughput. With priority queueing, the slow connection will be able to use only the portion of the upstream link that is not used for transporting acks of the fast connection. Thus, while the fast connection is free to increase its throughput until its acks use the entire available bandwidth on the slow link, the slow connection is made to adjust its throughput to the capacity left over on the slow link after transporting acks of the fast connection. In the extreme case, the slow link may be used exclusively for transporting acks, starving the slow connection completely.

In Section 4.1 we develop a simple analytical model to analyze the expected throughput of the fast and slow connections when acks. are transmitted with higher priority over data. In Section 4.2 we validate the analytical results with simulations.

4.1 Analysis

We assume that the window sizes of the two connections satisfy the necessary condition (2.1) for ack compression. As before, let α denote t_i^d/t_j^d, the ratio between the packet transmission times in the two directions. Similarly, let β denote the ratio of the packet transmission time to the ack transmission time on the slow link, that is, t_i^d/t_i^a. The portion of the slow link's bandwidth that will be used for transporting acks of the fast connection will be approximately t_i^a/t_j^d. The rest will be available to the slow connection for transmitting data. Therefore, the throughput of the slow connection under the priority queueing scheme is given by

$$F_i = 1 - \frac{t_i^a}{t_j^d} = 1 - \frac{\alpha}{\beta}, \quad \text{when } t_i^a < t_j^d. \tag{4.1}$$

Note that the throughput is zero if $t_i^a \geq t_j^d$.

More careful analysis is needed to determine the throughput of the fast connection. Let us first calculate the number of acks bunched behind a data packet being transmitted by the slow node. Since the fast node is transmitting data packets at the rate of $1/t_j^d$, the maximum number of acks that can be generated by node i during the transmission time of an outgoing data packet is t_i^d/t_j^d. In addition, this number can never exceed a full window of packets of the fast connection. Thus, the number of acks bunched behind a data packet on the slow link is given by

$$N_a = \min(n_j, t_i^d/t_j^d), \tag{4.2}$$

where n_j is the window size of the fast connection in number of packets. Immediately after transmitting the data packet, the slow node will begin clearing the bunched acknowledgments at the rate of $1/t_i^a$. While this clearing is in progress, new acks continue to join the outgoing queue at the rate of $1/t_j^d$. Thus, the number of acknowledgments transmitted by the slow node before starting transmission of the next data packet is given by

$$N_a \left(1 + (\frac{t_i^a}{t_j^d}) + (\frac{t_i^a}{t_j^d})^2 + \cdots\right) = N_a \frac{1}{1 - t_i^a/t_j^d} = N_a \frac{1}{1 - \alpha/\beta}, \tag{4.3}$$

and the total transmission time of the acks is given by

$$T_a = N_a \frac{t_i^a}{1 - \alpha/\beta}. \tag{4.4}$$

The elapsed time between transmission of two consecutive data packets of the slow connection will be equal to $t_i^d + T_a$. Meanwhile, the fast connection is able to maintain a steady flow of data as long as acks are flowing back and its congestion window is not exhausted. The maximum time the fast connection will be able to sustain a continuous flow of data will be $n_j t_j^d + T_a$. Thus, the efficiency for the fast connection will be

$$F_j = \frac{\min(n_j t_j^d, t_i^d) + T_a}{t_i^d + T_a} = \min\left(1, \frac{n_j\,(\beta/\alpha)}{\beta - \alpha + n_j}\right). \tag{4.5}$$

When $t_i^a \geq t_j^d$, the slow link is used exclusively for transporting acks of the opposite connection, and therefore the efficiency of the slow connection will be zero. The efficiency of the fast connection in this case reduces to the ratio of the data packet transmission time on the fast link to the ack transmission time on the slow link, that is $F_j = t_j^d/t_i^a = \beta/\alpha$.

In the following section we verify these analytical results with simulations.

4.2 Simulation Model and Results

We used the OPNET modeling tool for performing detailed TCP simulations. The model of TCP used in the simulations is based on the TCP-Reno version. It supports the congestion control mechanism described by Jacobson [1], exponential back-off, enhanced round-trip (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, the coarse-grain timers used in Unix TCP implementations (typically with a granularity of 500 ms) would make the comparison of the schemes difficult. To avoid the masking of the performance differences of TCP (which we believe will eventually be the case when we have finer-grained timers) due to coarse-grain timers, we used double-precision floating-point arithmetic in the RTT estimation algorithm. Therefore, both the RTT measurements and the timeout delays are represented by double-precision floating-point numbers. For each data segment received, an ack is generated immediately.

The service rate at the IP layer is set to be equal to the transmission rate of the physical link. This will cause all the interactions between data packets and acks to be confined to the IP queue, with no queueing at lower layers. Such an assumption will allow us to focus on a single queue and model its dynamics. In case the bottleneck is at a different layer, the same analysis applies to the queue built up at that layer.

We performed simulations to verify the analysis presented in Section 4.1, using the network configuration shown in Figure 4.1. In our simulation models all links are full duplex. The transmission rates in the two directions can be set independently. The capacity of the fast link (Fast node $\rightarrow$ Switch-2 $\rightarrow$ Switch-1 $\rightarrow$ Slow node) was set to $\rho_j = 5$ Mbits/sec and that of the slow link (Slow node $\rightarrow$ Switch-1 $\rightarrow$ Switch-2 $\rightarrow$ Fast node) to $\rho_i = 100$ Kbits/sec. The packet size for both TCP sessions was set to 1500 bytes and the maximum window size to 64 Kbytes.

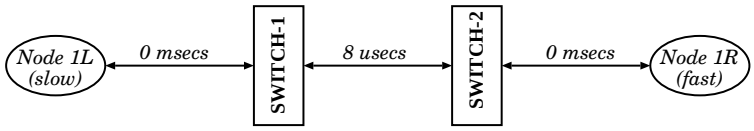


Figure 4.1: Network configuration used by the simulations.

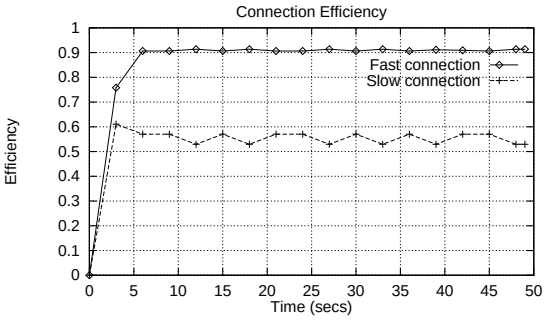


Figure 4.2: Throughput efficiencies of the two connections with priority queueing in end systems (Window size = 64 Kbytes, packet size = 1500 bytes, and ack size = 15 bytes).

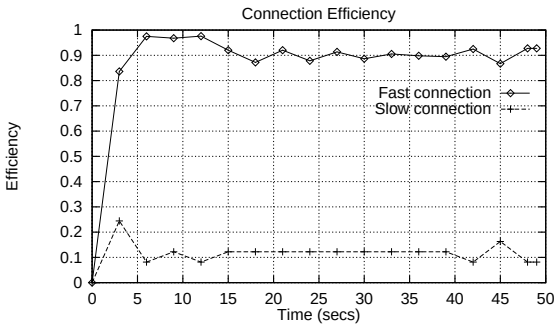


Figure 4.3: Throughput efficiencies of the two connections with priority queueing in end systems (Window size = 64 Kbytes, packet size = 1500 bytes, and ack size = 28 bytes).

In the first experiment, we consider the case where the acks have a size of 15 bytes. With this set of parameters, the packet transmission time on the fast link will be 2.4 msecs and the ack transmission time on the slow link 1.2 msecs. This suggests that the slow link will be used 50% of the time for transporting acks from the fast connection and the remaining capacity is available for data packets sent by the slow connection. The calculated efficiency from Eq. (4.5) is close to 92%. The simulation results for this case are shown in Figure 4.2. It is easy to verify that the measured throughputs closely follow the analytical estimates.

The important conclusion of this first experiment is that the efficiency of the fast connection is increased to almost the maximum achievable. Giving priority to acks has some impact on the performance of the slow connection which now has to share the slow upstream link with the acks from the fast connection. In this example, the slow connection operates acceptably, since it achieves more than 50% efficiency. In this first scenario, giving acks priority improved the system performance as a whole compared to the situation illustrated in Figure 2.3 where no provisions for reducing the ack delays are made.

The efficiency of the slow connection is highly sensitive to the ratio of the data packet transmission time on the fast link to the ack transmission time on the slow link. To demonstrate this, we performed a second experiment after increasing the ack size to 28 bytes, keeping the data packet size the same. This caused the transmission time of acks on the slow link to be only slightly lower than that of data packets on the fast link. This has no significant effect on the fast connection, but the throughput of the slow connection is severely degraded, as shown in Figure 4.3. If we further increase the ack size to 40 bytes, the ack transmission time on the slow link will now exceed the packet transmission time on the fast link, and the slow connection will be completely starved (Figure 4.4). Note that 40 bytes is close to the actual ack size when the IP header compression is turned off and no compression takes

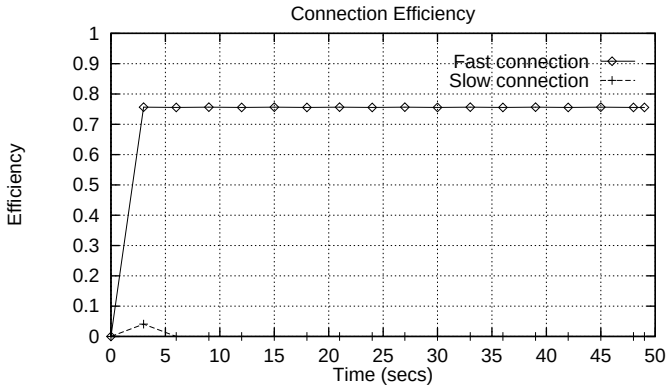


Figure 4.4: Throughput efficiencies of the two connections with priority queueing in end systems (Window size = 64 Kbytes, packet size = 1500 bytes, and ack size = 40 bytes).

place at the data-link layer. The efficiency of the fast connection in this case (approximately 75%) is determined by the ack transmission time and not by the capacity of the fast link.

Thus, in summary, priority queueing of acks can improve the throughput of the fast connection, but may make the performance of the slow connection considerably worse. Thus, to find a general solution to the problem, we must look beyond simple priority queueing to a scheme that provides some flexibility in controlling the throughput of *both* connections. We continue this investigation in the next section by evaluating the use of backpressure in the outgoing IP queue.

5 Use of Backpressure in Controlling Ack Compression

A more general solution to controlling the queueing delay of outgoing acknowledgments is to limit the number of data packets in the outgoing IP queue by applying backpressure to the TCP layer. This has the potential to allow control of the throughput efficiencies of the connections in both directions by varying the maximum allowable number of data packets in the outgoing IP queue. In this section we analyze the behavior of the TCP connections under such a scheme, derive their throughput efficiencies, and validate the results by simulation. We show that applying backpressure to data packets affords greater flexibility than the simple priority scheme of the last section, but still makes the throughput of the slow connection highly sensitive to the network and connection parameters.

The backpressure scheme we consider operates as follows: The transmitting TCP process in each node is allowed to send data packets to the IP layer as long as the backpressure threshold is not exceeded. When the backpressure threshold is reached, the transmitting TCP is suspended until space is available in the IP queue. The backpressure threshold can be set taking into account the target throughput efficiencies of the fast and slow connections.

We will now analyze this backpressure scheme with respect to the asymmetric network model of Figure 2.1 with two connections. We will derive simple analytical expressions for the throughput efficiencies of the fast and slow connections as a function of the link capacities in each direction and the parameters of the two connections. We will first ignore the link propagation delays in our analysis, but will later extend the analysis to cover non-zero propagation delays.

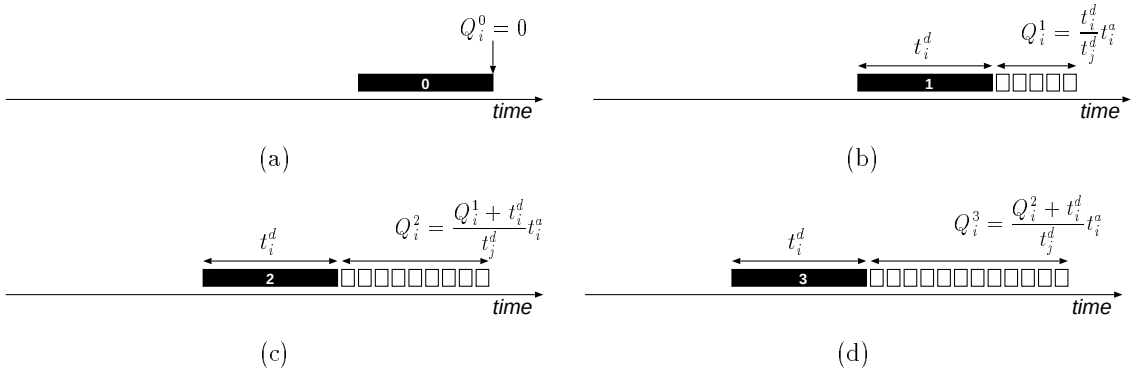


Figure 5.1: State of node i's IP queue for a threshold equal to one packet.

5.1 Analysis for Single-Packet Backpressure Threshold

We will first consider the simple case when the backpressure threshold is set to one data packet, and will then extend the results to the general case in Section 5.2 for multiple packets. To derive the throughput efficiencies of the connections, we need to determine the queueing delay seen by a data packet in the outgoing IP queue. We will assume that the transmission time of the entire window of the slow connection is more than the transmission time of a single data packet of the fast connection, that is, $W_i/\rho_i < t_j^d$. Otherwise it is easy to see that the backpressure scheme has no effect within node i. Figure 5.1 illustrates the evolution of the outgoing IP queue in node i, the slow node. The data packets are numbered sequentially from 0. Assume, for simplicity, that the slow and fast connections open simultaneously.

When the backpressure threshold is set to one packet, the queueing delay seen by any data packet is the time required to transmit acknowledgments accumulated in front of it. Thus, to calculate the queueing delay, we must determine the number of bunched acks in front of it. This can be calculated iteratively as follows: Let us denote by Q_i^k the queueing delay of the kth data packet added to the outgoing IP queue of node i. Obviously, $Q_i^0 = 0$, as shown in Figure 5.1(a). While packet 0 is under service, node i keeps receiving data packets of connection j at the rate of $1/t_j^d$ and an acknowledgment is added to the outgoing IP queue for every packet received. Thus, the number of acks aggregated behind packet 0 is given by $t_i^d/t_j^d = \alpha$. The state of the IP queue just when packet 0 completes transmission is shown in Figure 5.1(b). Transmission of packet 1 can begin only after all these acks are cleared from the queue. Therefore, the queueing delay of packet 1 is given by the transmission time of acks accumulated in behind packet 0, and is given by

$$Q_i^1 = \frac{t_i^d}{t_j^d} t_i^a = \left(\frac{\alpha}{\beta}\right) t_i^d.$$

Acknowledgments entering the outgoing IP queue after packet 1 will be delayed not only by the transmission time of packet 1, but also by the transmission time of the acks accumulated in front of it. Therefore, the queueing delay of the first ack accumulated behind packet 2 will be $Q_i^1 + t_i^d$, and the number of acks that will be bunched together behind packet 1 will be

$$\frac{Q_i^1 + t_i^d}{t_j^d}.$$

Consequently, the queueing delay of packet 2 is given by

$$Q_i^2 = \frac{Q_i^1 + t_i^d}{t_j^d} t_i^a.$$

This is illustrated in Figure 5.1(c). The behavior for subsequent packets is similar. Thus, the queueing delay for the kth data packet is given by

$$\begin{aligned} Q_i^k &= \frac{Q_i^{k-1} + t_i^d}{t_j^d} t_i^a \\ &= \left(\frac{\alpha}{\beta}\right) Q_i^{k-1} + \alpha t_i^a. \end{aligned} \tag{5.1}$$

Expanding Eq. (5.1) as a series,

$$Q_i^k = t_i^d \sum_{m=1}^{k} \left(\frac{\alpha}{\beta}\right)^m. \tag{5.2}$$

If $\alpha/\beta < 1$, the queueing delay will converge to a steady-state value given by

$$\begin{aligned} Q_i^S &= t_i^d \left(\sum_{m=0}^{\infty} \left(\frac{\alpha}{\beta}\right)^m - 1 \right) \\ &= t_i^d \frac{\alpha}{\beta - \alpha}. \end{aligned} \tag{5.3}$$

On the other hand, if $\alpha/\beta \geq 1$, then acknowledgments for an entire window of connection j will be accumulated behind each data packet in the outgoing IP queue of node i. The steady-state queueing delay of a data packet in node i's queue is then given by

$$Q_i^S = n_j t_i^a, \tag{5.4}$$

where n_j is the size of connection j's window in number of packets.

Knowing the steady-state queueing delay of outgoing data packets in node i, we can compute the throughput efficiencies of both connections:

Theorem 3: *The throughput efficiencies of connections i and j under the single-packet backpressure scheme are given by*

$$F_i = \begin{cases} 1 - \frac{\alpha}{\beta}, & \text{if } \alpha/\beta < 1; \\ \frac{\beta}{n_j + \beta}, & \text{otherwise.} \end{cases} \tag{5.5}$$

$$F_j = \begin{cases} \min\left(1, n_j(\frac{1}{\alpha} - \frac{1}{\beta})\right), & \text{if } \alpha/\beta < 1; \\ \min\left(1, \frac{n_j \beta}{(n_j + \beta)\alpha}\right), & \text{otherwise.} \end{cases} \tag{5.6}$$

Proof: The efficiency of each connection is simply the fraction of time during which the originating node transmits its data packets. Since each data packet of the slow connection, in steady state, experiences a queueing delay of Q_i^S, the efficiency of the slow connection will be

$$F_i = \frac{t_i^d}{Q_i^S + t_i^d}.$$

Substituting for Q_i^S from equations (5.3) and (5.4) for the cases of $\alpha/\beta < 1$ and $\alpha/\beta \geq 1$, respectively, this becomes

$$F_i = \begin{cases} 1 - \frac{\alpha}{\beta}, & \text{if } \alpha/\beta < 1; \\ \frac{\beta}{n_j + \beta}, & \text{otherwise.} \end{cases} \tag{5.7}$$

The efficiency of the fast connection can be computed by calculating the maximum delay seen by its acknowledgments in the outgoing IP queue of the slow node. It is easy to observe that this maximum delay is the queueing delay of a data packet plus its transmission time, that is, $Q_i^S + t_i^d$. The fast connection may transmit its entire window before receiving an ack. Thus, the efficiency of the fast connection will be

$$F_j = \min\left(1, \frac{n_j t_j^d}{Q_i^S + t_i^d}\right).$$

Again, substituting for Q_i^S from equations (5.3) and (5.4) for the cases of $\alpha/\beta < 1$ and $\alpha/\beta \geq 1$, respectively, this becomes

$$F_j = \begin{cases} \min\left(1, n_j(\frac{1}{\alpha} - \frac{1}{\beta})\right), & \text{if } \alpha/\beta < 1; \\ \min\left(1, \frac{n_j\beta}{(n_j+\beta)\alpha}\right), & \text{otherwise.} \end{cases} \tag{5.8}$$

This concludes the proof of Theorem 3.

5.2 Analysis for Multiple-Packet Backpressure Threshold

We can now generalize Theorem 3 for the case when the backpressure threshold is set to more than one data packet. Let b_i and b_j denote the backpressure thresholds in the outgoing IP queues of nodes i and j, respectively. As before, we need to calculate the queueing delay of data packets in the outgoing queue by computing the number of acks transmitted between data packets. To aid this computation, we can group the data packets transmitted out of node i into *batches*, with the number of packets in each batch equal to the backpressure threshold b_i. Figure 5.2 illustrates the evolution of the outgoing IP queue in node i, for $b_i = 2$. The first batch of data packets, batch 0, consists of packets 0 and 1. Since the queue is initially empty, these packets will be transmitted back-to-back with no intervening acks (Figure 5.2(a)). During the next batch, packets 2 and 3 are transmitted. However, during the transmission time of packet 0, a number of acks equal to $t_i^d/t_j^d = \alpha$ will be added to the IP queue and will be queued between packets 1 and 2 (Figure 5.2(b)). Thus, packet 2 will undergo an additional queueing delay of αt_i^a because of these acks. Similarly, during the transmission time of packet 1, α acks will be queued behind packet 2. Thus, it can be seen that every data packet in batch 1 will have a bunch of α acks queued ahead of it.

When the first packet of the next batch, packet 4, arrives into the queue, the number of acks accumulated in front of it is determined not only by the transmission time of packet 2, but also by the time needed to clear the acks in front of packet 2. This is evident from Figures 5.2(c) and (d). Let T_i^k denote the total transmission time of the acks in front of a packet in the kth batch in node i's IP queue. Then, the number of acks accumulated in front of packet 4 will be $(T_i^1 + t_i^d)/t_j^d$, where T_i^1 is the total transmission time of the acks accumulated in front of any packet in the previous batch. Therefore, the total transmission time of the acks in front of packet 4 is given by

$$T_i^2 = \frac{(T_i^1 + t_i^d)}{t_j^d} t_i^a.$$

Proceeding similarly, we can express T_i^k in terms of T_i^{k-1} by the recurrence

$$\begin{aligned} T_i^k &= \frac{T_i^{k-1} + t_i^d}{t_j^d} t_i^a \\ &= \left(\frac{\alpha}{\beta}\right) T_i^{k-1} + \alpha t_i^a. \end{aligned} \tag{5.9}$$

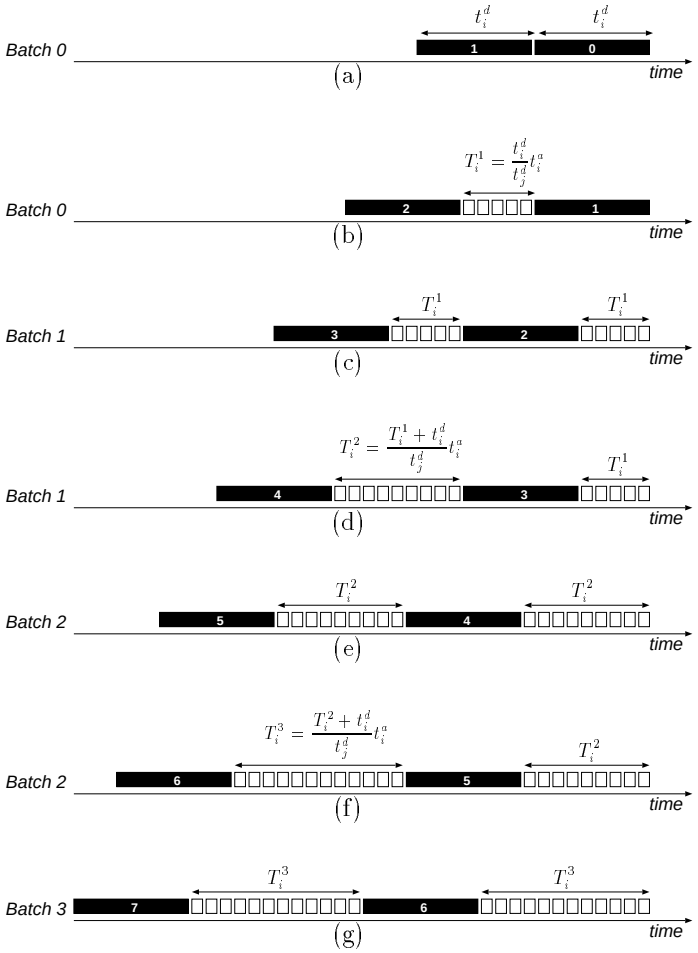


Figure 5.2: Example evolution of node i's outgoing IP queue when the backpressure threshold is two packets.

Note that this is the same recurrence as in Eq. (5.1). Therefore, the steady-state value of T_i^k is is given by

$$T_i^S = \begin{cases} t_i^d \frac{\alpha}{\beta - \alpha}, & \text{if } \alpha/\beta < 1; \\ n_j t_i^a, & \text{otherwise.} \end{cases} \tag{5.10}$$

In order to compute the maximum queueing delay of a data packet in node i's outgoing queue, we need to consider the transmission times of both data packets and acks ahead of them. At most $(b_i - 1)$ data packets can be in the IP queue when a new packet arrives, contributing to a delay of $(b_i - 1)t_i^d$. An additional delay of $b_i T_i^S$ is incurred in clearing the acks. Thus, using Eq. (5.10), the maximum queueing delay of a data packet in node i's queue in steady state is given by

$$Q_i^S = \begin{cases} t_i^d \frac{\alpha b_i}{\beta - \alpha} + (b_i - 1)t_i^d, & \text{if } \alpha/\beta < 1; \\ n_j t_i^a + (b_i - 1)t_i^d, & \text{otherwise.} \end{cases} \tag{5.11}$$

Knowing the queueing delay, the throughput efficiencies of the connections can be determined by proceeding as in the single-packet backpressure case.

Theorem 4: *When the backpressure thresholds in the nodes i and j are b_i and b_j, respectively, the throughput efficiencies of connections i and j are given by*

$$F_i = \begin{cases} 1 - \frac{\alpha}{\beta}, & \text{if } \alpha/\beta < 1; \\ \frac{\beta}{n_j + \beta}, & \text{otherwise.} \end{cases} \tag{5.12}$$

$$F_j = \begin{cases} \min\left(1, \frac{n_j}{b_i}(\frac{1}{\alpha} - \frac{1}{\beta})\right), & \text{if } \alpha/\beta < 1; \\ \min\left(1, \frac{n_j \beta}{(n_j + \beta b_i)\alpha}\right), & \text{otherwise.} \end{cases} \tag{5.13}$$

The proof of this theorem is similar to that of Theorem 3 and is therefore omitted. We can make some important observations from this theorem. The efficiency of the fast connection can be controlled by adjusting the backpressure threshold in the slow node's IP queue. A higher threshold, in general, increases its throughput. More important, however, is the fact that the efficiency of the slow connection is insensitive to the backpressure thresholds b_i and b_j. This establishes the important result that the backpressure scheme cannot overcome the basic weakness of the priority scheme in the last section, namely its inability to control the throughput of the slow connection. We will provide further evidence of this behavior in Section 5.3 using simulation.

5.3 Effect of Link Propagation Delay

So far we ignored link delays in our analysis. In this section we show that non-zero link propagation delays do not change the fundamental behavior of the connections when backpressure is used to limit the number of data packets queued in the nodes' outgoing IP queues. For simplicity, we consider only the single-packet backpressure threshold; extension to the multiple-packet case is straightforward and is omitted because of space constraints.

With non-zero link delays, the efficiency of the fast connection depends on its window size and the round-trip delay, among others. We need to consider two distinct cases: In the first case, the window size of the fast connection exceeds the bandwidth-delay product of the round-trip pipe plus the transmission time of a single packet from the slow connection. In this case, the maximum number of packets and acks of the fast connection that can be in transit at any time is given by $L_j = (D_{ij} + D_{ji} + t_i^d)/\max(t_j^d, t_i^a)$. Consequently, at most $\lambda_j = n_j - L_j$ acks may be queued behind a data packet in the slow node's outgoing IP queue. The efficiency of the fast connection can thus be determined by replacing n_j with λ_j in Eq. (5.6) derived for the zero-delay case.

The second case occurs when the window size of the fast connection is not able to fill the round-trip pipe. Since the data packets from the slow connection are subject to backpressure, the efficiency of the fast connection in this case is simply $(n_j t_j^d)/(L_j \max(t_j^d, t_i^a))$.

The efficiency for the slow connection can be derived directly from that of the fast connection by computing the percentage of time the slow link will be transmitting acks of the fast connection. The connection efficiencies calculated taking into account the link delays are summarized below:

$$F_i = \begin{cases} 1 - \frac{\alpha}{\beta}, & \text{if } \alpha/\beta < 1 \text{ and } n_j > L_j; \\ 1 - \frac{n_j \alpha}{L_j \beta} \min\left(1, \frac{\beta}{\alpha}\right), & \text{otherwise.} \end{cases} \tag{5.14}$$

$$F_j = \begin{cases} \min\left(1, \lambda_j(\frac{1}{\alpha} - \frac{1}{\beta})\right), & \text{if } \alpha/\beta < 1 \text{ and } n_j > L_j; \\ \frac{n_j}{L_j} \min\left(1, \frac{\beta}{\alpha}\right), & \text{otherwise.} \end{cases} \tag{5.15}$$

We have verified these analytical results with simulations. However, since the basic behavior of the connections was found to have little sensitivity to link delays, we show results only for the case of zero propagation delays in the next section.

5.4 Simulation Results

To validate the results from our analysis of the backpressure scheme, we performed simulations using the simple asymmetric configuration of Figure 2.1 with the same parameters as used previously. The size of acknowledgments was set as 28 bytes after IP header compression [14], and including link layer overhead. The backpressure threshold is set to one data packet.

For the first experiment, the size of data packets for both connections was set to 1500 bytes. Thus, $n_j = 43$ packets, $\beta = 54$, and $\alpha = 50$. Using Theorem 3, the expected efficiency of the slow connection i should be approximately 56% and the efficiency of the fast connection j about 48%. Figure 5.4 shows the efficiency for the two active TCP connections from simulation, matching our analysis. In this specific example, the transmission time of a window's worth of data from the fast connection is approximately the same as the transmission time of one data packet for the slow connection. Therefore, all the acks for data packets transmitted from the fast node j will be queued behind a single data packet in the slow node i, as in Figure 5.1. The transmission time for all the acks is about the time to transmit a single data packet on the slow link, resulting in approximately 50% efficiency for the slow connection. The efficiency of the fast connection j is significantly improved because of the backpressure mechanism. Note that the slow link is 100% utilized.

The exact efficiency of the slow connection depends on the amount of data transmitted from the slow connection for a window's worth of acks from the fast connection. Thus, reducing the size of data packets for the slow connection will have an even more significant impact on its performance The efficiency of the two TCP connections, when the size of data packets for the slow connection is reduced to 256 bytes is shown in Figure 5.4. The efficiency of the fast connection improves further, but that of the slow connection plummets. This is because a large portion of the slow link's bandwidth is used to carry acks for the fast connection. The analytically derived efficiencies would have been 17.2% and 87%, matching the simulation results quite well.

Notice that the dynamics of the backpressure mechanism are independent of the number of connections that have been setup in each direction and also independent of their relative timing. That is, the exact time when a new connection opens does not affect the fundamental system dynamics. To illustrate this point we simulated a situation where three connections were started in each direction in a staggered fashion, with a delay of 1 second between successive connections. Each connection is backpressured independently. The rest of the system parameters are the same as for the results shown in Figure 5.4: the data packet size is 1500 bytes and the ack size is 28 bytes. Figure 5.5(a) shows the efficiency achieved by each individual connection when three connections are set up in each direction. Observe that, in a given direction each connection gets a fair portion of the bandwidth available for data packets. The total efficiencies for the fast and slow connections (the sum of the efficiencies for connections transmitting in a given direction) are shown in Figure 5.5(b). The aggregate behavior is similar to the case where there is a single TCP session operating in each direction, shown in Figure 5.4.

To demonstrate the sensitivity of the performance of the slow connection to the number of acks generated for a window of the fast connection, we reduced the size of the data packets sent by the fast connection to 512 bytes. This caused the number of acks for a window worth of data sent by the fast connection to triple. Observe in Figure 5.6 that the efficiency of the slow connection i is now only about 6%. That is, only about 6 Kbits/sec out of the 100 Kbits/sec link capacity is available to the slow connection for transporting data packets. The remaining capacity is used for carrying acks for the fast connection. More importantly, the efficiency of the fast connection is also

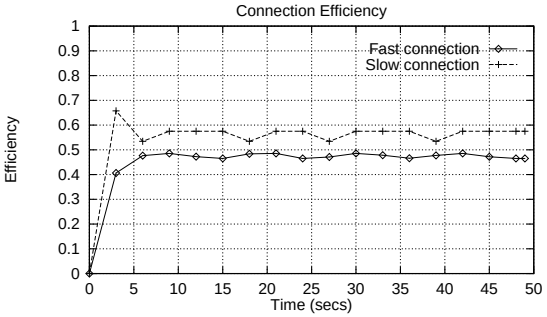


Figure 5.3: Efficiencies of the two connections under the backpressure scheme (Max window size = 64 Kbytes, packet size = 1500 bytes, ack size = 28 bytes).

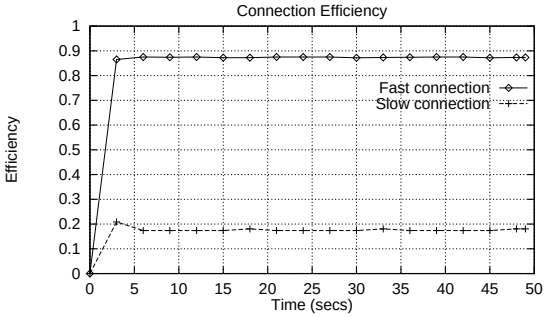
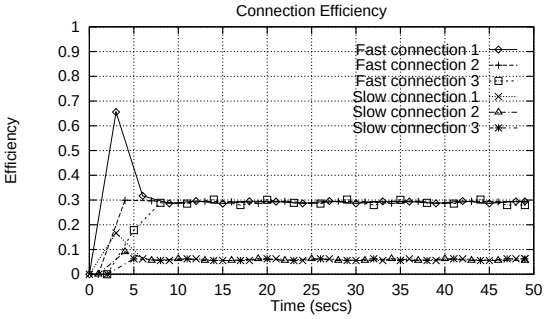
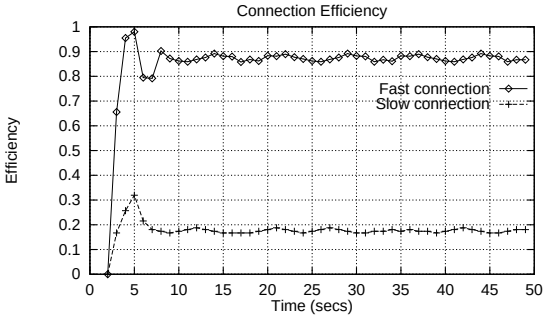


Figure 5.4: Efficiencies of the two connections under the backpressure scheme with 1500-byte packets for the fast connection and 256-byte packets for the slow connection (Max window size = 64 Kbytes, packet size = 1500 bytes, ack size = 28 bytes).



(a)



(b)

Figure 5.5: Efficiencies of six connections (three with a slow upstream link and three with a fast one) with the backpressure scheme for 64 Kbyte maximum window size, 1500-bytes packet size for the fast connection, 256-byte packets for the slow one, and 28 bytes ack size. (a) Efficiencies of individual connections and (b) the aggregate efficiencies of connections in each direction.

hurt. However, this is for a different reason: the transmission of a 512 byte packet over the fast link takes approximately 0.8 msecs while the transmission of an ack over the slow link takes about 2.2 msecs. Therefore, the ack transmission time on the slow link controls the throughput for the fast connection. This limitation however is simply due to the asymmetry in link speeds, and is not just due to bi-directional TCP effects. This suggests that the packets on the fast down-link should be sufficiently large to limit the bandwidth demand for sending acks on the slow up-link, even in the uni-directional case.

The simulations suggest that backpressuring the data can improve the efficiency of the fast connection. However, it results in an undesirable sensitivity of the two connections to each other's parameters, such as packet and ack sizes. A significant change in the packet size for the one connection, for example, results in a change in the throughput achieved by the other connection. There is no simple means by which the slow connection can get a fair portion of the capacity of the

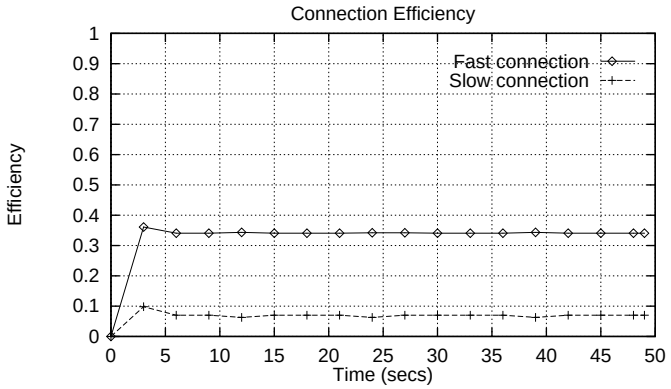


Figure 5.6: Efficiency for the two connections computed in 3 second intervals for 64 Kbytes maximum window size, 512 bytes packet size for the fast connection, 256 byte packets for the slow one, and 28 bytes ACK size (data packets are backpressured).

slow link even though acks and new data packets have equal priority, without undue dependence on variables that we cannot control, such as the packet size.

6 Connection-Level Bandwidth Allocation

Both priority queueing of acknowledgments and the use of backpressure are able to mitigate the effects of ack compression, but are inherently unfair to the slow connection. Both schemes make the throughput of the slow connection highly sensitive to parameters of the fast connection, such as its packet size and window size. An ideal solution to the ack compression problem should allow control of the throughput efficiencies of both the slow and fast connections. That is, such a scheme should maintain the slow link always fully utilized but allow control of the fraction of the data packets to acks transmitted on it. None of the previous schemes allow this flexibility. In this section, we propose and evaluate a simple connection-level bandwidth allocation scheme that achieves this objective.

The basic goal of our scheme is to provide a guaranteed minimum efficiency to the slow connection while providing the maximum possible throughput for the fast connection under this constraint. This is achieved by limiting the maximum number of acks a node is allowed to transmit before transmitting a data packet when one or more data packets are queued. This controls the fraction of time the slow link is used for transmitting acks when data packets are waiting, and vice-versa. Such a scheme can be implemented using two queues — one for data packets and the other for acks — and using a scheduler to make the choice between the two queues for transmission. A fair-queueing scheduler can be used to perform this task, but we can avoid the complexity of the scheduler by resorting to a simple counter-based implementation: This implementation simply counts the total number of bytes transmitted in a sequence of acks, and will force the transmission of a data packet if that number exceeds a preset threshold. Similarly, after the transmission of a data packet, waiting acks are given priority over data until the threshold on acks is again reached.

The pseudocode presented in Figure 6.1 illustrates how such a mechanism can be implemented with two counters *data_bytes_tx* and *ack_bytes_tx*, that keep track of the discrepancy between the number of data and ack bytes transmitted, and two flags indicating the presence of data or acks in the two outgoing queues (*data_present* and *ack_present*). The pseudocode shown is executed whenever the transmission of a data packet or ack is completed. The implementation shown in

```
if (system empty) then
        wait for next arrival;

if (data_present) and (! ack_present) then {
        ack_bytes_tx ← 0;
        data_bytes_tx ← data_pkt_size; /* in bytes */
        send( data packet ); }

if (ack_present) and (! data_present) then {
        data_bytes_tx ← 0;
        ack_bytes_tx ← ack_size; /* also in bytes */
        send (ack); }

if (data_present) and (ack_present) then {
        if (ack_bytes_tx ≥ data_bytes_tx) then { /* more acks transmitted than data */
                send (data packet);
                data_bytes_tx += data_pkt_size;
                if (data_bytes_tx > ack_bytes_tx) then {
                        data_bytes_tx − = ack_bytes_tx;
                        ack_bytes_tx = 0; } }
        else { /* more data transmitted than acks */
                send (ack);
                ack_bytes_tx += ack_size;
                if (ack_bytes_tx > data_bytes_tx) then {
                        ack_bytes_tx − = data_bytes_tx;
                        data_bytes_tx = 0; } } }
```

Figure 6.1: Example pseudocode of the bandwidth allocation mechanism to guarantee a mininum of 50% allocation to data packets sent from the slow connection.

Figure 6.1 assumes that available bandwidth on the slow link will be shared equally among data packets and acks. If data and acks are to be allocated a different fraction, then the *ack_byte_tx* counter needs to be scaled appropriately. For example if we want to guarantee 80% of the bandwidth to data from the slow connection then the *ack_byte_tx* counter must be multiplied by a factor of four. The objective of the mechanism is to minimize the difference between the two counters, *data_bytes_tx* and *ack_bytes_tx*. After sending a data packet, the counter *data_bytes_tx* is increased by the length of the packet. If the updated count crosses the value in *ack_bytes_tx*, both counters are decreased by *ack_bytes_tx* so as to bound their range. The operations after sending an ack are symmetric.

It can be shown easily that the above mechanism is able to provide a throughput guarantee to the slow connection. Let f denote the fraction of the capacity of the slow link that is to be allocated to the slow connection's data. The remaining fraction $1 - f$ is then available for transporting the acks of the fast connection. Given f, we can compute the threshold on the number of acks, γ, which triggers the transmission of the next data packet, from the following equality:

$$\frac{t_i^d}{t_i^d + \gamma t_i^a} = f,$$

which gives

$$\gamma = \frac{(1-f)\beta}{f}. \tag{6.1}$$

We will now derive closed-form expressions for the throughput efficiencies of the two connections under this scheme. The maximum time T_{max} between successive transmissions of data packets from the slow connection will be

$$T_{max} = \min(n_j t_i^a, \gamma t_i^a). \tag{6.2}$$

Therefore, the efficiency of the slow connection i will be

$$\begin{aligned} F_i &= \frac{t_i^d}{T_{max} + t_i^d}, \\ &= \frac{1}{\min(n_j, \gamma)\frac{t_i^a}{t_i^d} + 1}, \\ &= \max\left(\frac{\beta}{n_j + \beta}, f\right). \end{aligned} \tag{6.3}$$

Notice from Eq. (6.3) that the efficiency of the slow connection is guaranteed to be at least f.

The efficiency of the fast connection j can be computed by observing that at most $\min(n_j, \gamma)$ packets may be acknowledged within in an interval of time $\min(n_j, \gamma)t_i^a + t_i^d$. Therefore, the efficiency will be

$$\begin{aligned} F_j &= \min\left(1, \frac{\min(n_j, \gamma)t_j^d}{\min(n_j, \gamma)t_i^a + t_i^d}\right), \\ &= \min\left(1, \frac{n_j t_j^d}{n_j t_i^a + t_i^d}, \frac{\gamma t_j^d}{\gamma t_i^a + t_i^d}\right), \end{aligned} \tag{6.4}$$

After some manipulation, this reduces to

$$F_j = \min\left(1, \frac{\beta n_j}{\alpha(n_j + \beta)}, \frac{(1-f)\beta}{\alpha}\right). \tag{6.5}$$

We now discuss the results from a series of simulations we performed to validate the above analysis. In all of the simulation experiments discussed below the parameter f is set to 50%, unless otherwise specified. This allows the slow connection to reach a minimum efficiency of 50%.

Figure 6.2 considers the case where the size of data packets in each direction is set to 1500 bytes. The link capacities are the same as those considered in the previous section, that is, 5 Mbits/sec and 100 kbits/sec. The window size for both connections is set to 64 Kbytes, so that $n_j = 43$, $\alpha = 50$, and $\beta = 50$. Therefore, the expected efficiency for connection i from analysis is approximately 54%, and that of connection j is approximately 46%. The simulation results in Figure 6.2 are in close agreement with these estimates. In this case both connections achieve acceptable performance. Note that the throughput of the fast connection is limited only by the bandwidth available for transporting its acks over the slow link.

To test the sensitvity of the throughput on the parameters of the opposite connection, we decreased the packet size of the slow connection to 256 bytes, maintaining the same 1500-byte packets for the fast connection. The results are shown in Figure 6.3. Note that the efficiencies of both connection remains almost unaffected because the bandwidth needed for the additional ack

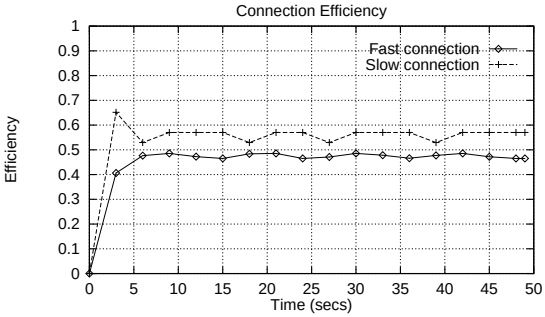


Figure 6.2: Efficiencies for the two connections with connection-level bandwidth allocation (Max window size = 64 Kbytes, packet size = 1500 bytes, ack size = 28 bytes).

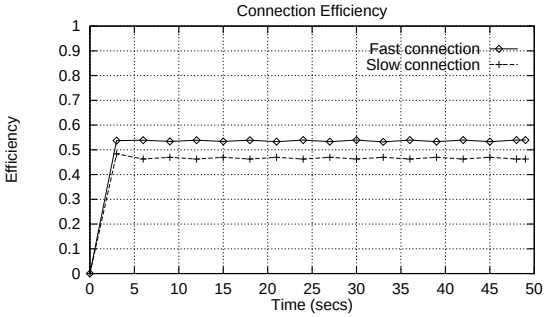


Figure 6.3: Efficiencies for the two connections with connection-level bandwidth allocation with 1500-byte packet size for the fast connection and 256-byte packets for the slow connection (Max window size = 64 Kbytes, ack size = 28 bytes).

traffic on the fast link is easily available. The slight discrepancy between the observed efficiency of the slow connection and its ideal value (50%) is due to the rounding error resulting from the size of data packets not being an exact multiple of the size of acknowledgments. On the other hand, use of the backpressure scheme in this case reduces the efficiency of the slow connection to about 20%.

In Figure 6.4 we consider the case where the size of connection j's data packets is also reduced to 512 bytes. With these parameters, the slow connection was able to get only about 7% of the link capacity when the backpressure scheme was used. On the other hand, with the bandwidth allocation mechanism, the throughput of the slow connection remains almost unaffected (Figure 6.4). The smaller packet size of the fast connection, however, causes a reduction in its own efficiency. This is because the smaller packet size causes an increase in ack traffic for the same amount of data transmitted; since the bandwidth available to ack traffic on the slow link is fixed, this results in a reduction in the fast connection's throughput. This is in fact the desired behavior, since the throughput of a connection is now sensitive to only its own parameters.

The ability to control the parameter f allows a great deal of flexibility in achieving the desired performance objectives. For example consider the case where the packets of the fast connection are 1500 bytes long and those of the slow connection 256 bytes. When 50% of the slow link's bandwidth is allocated for transporting data packets, each of the connections achieves approximately 50% efficiency (Figure 6.3). However, for the same parameters settings, the use of the backpressure results in the fast connection achieving about 85% efficiency and the slow connection 18%. By setting the percentage of the bandwidth that is guaranteed for data packets to 18%, we can produce the same behavior with connection-level bandwidth allocation. This is demonstrated in the simulation results of Figure 6.5, which were obtained by changing the parameter f from 50% to 18%.

7 Conclusion

Asymmetric link speeds are likely to be common in data communication networks in the future, particularly with the deployment of cable and high-speed Digital Subscriber Line (DSL) technologies. With the increasing use of powerful PCs and workstations at homes and small businesses, it is not

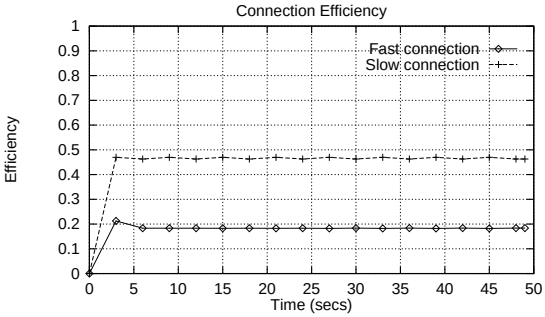


Figure 6.4: Efficiencies for the two connections with connection-level bandwidth allocation with 512-byte packet size for the fast connection and 256-byte packets for the slow connection (Max window size = 64 Kbytes, ack size = 28 bytes).

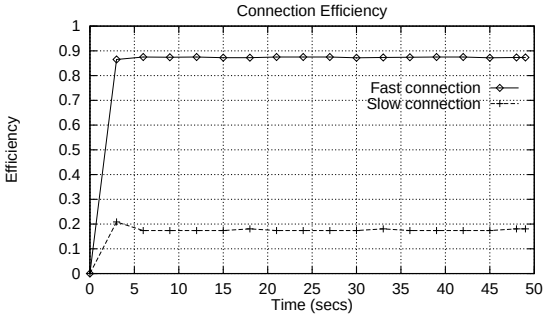


Figure 6.5: Efficiencies for the two connections with connection-level bandwidth allocation with 1500-byte packet size for the fast connection and 256-byte packets for the slow connection (Max window size = 64 Kbytes, ack size = 28 bytes). The bandwidth allocation parameter f is set to 18%.

unrealistic to imagine users running multiple applications simultaneously (download files, transfer data on the uplink) that result in two-way TCP traffic. In such an environment, TCP has been known to suffer a reduction in the overall throughput of one or both connections due to the well-known effect of ack compression. This effect is exacerbated in an environment where the link speeds in the two directions are widely different.

In this paper, we quantitatively analyzed the degradation in the throughput of TCP connections in the two-way environment. We first presented measurement results from a network using cable modems, where the degradation in throughput of the TCP connection on the faster downlink because of the second TCP connection on the slow uplink was dramatic. The measurement results matched well with the results from our quantitative analysis, lending credence to the use of a relatively simple network model for studying the problem. Further, through simulations, we showed that the throughput of the fast connection can become a small fraction of the link capacity due to the effect of ack compression.

We then examined several schemes for improving the performance of bi-directional TCP connections, by carefully controlling the flow of packets and acknowledgments, particularly on the slow link. We first examined the scheme that provides higher priority to acks waiting for transmission on the slow link, so as to minimize the degradation in throughput for the fast connection. Through both analysis and simulation, we showed that the throughput for the fast connection consistently improves, in one case going from 2% when there was no control to 92% with priority queueing of acks. Unfortunately, this is achieved at the expense of the slow connection. In addition, as the total bandwidth consumed by acks on the slow link becomes a significant portion of its capacity, the slow TCP connection in this direction may become completely starved. This is clearly an unacceptable behavior.

As a logical extension of the mechanism to provide priority to acks, we then examined a scheme that limits the number of data packets in the outbound queue for the slow link. This form of backpressure modulates the priority given to acks. by allowing the slow connection to place no more than a set number of data packets in the outbound queue. Our analysis technique accurately predicts

throughput for the two connections for different values of the backpressure threshold, validated by simulations. The backpressure mechanism, while improving the throughput for the fast connection, again results in an undesirable sensitivity of the two connections to each other's parameters, such as packet and ack sizes. A decrease in the packet size for the fast connection, for example, results in a corresponding degradation of the throughput achieved by the slow connection.

Finally, our preferred solution to the problem is to use a connection-level bandwidth allocation mechanism to overcome the sensitivity of the throughput of each connection to the parameters of the opposite connection. Through a simple counter-based implementation, we showed that it is possible to guarantee a minimum throughput for the slow connection, making the fast connection's throughput sensitive to only its own parameters.

Our contribution in this paper is thus two-fold: our analytical results, and the simple closed-form expressions we derive can be used to accurately predict the performance of the TCP connections in the two directions. Secondly, our proposal to use the simple bandwidth allocation mechanism allows us to not only improve, but also to "tune" the throughputs of the two connections in a flexible way using our analytical results for guidance.

References

[1] V. Jacobson, "Congestion avoidance and control," in *Proc. 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] 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.

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

[5] 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.

[6] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," in *Proc. of ACM SIGCOMM'91*, pp. 133–147, September 1991.

[7] 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.

[8] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Two-way TCP traffic over ATM: Effects and analysis," in *Proc. of IEEE INFOCOM'97*, April 1997.

[9] K. Maxwell, "Asymmetric digital subscribe line: Interim technology for the next forty years," *IEEE Communications Magazine*, vol. 34, no. 10, pp. 100–106, October 1996.

[10] W. R. Stevens and G. R. Wright, *TCP/IP Illustrated*, vol. 2. Addison-Wesley Publishing Company, 1995.

[11] K. K. Ramakrishnan, "Performance considerations in designing network interfaces," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, pp. 203–219, February 1993.

[12] C.-H. Chang, D. Flower, J. Forecase, H. Gray, B. Hawe, A. Nadkarni, K. K. Ramakrishnan, U. Shikarpur, and K. Wilde, "High-performance TCP/IP and UDP/IP networking in DEC OSF/1 for Alpha AXP," in *Proc. of the Third IEEE International Symposium on High Performance Distributed Computing*, pp. 35–42, August 1994.

[13] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Two-way TCP traffic over ATM: Effects and analysis," Tech. Rep. UCSC-CRL-96-23, Univ. of California, Santa Cruz, 1996.

[14] V. Jacobson, "Compressing TCP/IP headers for low-speed serial links," *Request for Comments: 1144*, February 1990.