TCP Improvements for Heterogeneous Networks: The Daedalus Approach

Hari Balakrishnan*, Venkata N. Padmanabhan*, Srinivasan Seshan+, Mark Stemm*, Elan Amir*, Randy H. Katz*
{hari,padmanab,stemm,elan,randy}@cs.berkeley.edu, srini@watson.ibm.com

**Computer Science Division*
University of California at Berkeley
Berkeley, CA 94720

+IBM T.J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

In this paper, we present a summary of the research work done by the Daedalus group at the University of California at Berkeley to improve TCP performance in a variety of situations, including lossy wireless networks, asymmetric networks, and Web traffic workload. To improve the performance of TCP over cellular wireless networks, we have implemented a TCP-aware link layer protocol called the *snoop protocol* that isolates wired senders from the lossy characteristics of a wireless link. To improve TCP performance over networks exhibiting asymmetry, we have designed and evaluated techniques that reduce the adverse effects of the constrained reverse path, which TCP acks traverse. To improve TCP performance for Web-like traffic, we have designed and evaluated an enhanced data-driven loss recovery scheme that improves the performance of short transfers, as well as an *integrated congestion control and loss recovery* approach that improves the end-to-end performance of multiple, short TCP connections from a single host.

1. Introduction

The growth of the Internet and the World-Wide Web has increased the diversity of situations in which TCP connections are used. There are two aspects to this diversity: heterogeneous networks and new traffic workload. Networks with wireless links and asymmetric networks such as cable modem, direct broadcast satellite and packet radio networks are increasingly being used to provide Internet access to the workplace and home. Also, the preponderance of Web traffic has resulted in most TCP connections being relatively short in duration. These trends have impacted TCP significantly.

First, the traditional assumption made by TCP that packet losses are indicators of network congestion is correct for wired networks but breaks down in wireless networks where packets are often corrupted by bit errors. Second, the design of TCP algorithms heretofore has primarily focussed on the network path characteristics on the direction of data transfer. The flow of TCP acknowledgments has received only limited attention (e.g., [23]). However, in the presence of bandwidth and other types of asymmetry, the constrained flow of acks can have a significant adverse effect on performance. Finally, TCP's window growth and data-driven loss recovery algorithms have been designed with long data transfers in mind. Their effectiveness is often limited in the context of short Web transfers. In summary, TCP falls short on several fronts in the presence of new types of networks and traffic workloads.

The Daedalus group at the University of California at Berkeley [11] has been working on improving TCP performance in each of the contexts mentioned above: wireless links, asymmetric networks and Web traffic. This paper presents a brief summary of the techniques that we have investigated and our findings:

- To improve performance over wireless links, we have implemented a TCP-aware link layer protocol called the *snoop protocol* that shields a TCP sender from the lossy characteristics of the wireless link [6, 7]. We have also compared this approach to a variety of other approaches for improving wireless TCP performance[4].
- To improve performance over asymmetric links, we have implemented techniques for eliminating the acknowledgment (ack) bottleneck that comes about as a result of *bandwidth asymmetry*, when the forward link has significantly higher bandwidth than the reverse link [3].
- To improve the performance of TCP connections for Web traffic, we have proposed an enhanced loss recovery scheme that improves performance of short transfers and an integrated connection approach to loss recovery and congestion control that improves performance for concurrent Web connections [5].

The rest of this paper is organized as follows. In Section 2 we describe our improvements for lossy wireless links. In Section 3 we compare this approach to other approaches for improving TCP performance over wireless links. In Section 4 we describe our improvements for overcoming asymmetries in network characteristics between the forward and reverse directions. In Section 5 we describe our improvements for Web traffic and conclude in Section 6.

2. The Snoop Protocol [6]

Reliable transport protocols like TCP [10, 20, 8] have been tuned for traditional networks made up of wired links and stationary hosts. Due to the relatively low bit-error rates over wired networks, all packet losses are assumed to be due to congestion. In the presence of the high error rates and periods of intermittent connectivity characteristic of wireless links, TCP reacts to packet losses as it would in the wired environment: it drops its transmission window size before retransmitting packets, initiates congestion control or avoidance mechanisms (e.g., slow start [14]) and resets the retransmission timer (Karn's Algorithm [16]). These measures result in a unnecessary reduction in the link's bandwidth utilization, thereby causing a significant degradation in performance in the form of poor throughput and very high interactive delays[9].

In [7], we present the details of the *snoop* protocol, a protocol designed to improve performance for data transfers to and from the mobile host. Here, we outline the basic schemes used to improve performance for data transfers to the mobile host.

2.1 Details of the Snoop Protocol

The snoop protocol is a TCP-aware link protocol, running primarily at the base station to the wireless network. The snoop agent at the base station monitors every TCP packet that passes through the

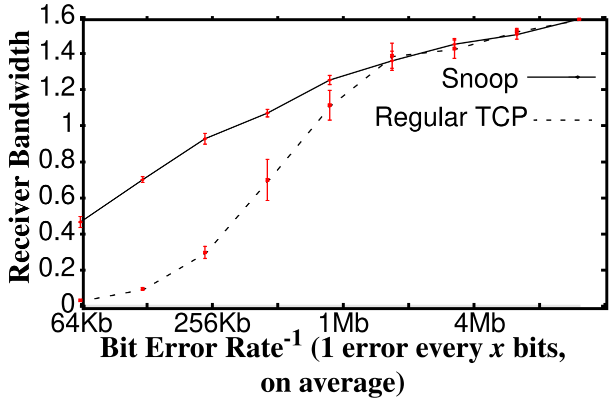

Figure 1. Throughput received by the mobile host at different bit-error rates ($\log_2$ scale).

connection in either direction. The snoop agent maintains a cache of TCP packets sent from the Fixed Host (FH) that haven't yet been acknowledged by the Mobile Host (MH). This is possible because TCP has a cumulative acknowledgment policy for received packets. When a new packet arrives from the FH, the snoop agent adds it to its cache and passes the packet on to the routing code which performs the normal routing functions. The agent also keeps track of all the acknowledgments sent from the mobile host. It detects the loss of packets either by the arrival of duplicate acknowledgments, or by a local timeout. Upon detecting packet loss, it retransmits the lost packet to the mobile host if it has the packet cached. In addition, the basestation hides the packet loss from the sender by not propagating duplicate acknowledgments, thereby preventing unnecessary congestion control mechanism invocations at the source, which would other wise result because of TCP's fast retransmission mechanism.

2.2 Results

We performed several experiments with the snoop protocol on our wireless testbed, consisting of Pentium PC base stations and IBM ThinkPad mobile hosts communicating over a 915 MHz AT&T Wavelan [21], all running BSD/OS 2.0, and compared the resulting performance with unmodified TCP. In the presence of no packet losses, the wireless link was able to support about 1.5 Mbits/s out of a rated maximum of 2 Mbits/s.

A preliminary design of the snoop protocol with some simulation results appeared in [1]. To measure the performance of the implementation under controlled conditions, we used a Poisson-distributed bit error model. Figure 1 compares the bandwidth of a connection using the snoop protocol with that of a connection using an unmodified TCP implementation for various Poisson-distributed bit-error rates shown on a log scale. We generated a Poisson distribution for each bit-error rate and changed the TCP checksum of the packet at the basestation if the error generator determined that the packet should be dropped at the receiver before forwarding the packet over the wireless link.

We see that for error rates of over 10^{-6} (close to the 1 Mb point on the x-axis of the graph) the snoop protocol performs significantly better than unmodified TCP, achieving a bandwidth improvement factor of 2 to 20 depending on the bit error rate. In fact, the snoop protocol was robust and completed the run even when every other packet was being dropped over the last link, while the regular TCP connection didn't make any progress. Under conditions of very low bit error rates ($< 10^{-6}$), there does not seem to be any significant difference between the snoop and regular TCP implementations. At such low bit errors the snoop layer is essentially just a store-and-forward agent, and incurs no additional overhead.

In the next section, we compare snoop to other link-layer approaches for improving TCP performance on high-loss wireless networks in order to understand precisely why the protocol performs well.

3. Comparison of Snoop with Other Approaches

Several schemes have been proposed to the alleviate the effects of non-congestion-related losses on TCP performance over networks that have wireless or similar high-loss links [2, 7, 22]. These schemes choose from a variety of mechanisms, such as local retransmissions, split-TCP connections, and forward error correction, to improve end-to-end throughput. However, it is unclear to what extent each of the mechanisms contributes to the improvement in performance. In [4], we examine and compare the effectiveness of these schemes and their variants, and experimentally analyze the individual mechanisms and the degree of performance improvement due to each.

We break the approaches into three categories: (1) *link-layer protocols*, ones that operate solely at the link layer, (2) *end-to-end protocols*, ones that operate solely at the connection endpoints, and (3) *split-connection protocols*, ones that break the TCP connection at the wireless endpoints.

We seek to answer the following specific questions:

1. What combination of mechanisms results in the best performance for each of the protocol categories?
2. How important is it for link-layer schemes to be aware of TCP algorithms to achieve high end-to-end throughput?
3. How useful are selective acknowledgments in dealing with lossy links, especially in the presence of burst losses?
4. Is it important for the end-to-end connection to be split in order to effectively shield the sender from wireless losses and obtain the best performance?

In the next section, we focus on the importance of transport-awareness in link-layer protocols.

3.1 Results

The results reported in this section were obtained through implementing and evaluating the various protocols in a wireless testbed consisting of Pentium PC base stations and IBM ThinkPad mobile hosts communicating over a 915 MHz AT&T Wavelan [21], all running BSD/OS 2.0.

In order to measure the performance of the protocols under controlled conditions, we generate errors on the lossy link using a Poisson-distributed bit-error model. The receiving entity on the lossy link generates a Poisson distribution for each bit-error rate and changes the TCP checksum of the packet if the error generator determines that the packet should be dropped. Losses are generated in both directions of the wireless channel, so TCP acknowledgments are dropped too, albeit at a lower per-packet rate. For most of the experiments, the TCP data packet size is 1400 bytes and the average error rate is one every 64 KBytes (this corresponds to a bit-error rate of about 1.9×10^{-6}). The choice of the Poisson-distributed error model is motivated by our desire to understand the precise dynamics of each protocol in response to a wireless loss, and is not an attempt to empirically model a wireless channel. While the actual performance numbers will be a strong function of the exact error model, the relative performance is dependent on how the protocol behaves after one or more losses in a

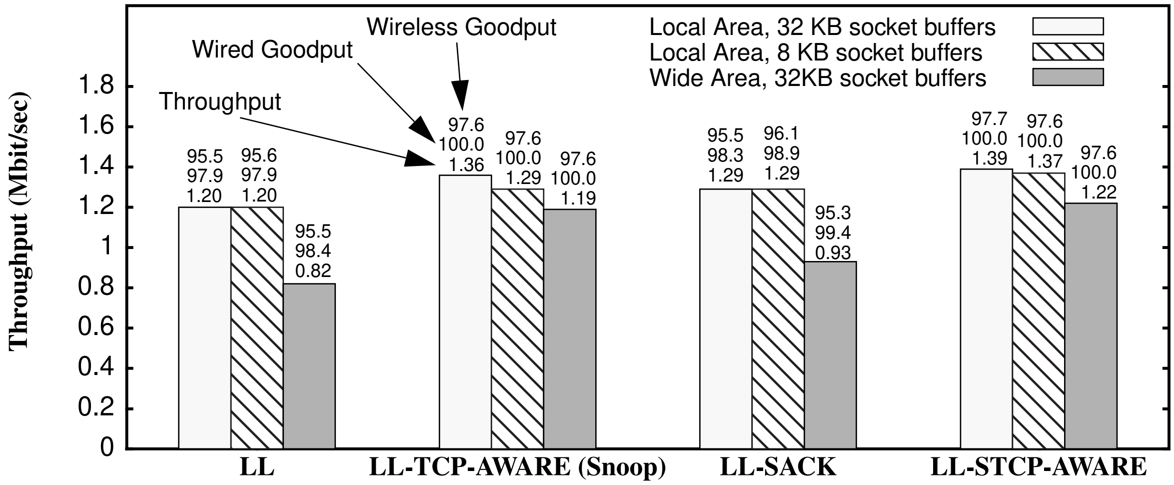

Figure 2. Performance of link-layer protocols: bit-error rate = 1.9×10^{-6} (1 error/65536 bytes).

single TCP window. Thus, we expect our overall conclusions to be applicable under other patterns of wireless loss as well.

In our experiments, we attempt to ensure that losses are only due to wireless errors (and not congestion). This allows us to focus on the effectiveness of the mechanisms in handling such losses. The WAN experiments are performed across 16 Internet hops with minimal congestion[1] in order to study the impact of large delay-bandwidth products.

Each run in the experiment consists of an 8 MByte transfer from the source to receiver across the wired net and the WaveLAN link. A particular metric of interest is the *goodput* of a path (or link) which is defined as the ratio of the actual transfer size to the total number of bytes transmitted over that path. In general, the wired and wireless goodputs differ because of wireless losses, local retransmissions and congestion losses in the wired network. For each protocol, we measured the end-to-end throughput and goodputs for the wired and one-hop wireless paths. The rest of this section presents and discusses the results of these experiments.

3.1.1 Link-Layer Protocols

The performance of the various link-layer protocols is summarized in Figure 2. In this section, we compare four protocols: (i) LL, a basic link layer protocol with cumulative acks and timer-based retransmissions, (ii) LL-TCP-AWARE, a link layer protocol that is aware of TCP (such as snoop), (iii) LL-SACK, a link layer protocol with selective acknowledgments, and (iv) LL-STCP-AWARE, a link layer protocol that combines TCP awareness with selective acknowledgments.

Traditional link-layer protocols operate independently of the higher-layer protocol, and consequently, do not necessarily shield the sender from the lossy link. This could adversely impact TCP performance for two reasons: (i) competing retransmissions caused by an incompatible setting of timers at the two layers, and (ii) the effect of the link layer protocol on the TCP fast retransmission mechanism. However, (i) is not the dominating effect when link layer schemes such as LL are used with TCP Reno and its variants. These TCP implementations have coarse retransmission timeout granularities that are typically multiples of 500 ms, while link-layer protocols typically have much finer timeout granularities. The real problem is that when packets are lost, link-layer protocols that do not attempt in-order delivery across the link (e.g., LL) cause packets to reach the TCP receiver out-of-order. This leads to the generation of duplicate acknowledgments by the TCP receiver, which causes the sender to invoke fast retransmission and recovery and can potentially cause degraded throughput and goodput, especially when the delay-bandwidth product is large.

Our results substantiate this claim, as can be seen by comparing the LL and LL-TCP-AWARE results. For a packet size of 1400 bytes and a bit error rate of 1.9×10^{-6} (1/65536 bytes), the packet error rate is about 2.3%. Therefore, an optimal link-layer protocol that recovers from errors locally and does not compete with TCP retransmissions should have a wireless goodput of 97.7% and a wired goodput of 100% in the absence of congestion. In the LAN experiments, the throughput difference between LL and LL-TCP-AWARE is about 10%. However, the LL wireless goodput is only 95.5%, significantly less than LL-TCP-AWARE's wireless goodput of 97.6%, which is close to the maximum achievable goodput. When a loss occurs, the LL protocol performs a local retransmission relatively quickly. However, enough packets are typically in transit to create more than 3 duplicate acknowledgments. These duplicates eventually propagate to the sender and trigger a fast retransmission and the associated congestion control mechanisms. These fast-retransmissions result in reduced goodput; about 90% of the lost packets are retransmitted by both the source (due to fast retransmissions) and the base station.

The effects of this interaction are much more pronounced in the wide area experiments — the throughput difference is about 30% in this case. The cause for the more pronounced deterioration in performance is the higher bandwidth-delay product of the wide-area connection. The LL scheme causes the sender to invoke congestion control procedures often due to duplicate acknowledgments and causes the average window size of the transmitter to be lower than for LL-TCP-AWARE.

In summary, our results indicate that a simple link-layer retransmission scheme could adversely impact TCP performance. An enhanced link-layer scheme, that uses knowledge of TCP semantics to prevent duplicate acknowledgments caused by wireless losses from reaching the sender, achieves significantly better performance.

3.1.2 End-To-End Protocols

In this section, we compare 5 protocols: (i) E2E, basic TCP Reno, (ii) E2E-NEW, TCP Reno enhanced with partial fast recovery, (iii) E2E-SACK, TCP Reno enhanced with selective acknowledgments, (iv) E2E-ELN, TCP Reno enhanced with Explicit Loss Notification

1. WAN experiments were performed between 10 pm and 4 am, PST.

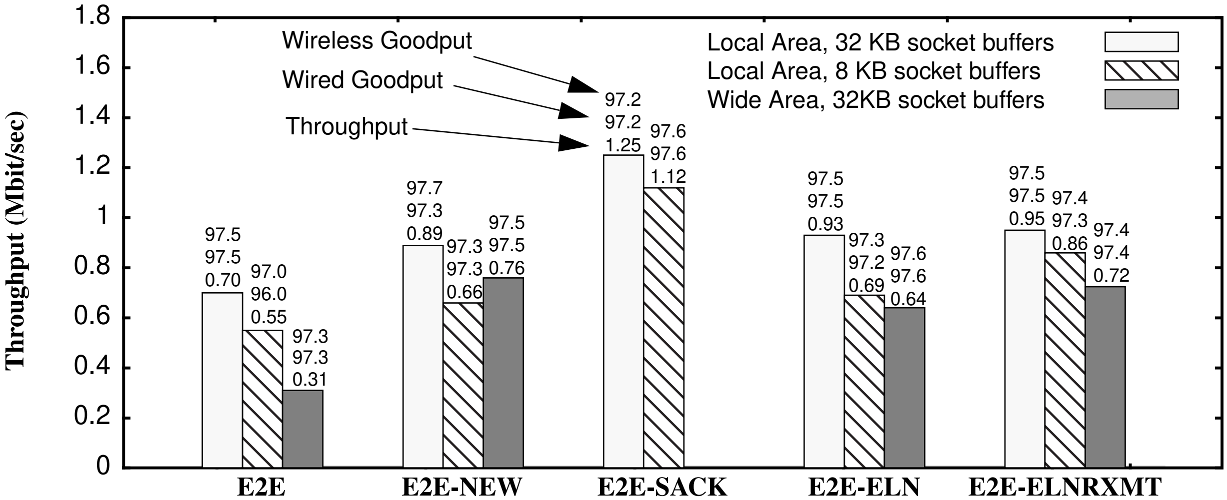

Figure 3. Performance of end-to-end protocols: bit error rate = 1.9×10^{-6} (1 error/65536 bytes).

(ELN), and (v) E2E-ELNRXMIT, an enhancement to E2E-ELN that immediately retransmits missing packets when receiving acknowledgments with the ELN bit set.

The performance of the various end-to-end protocols is summarized in Figure 3. The performance of unmodified TCP Reno, the baseline E2E protocol, highlights the problems with TCP over lossy links. With a 2.3% packet loss rate (as explained in Section 3.1.1), the E2E protocol utilizes less than 50% of the available bandwidth in the local area and less than 25% of the available bandwidth in the wide area experiments. However, all the end-to-end protocols achieve goodputs close to the optimal value of 97.7%. The primary cause for the low bandwidth is the large number of timeout-based retransmissions that occur during the transfer and the small average window size during the transfer that prevents the "data pipe" from being kept full and reduces the effectiveness of the fast retransmission mechanism.

The modified end-to-end protocols improve throughput by retransmitting packets known to have been lost on the wireless hop earlier than they would have been by the baseline E2E protocol, and by reducing the fluctuations in window size. The E2E-NEWRENO, E2E-ELN and E2E-SACK each use new TCP options and more sophisticated acknowledgment processing techniques to improve the speed and accuracy of identifying and retransmitting lost packets. E2E-NEWRENO, which uses partial acknowledgment information to recover from multiple losses in a window at the rate of one packet per round-trip time, performs between 10 and 25% better than E2E over a LAN and about 2.3 times better than E2E in the WAN experiments.

One way of eliminating long delays caused by coarse timeouts is to maintain as large a window size as possible. E2E-NEWRENO remains in fast recovery if the new acknowledgment is only partial, but reduces the window size to half its original value upon the arrival of the first new acknowledgment. The E2E-ELN and E2E-ELN-RXMT protocols use ELN (Explicit Loss Notification) information to prevent the sender from reducing the size of the congestion window in response to a wireless loss. Both these schemes perform better than E2E-NEWRENO, and over two times better than E2E. This is because there is usually enough data in the pipe to trigger a fast retransmission for E2E-ELN. The performance benefits of E2E-ELN-RXMT are more pronounced when the socket buffer size is smaller, as the numbers for the 8 KB socket buffer size indicate.

Finally, we also experimented with a simple SACK scheme based on a subset of the SMART proposal [16] in the local area. This protocol was the best of the end-to-end protocols in this situation, achieving a throughput of 1.25 Mbps (in contrast, the best local scheme, LL-OPT, obtained a throughput of 1.40 Mbps).

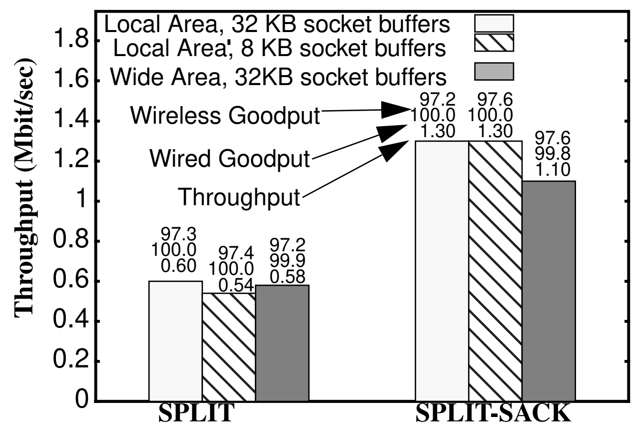

Figure 4. Performance of split-connection protocols: bit error rate = 1.9×10^{-6} (1 error/65536 bytes).

In summary, E2E-NEWRENO is better than E2E, especially for large socket buffer sizes. Adding ELN to TCP improves throughput significantly by successfully preventing unnecessary fluctuations in the transmission window. Finally, SACKs provide significant improvement over TCP Reno, but perform about 10-15% worse than the best local schemes in the LAN tests.

3.2 Split-Connection Protocols

In this section, we examine two algorithms: (i) SPLIT, where the connection is split at the base station into two TCP Reno connections, and (ii) SPLIT-SACK, where the connection is also split at the base station into two TCP connections, but the wired connection uses TCP Reno and the wireless connection uses TCP Reno enhanced with selective acknowledgments. The main advantage of the split-connection approaches is that they isolate the TCP source from wireless losses. The TCP sender of the second, wireless connection performs all the retransmissions in response to wireless losses.

Figure 4 shows the throughput and goodput for the split connection approach in the LAN and WAN environments. We see that the throughput achieved by the SPLIT approach (0.6 Mbps) is quite low, about the same as that for end-to-end TCP Reno (labeled E2E in

Figure 3). This is because a split-connection approach (eventually) stalls whenever the sender of the wireless connection experiences a timeout, since the amount of buffer space at the base station (64 KB in our experiments) is bounded. In the WAN case, the throughput of the SPLIT approach is about 0.58 Mbps which is significantly better than the 0.31 Mbps that the E2E approach achieves (Figure 3), but not as good as several other protocols described earlier. The large congestion window size of the wired sender in SPLIT enables a higher bandwidth utilization over the wired network, compared to an end-to-end TCP connection where the congestion window size fluctuates rapidly.

As expected, throughput for the SPLIT-SACK scheme is much higher. It is about 1.3 Mbps in the LAN case and about 1.1 Mbps in the WAN case. The SMART-based selective acknowledgment scheme operating over the wireless link performs very well, especially since no reordering of packets occurs over this hop. However, there are a few times when both the original transmission and the first retransmission of a packet get lost, which sometimes results in coarse timeouts. This explains the difference in throughput between the SPLIT-SACK scheme and the LL-OPT scheme (Figure 2).

In summary, while the split-connection approach results in good throughput if the wireless connection uses some special mechanisms, the performance does not exceed that of a well-tuned, TCP-aware link-layer protocol (LL-OPT). Moreover, the link-layer protocol preserves the end-to-end semantics of TCP acknowledgments, unlike the split-connection approach. This demonstrates that the end-to-end connection need not be split at the base station in order to achieve good performance.

In the next section, we focus on improving TCP performance when the wireless link exhibits asymmetric properties.

4. TCP over Asymmetric Networks

In this section, we focus on the challenges to end-to-end TCP performance that arise due to network asymmetry, especially in the context of wide-area wireless networks. The increased interest in asymmetric networks is motivated by technological and economic considerations as well as by popular applications such as Web access, where substantially more data flows towards the client (the *forward* direction) than from it (the *reverse* direction).

Examples of networks that exhibit asymmetry include *wireless cable modem* networks, *Direct Broadcast Satellite* (DBS) networks, and *Asymmetric Digital Subscriber Loop* (ADSL) networks, where bandwidths in the forward direction are often orders of magnitude higher than in the reverse. Such asymmetry is accentuated when the channel is uni-directional, necessitating the use of a different, often low-bandwidth channel (e.g., a dialup line or a wireless channel) for communication in the reverse direction.

As a general definition, we say that a network exhibits asymmetry with respect to TCP performance if the throughput achieved in one direction depends significantly on the link and traffic characteristics in the other direction. We focus on *bandwidth asymmetry* here, where the bandwidth in the forward direction is greater than the bandwidth in the reverse direction. We also briefly discuss asymmetry in *latency* and *media access*. A more detailed treatment appears in [3].

The simulation topology we used to investigate the effects of bandwidth asymmetry is shown in Figure 5. In the following sub-sections, we discuss several performance problems that we observed based on experiments conducted in a real testbed as well as in a simulator. We then discuss some solution techniques and evaluate their efficacy via simulations.

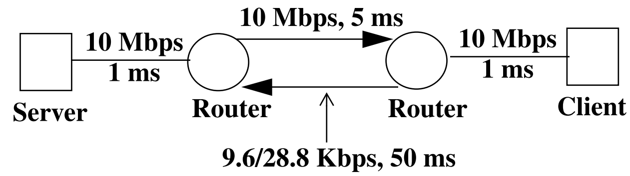

Figure 5. The simulation topology used to model a network with bandwidth asymmetry. The bandwidth and delay parameters have been chosen to closely

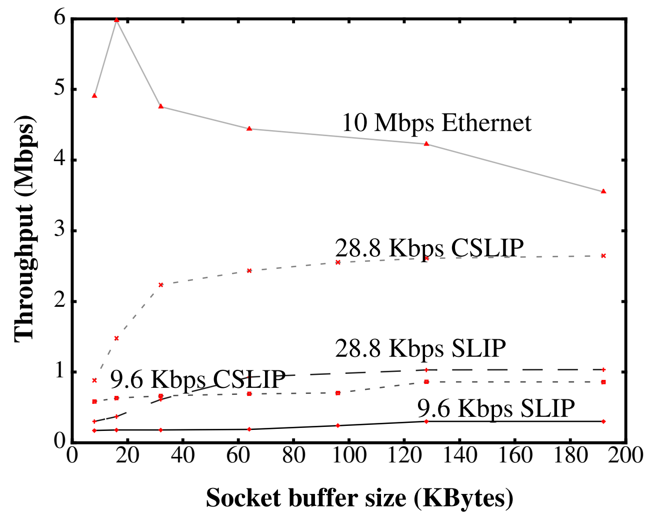

Figure 6. Performance of the wireless cable system using different return channels, across a range of socket buffer sizes.

4.1 Problems due to Bandwidth Asymmetry

The main problem with bandwidth asymmetry for unidirectional transfers is that the constrained reverse bandwidth can prevent TCP from achieving the full bandwidth of the forward path because of the slow arrival of acks. This is because TCP ack clocking, required for the connection to make steady progress, can break down. For example, consider two data packets transmitted by the sender in quick succession. While in transit to the receiver, these packets get spaced apart according to the bottleneck link bandwidth in the forward direction. The principle of ack clocking is that the acks generated in response to these packets retain the same spacing (in time) all the way back to the sender, enabling it to clock out new data packets with the same spacing.

However, the limited reverse bandwidth and consequent queuing effects could alter the inter-ack spacing. When acks arrive at the bottleneck link in the reverse direction at a faster rate than the link can support, they get queued behind one another. The spacing between them when they emerge from the link is dilated with respect to their original spacing. Thus the sender clocks out new data at a slower rate than if there had been no queuing of acks. Further, a point is reached when the reverse channel buffer gets filled up with acks. From that point on, acks often get dropped. As a consequence, the sender becomes bursty and its window growth is slowed down.

Figure 6 shows TCP throughput measurements from the Hybrid wireless cable modem network with various reverse channels. The x-axis shows the socket buffer size and the y-axis shows the throughput in Mbps. We see that ack dilation limits the throughput for dialup reverse channels without SLIP header compression. Header compression allows more acknowledgments to traverse the bottleneck link in the backward direction, increasing performance. The performance with an Ethernet return channel is much better because of the absence of bandwidth asymmetry and much lower link delay than the dialup lines.

4.2 Solutions

The problems discussed above arise because of contention for the bottleneck resources in the reverse direction — link bandwidth and buffer space. This observation serves as the starting point for the solutions discussed below. We present two techniques — ack congestion control and ack filtering — for alleviating the effects of congestion of ack packets on the reverse channel.

4.2.1 Ack Congestion Control (ACC)

TCP acknowledgments impose non-negligible demands on resources at the low-bandwidth bottleneck link in the reverse direction. Acks occupy slots in the reverse channel buffer, whose capacity is often limited to a certain number of *packets* (rather than bytes), as is the case in our systems. This motivates a scheme which extends congestion control mechanisms to TCP acks which we describe in this section.

Our approach is to use the RED (*Random Early Detection*) algorithm [13] at the gateway of the reverse link to aid congestion control. The gateway detects incipient congestion by tracking the average queue size over a time window in the recent past. If the average exceeds a threshold, the gateway selects a packet at random and marks it, i.e. sets an *Explicit Congestion Notification* (ECN) bit using the RED algorithm[2]. This notification is reflected to the sender of the packet by the receiver. Upon receiving a packet with ECN set, the sender reduces its sending rate.

The important point to note is that with ACC, *both* data packets and TCP acks are candidates for being marked. The TCP receiver maintains a dynamically varying delayed-ack factor, d, and sends one ack for every d data packets. When it receives a packet with the ECN bit set, it increases d multiplicatively, thereby decreasing the frequency of acks also multiplicatively. Then for each subsequent round-trip time (determined using the TCP timestamp option) during which it does not receive an ECN, it linearly decreases the factor d, thereby increasing the frequency of acks. Thus, the receiver mimics the standard congestion control behavior of TCP senders in the manner in which it sends acks.

There are bounds on the delayed-ack factor d. Obviously, the minimum value of d is 1, since at most one ack is sent per data packet. The maximum value of d is determined by the sender's window size, which is conveyed to the receiver in a TCP option. The receiver should send at least one ack for each window of data from the sender. Otherwise, it could cause the sender to stall until the receiver's delayed-ack timer (usually set at 200 ms) causes the resumption of the transfer.

4.2.2 Ack Filtering (AF)

The ACC mechanism described above modifies the TCP stack at the receiver in order to decrease the frequency of acks on the constrained reverse link. Ack filtering, based on an idea suggested by Karn [15], is a gateway-based technique that decreases the number of TCP acks sent over the constrained channel by taking advantage of the fact that TCP acks are cumulative.

When an ack from the receiver is about to be enqueued, the router (or the end-host's routing layer, if the host is directly connected to the constrained link) traverses its queue to check if any previous acks belonging to the same connection are already in the queue. It then removes some fraction (or possibly all) of them, depending on how full the queue is. The removal of these "redundant" acks frees up space for other data and ack packets. The policy that the filter uses to drop packets is configurable and can either be deterministic or random (similar to a random-drop gateway, but using the semantics of the items in the queue). There is no need for any per-connection state to be maintained at the router — all the information necessary to implement the drop policy is already implicitly present in the packets in the queue.

In the experiments reported here, AF deterministically clears out all preceding acks belonging to a connection whenever a new ack for the same connection with a larger cumulative ack value enters the queue.

4.3 Results

We conducted a set of experiments, each involving a 50-second long transfer in the forward direction. There was no traffic in the reverse direction other than the acks for the forward transfer. Table 1 summarizes the throughputs obtained for three protocol configurations — regular TCP Reno, Reno with ACC and Reno with AF — with different types of return channels. With both ACC and AF, we include the sender adaptation techniques and perform traffic shaping to smooth out the spacing of acknowledgments.

The socket buffer size at the sender and receiver was set to 100 KB and each data packet was 1000 bytes long. The buffer size at each router was set to 10 packets. The ack size was set to 6 bytes and 40 bytes, respectively, with and without header compression.

The main observation here is that since the transfers are long, the reverse buffer fills up quickly. Beyond that point, the reverse bottleneck link limits the transfer. The greater the normalized asymmetry ratio (called k) is, the more limited the transfer becomes. In the steady state, only one ack in k gets through on average, causing the sender to send out bursts of k packets. As long as k does not exceed the bottleneck buffer size in the forward direction (which is 10 packets in our topology), the increased burstiness of the sender does not lead to losses.

Reverse Channel Bandwidth	Reno	ACC	AF
9.6 Kbps	1.78	3.64	6.28
9.6 Kbps C	6.67	7.69	7.93
28.8 Kbps	4.35	7.58	9.49
28.8 Kbps C	9.78	9.77	9.88

Table 1. Throughputs (in Mbps) for a single one-way transfer in the forward direction. "C" indicates the use of SLIP header compression.

The factor k exceeds 10 for the cases of SLIP without header compression, which explains the poor throughput of TCP Reno in those cases (1.78 and 4.35 Mbps). The sender adaptation employed in conjunction with ACC and AF smooths out sender bursts, avoiding performance degradation in those cases.

For the 9.6 Kbps reverse channel with header compression, k is 6.25, which is less than 10. Still the throughput obtained with TCP Reno

2. The gateway can also be configured to drop the selected packet (Random Early Drop), but we chose to mark it instead.

(6.67 Mbps) is worse than that for the other schemes. This happens because the reverse channel buffer gets filled with acks (totalling 10*6 = 60 bytes), which adds a significant delay (60*8/9.6 = 50 ms) to the connections round-trip time (RTT). The same effect also explains why the performance with ACC is somewhat worse than that with AF in both the 9.6 Kbps and 28.8 Kbps cases. The former only tries to ensure that the reverse channel queue does not get completely filled up. The latter ensures that there is not more than one ack per connection in the queue, which minimizes the effect of queuing on round-trip time.

To summarize, TCP Reno suffers performance degradation when *k* is large and there is significant queuing delay. ACC and AF alleviate these problems by decreasing the frequency of acks.

4.4 Summary of Other Results

4.4.1 Two-Way Traffic

The presence of two-way traffic aggravates the problems due to bandwidth asymmetry discussed in Section 4.1. The transfer in the reverse direction uses up part of the limited reverse channel bandwidth. In addition, large data packets of the reverse transfer could cause acks of the forward transfer to be queued behind them for significant periods of time, further worsening performance.

Our solution to this is to have priority scheduling of acks (a*cks-first* scheduling) at the reverse channel router. This minimizes the amount of time for which the ack stream is stalled. Our results show that combination of ACC and acks-first scheduling results in good performance in both directions.

4.4.2 Latency and Media-Access Asymmetry

In packet radio networks such as Metricom's Ricochet network *(http://www.metricom.com)*, there may not be asymmetry in bandwidth, but the significant and variable latency for sending each packet causes problems. The presence of interfering traffic, including TCP acks, has an adverse effect on TCP data throughput because of variable latencies and large variations in round-trip time estimates. Our simulation experiments demonstrate that using ack filtering to decrease the frequency of acks results in significant improvement in both throughput for individual transfers as well as fairness when multiple connections are simultaneously active.

In the next section, we focus on how to improve TCP performance in the context of Web traffic.

5. Interactions of TCP and Web Traffic[5]

In the paper, we analyze the way in which Web browsers use TCP connections based on extensive traffic traces obtained from a busy Web server (the official Web server of the 1996 Atlanta Olympic games). At the time of operation, this Web server was one of the busiest on the Internet, handling tens of millions of requests per day from hundreds of thousands of clients. We present a detailed analysis of TCP's loss recovery and congestion control behavior from the recorded transfers. The two most important results are that: (1) short web transfers lead to poor loss recovery performance and (2) the use of parallel connections leads to overaggressive use of the network. We then discuss techniques designed to solve these problems. To improve the loss recovery performance of short transfers, we present a new technique for TCP loss recovery. To improve the congestion control and loss recovery performance of multiple parallel connections, we present a new transport-level *integrated connection* approach to congestion control and loss recovery. Simulation and trace analysis results show that our enhanced loss recovery scheme could have eliminated 25% of all timeout events, and our integrated connection approach provides greater fairness and improved startup performance for parallel connections. Our solutions are more general than a specific protocol enhancement such as the use of persistent connections in P-HTTP [19] and HTTP/1.1 [12], and include techniques, such as improved TCP loss recovery, that are not addressed by these protocols. Although the full paper presents a detailed analysis of connection behavior, we only discuss here the solutions to the problems found in the analysis.

5.1 Improving Single Connection Performance

5.1.1 Enhanced TCP Loss Recovery

Our analysis found that over 50% of all retransmissions on the Olympic Web server happened after one or more coarse timeouts kept the link to the client idle for periods from hundreds of milliseconds to seconds. Our analysis showed two main reasons for the occurrence of these timeouts:

1. Fast retransmit followed by a timeout: The TCP Reno sender was unable to recover from multiple losses within the same transmission window. This situation can be recognized by the occurrence of a fast retransmission shortly before the coarse timeout.
2. Insufficient duplicate acknowledgments: Either the number of outstanding packets was too small, or most of the packets in the window were lost, preventing the sender from receiving enough acknowledgment information to trigger a retransmission.

The use of TCP selective acknowledgments (SACKs) has often been suggested as a technique to improve loss recovery and avoid timeouts unless there is glan.tclenuine persistent congestion in the network. However, SACKs can only provide information to help recovery from multiple losses in the same window. A detailed analysis of the trace data showed that out of the 422025 coarse timeouts over a 3 hour period, SACKs could have helped avoid at most 18713 (4.43%) of them. *In other words, current approaches to TCP Reno enhanced with SACKs do not really avoid most timeouts*. It is clear that an alternative technique is needed to recover from the bulk of these losses.

During the same 3 hour period, approximately 403312 (95.6%) coarse timeouts occurred as a result of insufficient acknowledgment information (i.e., an insufficient number of duplicate acknowledgments arrived to trigger a retransmission). Of these timeouts, no duplicate acknowledgments arrived at all for 70% of them. In these situations the network is most likely experiencing severe congestion. The best solution is for the sender to wait for a coarse timeout to occur before transmitting any packets. For the remaining timeouts (about 25% of them, in which at least a single duplicate acknowledgment arrives), we propose that a single new segment, with a sequence number higher than any outstanding packet, be sent when each duplicate acknowledgment arrives. When this packet arrives at the receiver, it will generate an additional duplicate acknowledgment. When this acknowledgment later arrives at the sender, it can be assured that the appropriate segment has been lost and can be retransmitted. We call this form of loss recovery *enhanced* or *"right-edge"* recovery.

5.1.2 Results

An ns [18] based simulation was performed to test the enhanced loss recovery algorithm. A significant amount of additional cross traffic

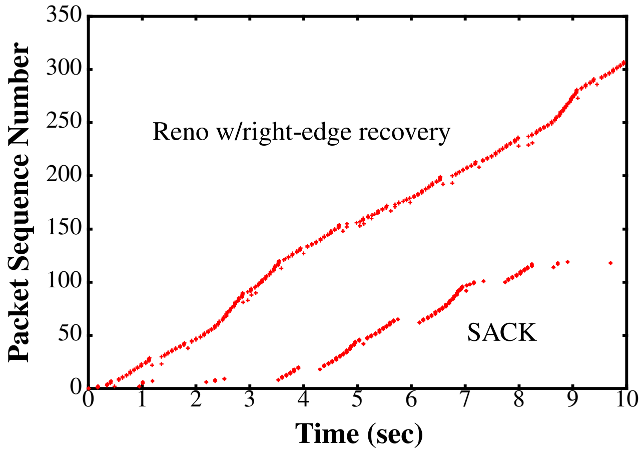

Figure 7. TCP with and without right-edge recovery.

was generated to force the transfers through the router to cope with frequent losses and small congestion windows. This was done to recreate situations that occur in the traces in a controlled fashion and not to simulate any existing or typical network topology. The simulation tests consisted of a single TCP transfer from node S to node R for a duration of 10 seconds. Each test used a different variant of TCP sender protocol on node S.

Figure 7 shows the simulated sequence plots for TCP with SACK and our enhancement to TCP-Reno with right-edge recovery. As indicated by the large gaps in the sequence plot, the SACK transfer experiences many more coarse timeouts than the transfer using right-edge recovery. These coarse timeouts are a result of frequent losses that generate two or fewer duplicate acknowledgments. These duplicates are not sufficient to trigger the fast retransmission algorithm. Therefore, in a traditional TCP implementation the loss is only retransmitted after a coarse timeout occurs. When the right-edge recovery algorithm is used, the sender transmits a new packet when these duplicate acknowledgments arrive. These new packets trigger the transmission of additional duplicate acknowledgments from receiver. These additional duplicates allow the sender to use fast retransmissions to recover from the losses. By eliminating the coarse timeouts, the sender using right-edge recovery performs the transfer more than twice as fast as a standard transmitter.

5.2 Improving Performance of Multiple TCP Connections

Current applications often use multiple logically separate data streams between a Web server and client. We showed in the analysis that the use of independent TCP connections for each transfer has a significant negative impact on network behavior. In this section, we present the design and implementation of a scheme that eliminates the negative impact of concurrent TCP connections and actually improves TCP loss recovery in such situations. This modified TCP uses integrated congestion control and loss recovery that exploits temporal parallelism and shares state across concurrent connections.

5.2.1 Integrated Congestion Control/Loss Recovery

The motivation behind kernel-level integrated congestion control and loss recovery is to allow applications to use a separate TCP connection for each transfer (just as they do today), but to avoid the problems that normally arise from multiple connections by making appropriate modifications to the network stack. We divide TCP functionality into two categories: that having to do with the reliable, ordered byte-stream abstraction of TCP and that pertaining to congestion control and data-driven loss recovery.

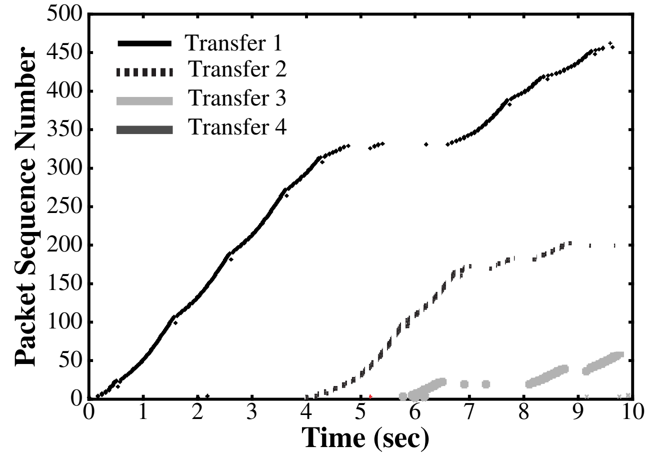

Figure 8. Four TCP-SACK transfers through a router with buffer size 3. Transfers start at 0, 2, 4 and 6 seconds.

By using a separate TCP connection for each transfer, an application can avoid unnecessary coupling between parallel transfers. Each connection provides a reliable, ordered byte-stream abstraction independently of the others. The flow control for each connection happens independently of the others, so the delivery of data to the receiving application also happens independently for each connection.

At the same time, congestion control is integrated across the TCP connections. There is a single congestion window for the set of TCP connections between a client and a server which determines how much total outstanding data the set of connections can have in the network. When a loss occurs on any of the connections, the combined congestion window is halved, thereby achieving the same effect as when a single, persistent TCP connection is used.

Data-driven loss recovery is also integrated across the set of TCP connections. When a packet is lost on one connection, the successful delivery of later packets on other connections allows the sender to reliably detect the packet loss without resorting to a timeout, thereby improving performance.

With such a network stack-based solution, existing unchanged applications can set up as many (possibly short) TCP connections as they wish without having an adverse effect on either congestion control or loss recovery.

5.2.2 Simulation Results: One Client Host Case

In this section, we describe results from an ns simulation designed to examine integrated congestion control and loss recovery across simultaneous TCP connections.

In the first test, the router buffers size was set to 3 packets. This is small enough to force transfers to have small congestion windows and experience frequent losses. Once again, the topology and parameters were chosen to recreate situations that frequently occur in our traces but not necessarily to mimic an actual network. In this test, the transmitting node performs 4 TCP transfers to the receiver. The transfers start at 0, 2, 4 and 6 seconds and all end at 10 seconds. The actual choices of the values 0, 2, 4, and 6 are not important, just that the start times of each connection are slightly shifted in time.

Figure 8 shows the sequence plot for the test using a selective acknowledgment based transmitter. It shows that typically only one

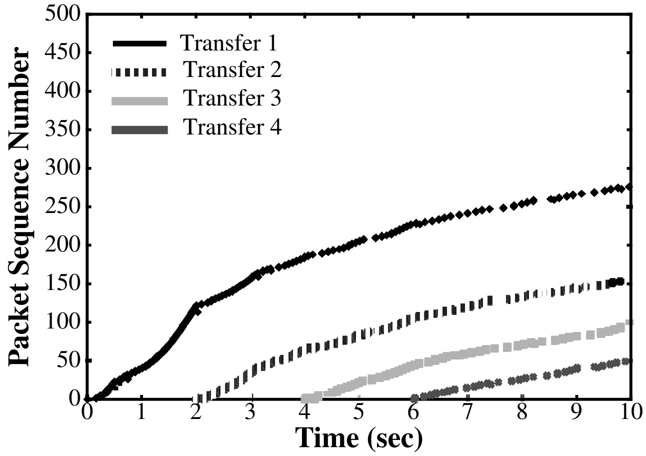

Figure 9. Four TCP-INT transfers through a router with buffer size 3. Transfers start at 0, 2, 4 and 6 seconds.

connection performs satisfactorily at any one time. For example, at time 2 seconds, the connection starting up experiences several early losses and is forced to recover them via coarse timeouts. In fact, this connection does not send a significant amount of data until 4 seconds later (at time 6 sec). Similarly, the first connection experiences many losses at time 4 seconds and spends several seconds recovering from these losses. Over the 10 second period, the connection starting at time 2 sec. and time 6 sec. account for a minuscule (<10%) fraction of the total bytes transferred. Although the combination of the four connections transmit data at a reasonable rate, the bandwidth is clearly not evenly split among the active connections. In addition, the numerous coarse timeouts make the performance of individual connections unpredictable. From an application's point of view, such unpredictability is undesirable because it may happen that connections carrying critical data get slowed down while others carrying less important data do better.

Figure 9 shows the sequence plot for the same test with the senders using TCP-Reno with integrated congestion control and loss recovery (henceforth called TCP-INT). Integrated loss recovery helps this TCP variant avoid resorting to coarse timeouts. Integrated congestion control allows the different connections to each obtain an equal share of the total bandwidth. Although the total number of bytes transferred here is actually slightly less than in the case with the selective acknowledgment protocol, the performance of the transfers is much more consistent and predictable. Also note that since the slow-start window growth happens in an integrated manner for the ensemble of connections, later connections can build on the window size attained by earlier connections rather than initiate slow start with an initial window size of one segment (which would happen with unmodified TCP). This can greatly help reduce the connection start-up latency for Web clients with short overlapping parallel connections.

5.2.3 Simulation Results: Multiple Client Hosts Case

We now investigate how the bottleneck link bandwidth is shared by connections initiated from more than one host. Our test uses the network topology shown in Figure 10. At time 0, a single TCP transfer is started from node S to each of nodes R1 and R2. Some time later (in this case, fifteen seconds), a second transfer is started between node S and R2. In addition, several cross traffic connection are maintained between nodes CT1 and CT2. This is intended to make the simulation more realistic.

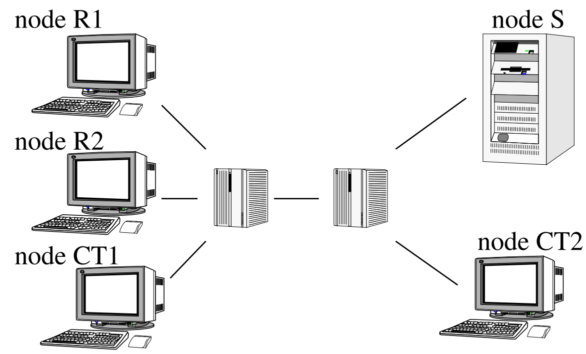

Figure 10. Topology for simulation tests.

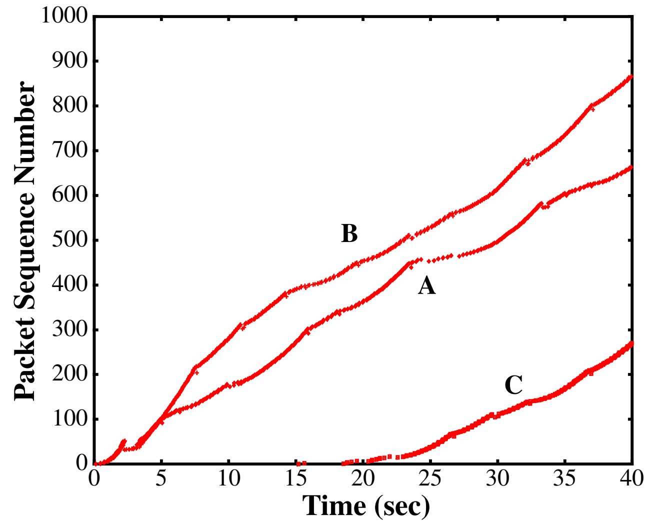

Figure 11. Three TCP-SACK transfers from two hosts through a single bottleneck router. Connection A originates from the first host and starts at time 0. Connections B & C originate at the second host and start at time 0 and 15 sec respectively.

Figure 11 shows the sequence plot for a test where node S uses TCP with selective acknowledgments. This graph shows that each connection receives approximately the same share of the bottleneck link bandwidth. As a result, node R2 receives approximately twice the bandwidth of node R1 after the second connection starts up. This is because each *connection* independently performs the same congestion control algorithms and are each equally likely to experience congestion. Figure 12 shows the same test using TCP-INT. After the second connection from node R2 starts, each of the transfers from R2 receive approximately half the bandwidth of the transfer on node R1. This is because each *host* independently performs the same congestion control algorithm. Therefore, each host receives an equal portion of the bottleneck link.

6. Conclusions

In this paper, we have presented improvements to TCP to improve performance in the context of wireless networks, asymmetric networks, and Web traffic. To improve the performance of TCP over wireless networks, we have implemented a TCP-aware link layer protocol called the snoop protocol that isolates wired senders from the lossy characteristics of a wireless link. To improve TCP performance over asymmetric networks, we have implemented techniques that reduce the effects of bandwidth asymmetry, when the forward path of a link has a higher link bandwidth than the reverse path. To improve

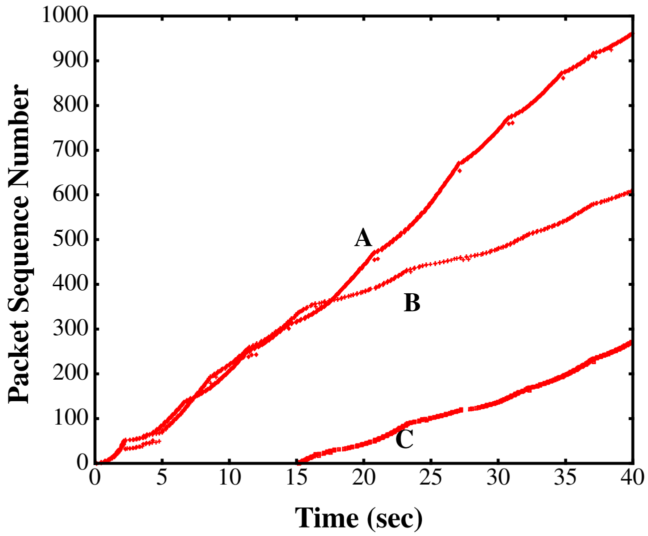

Figure 12. Three TCP-INT transfers from two hosts through a single bottleneck router. Connection A originates from the first host and starts at time 0. Connections B & C originate at the second host and start at time 0 and 15 sec respectively.

TCP performance in the context of Web traffic, we have implemented an improved loss recovery scheme that improves the performance of short transfers and an integrated connection scheme that improves the loss recovery and congestion control performance of parallel TCP connections from a single machine.

7. Acknowledgments

This work is supported by DARPA contract DAAB07-95-C-D154 and grants from the California MICRO Program, Hughes Aircraft Corporation, and IBM. Hari is partially supported by a Research Grant from the Okawa Foundation, and Mark is supported by an IBM fellowship.

8. References

[1] E. Amir, H. Balakrishnan, S. Seshan, and R. H. Katz. Efficient TCP over Networks with Wireless Links. In *Proc. Fifth IEEE Workshop on Hot Topics in Operating Systems*, May 1995.

[2] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proc. 15th International Conf. on Distributed Computing Systems (ICDCS)*, May 1995.

[3] H. Balakrishnan, V. N. Padmanabhan, and R.H. Katz. The Effects of Asymmetry on TCP Performance. In *Proc. ACM MOBICOM '97*, September 1997.

[4] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R.H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *Proc. ACM SIGCOMM '96*, August 1996.

[5] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. Technical report, University of California at Berkeley, 1997.

[6] H. Balakrishnan, S. Seshan, E. Amir, and R.H. Katz. Improving TCP/IP Performance over Wireless Networks. In *Proc. 1st ACM Conf. on Mobile Computing and Networking*, November 1995.

[7] H. Balakrishnan, S. Seshan, and R.H. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, 1(4), December 1995.

[8] R. T. Braden. *Requirements for Internet Hosts – Communication Layers*. Information Sciences Institute, Marina del Rey, CA, October 1989. RFC-1122.

[9] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

[10] D. E. Comer and D. L. Stevens. *Internetworking with TCP/IP, Volume II*. Prentice Hall, Englewood Cliffs, N.J, 1994.

[11] The Daedalus Project Home Page. http://daedalus.CS.Berkeley.EDU/, 1995.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, Jan 1997. RFC-2068.

[13] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[14] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM 88*, August 1988.

[15] P. Karn. Dropping TCP acks. Mail to the end-to-end mailing list, February 1996.

[16] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM Transactions on Computer Systems*, 9(4):364–373, November 1991.

[17] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In *Proc. Winter '93 USENIX Conference*, San Diego, CA, January 1993.

[18] ns: UCB/LBNL Network Simulator. http://www-mash.cs.berkeley.edu/ns/, 1997.

[19] V. N. Padmanabhan and J. C. Mogul. Improving HTTP Latency. In *Proc. Second International WWW Conference*, October 1994.

[20] W. R. Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley, Reading, MA, Nov 1994.

[21] AT&T WaveLAN: PC/AT Card Installation and Operation. AT&T manual, 1994.

[22] R. Yavatkar and N. Bhagwat. Improving End-to-End Performance of TCP over Mobile Internetworks. In *Mobile 94 Workshop on Mobile Computing Systems and Applications*, December 1994.

[23] L. Zhang, S. Shenker, and D. D. Clark. Observations and Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proc. ACM SIGCOMM '91*, pages 133–147, 1991.