Everything's got a moral, if only you can find it.

— Alice in Wonderland, Chapter 9, Lewis Carroll

Chapter 5

Cross-Layer Protocol Optimizations

In this chapter, we compare the performance of the Snoop protocol to conventional link-layer protocols and to split-connection approaches. Our goal is to understand the precise reasons for performance improvement in each scheme. In the process, we demonstrate the performance benefits of cross-layer protocol optimizations and transport-awareness in the Snoop protocol. We also show that the split-connection architecture is not the reason for good performance in certain split approaches.

We then discuss enhancements to the Snoop protocol to improve performance in wireless transit networks using the ELN mechanism introduced in the previous chapter. We also describe how the Snoop protocol can be enhanced using selective acknowledgment (SACK) information from the receiver to perform better recovery from burst losses. We conclude this chapter with a discussion of some key lessons learned from our experience with improving reliable transport over error-prone wireless networks.

5.1 Introduction

There are two different approaches to improving TCP performance in error-prone wireless networks. The first approach hides any non-congestion-related losses from the TCP sender and therefore requires no changes to existing sender implementations. Protocols that adopt this approach attempt to make the error-prone link appear as a higher quality link with a reduced effective bandwidth. As a result, most of the losses seen by the TCP sender are caused by congestion. Examples of this approach include wireless links with reliable link-layer protocols (Section 2.5.2), split connection approaches (Section 2.5.3), and TCP-aware link-layer schemes such as the Snoop protocol (Chapter 4). The second approach attempts to make the sender aware of the existence of wireless hops and realize that some packet losses are not due to congestion. The sender can then avoid invoking congestion control algorithms when non-congestion-related losses occur. Examples of this approach include the Explicit Loss Notification (ELN) scheme described in Chapter 4 and approaches that inform the TCP sender of on-going handoffs [90]. Of course, it is possible for a wireless-aware transport protocol to coexist with link-layer schemes.

Each approach admits a number of different protocols. We classify the many protocols into three basic groups, based on their fundamental philosophy: *link-layer protocols, end-to-end protocols* and *split-connection protocols.*

We investigate ARQ-based link-layer protocols whose local retransmissions are tuned to the characteristics of the wireless link to provide significant improvements in performance. However, since the end-to-end TCP connection passes through the error-prone link, the TCP sender may not be fully shielded from wireless losses. Furthermore, if the link-layer protocol operates independently of the end-to-end transport, adverse interactions could occur between the two that degrade performance, as described in Section 2.5.2.

The end-to-end protocols we investigate attempt to make the TCP sender handle losses through the use of two techniques. First, they use some form of SACK to allow the sender to recover from multiple packet losses in a window without incurring a coarse timeout. Second, they attempt to have the sender distinguish between congestion and other forms of losses using the ELN mechanism.

Finally, split-connection protocols completely hide the wireless link from the sender by terminating the TCP connection at the base station. Such schemes use a separate reliable connection between the base station and the destination host. The second connection can use techniques optimized for wireless such as negative or selective ACKs, rather than just TCP Reno.

In this chapter, we evaluate the performance of several link-layer, end-to-end and split-connection protocols using end-to-end throughput and goodput as performance metrics, in both LAN and WAN configurations. In particular, we seek to answer the following specific questions:

1. What combination of mechanisms results in best performance for each class of protocols?
2. How important is it for link-layer schemes to be aware of TCP algorithms to achieve good end-to-end throughput? What is the quantitative benefit of TCP-awareness in the Snoop protocol?
3. How useful are selective ACKs in dealing with error-prone links, especially when losses occur in bursts?
4. Is splitting the end-to-end connection fundamentally important to achieving good wireless TCP performance?

We answer these questions by implementing and evaluating the various protocols in a WaveLAN-based cellular wireless testbed. For each protocol, we measure the end-to-end throughput and goodputs for the wired and wireless paths. For any path (or link), goodput is defined as the ratio of the actual transfer size to the total number of bytes transmitted over that path (Section 3.3). In general, the wired and wireless goodputs differ because of wireless losses and local retransmissions over the wireless link. These metrics allow us to determine the end-to-end performance as well as the transmission efficiency across the network.

We reinforce the results of Chapter 4, showing that transport-aware link protocols like the Snoop protocol lead to very good performance. Our experiments indicate that shielding the TCP sender from duplicate ACKs caused by wireless losses and being aware of TCP messaging improves throughput by at least 10-30% compared to a conventional reliable link-layer protocol in both LAN and WAN topologies. We show that split connections are not fundamental to achieving good performance. Split-connection approaches can be tuned to perform well, but their good performance is not due to the split *architecture*. Rather, their good performance derives from specialized techniques over the wireless hop, and the same techniques can be used without splitting the end-to-end connection. We also demonstrate that a simple SACK scheme and ELN lead to significant performance improvements. In particular, our ELN scheme improves end-to-end throughput by more than 100% compared to TCP Reno, with comparable goodput values.

The rest of this chapter is organized as follows. Section 5.2 presents a taxonomy of the protocols we study. Section 5.3 describes the implementation details of the different protocols in our wireless testbed. Section 5.4 presents the results and analysis of several experiments and discusses the reasons for observed performance in each case. In Section 5.5, we present enhancements to the Snoop protocol to handle wireless transit links and burst losses. We discuss some key lessons learned about protocol design for error-prone networks in Section 5.6 and conclude with a summary in Section 5.7.

5.2 Taxonomy

Table 5.1 summarizes the key ideas in each protocol and the main differences between them. Each protocol reacts to these losses and recovers from them in different ways.

Name	**Category**	**Special Mechanisms**
E2E	end-to-end	Standard TCP-Reno
E2E-NEWRENO	end-to-end	TCP Newreno
E2E-SMART	end-to-end	SMART-based selective ACKs
E2E-IETF-SACK	end-to-end	IETF selective ACKs
E2E-ELN	end-to-end	Explicit Loss Notification (ELN)
E2E-ELN-RXMT	end-to-end	ELN with retransmit on first dup ACK
LL	link-layer	Local retransmissions
LL-TCP-AWARE	link-layer	Local retransmissions + dup ACK suppression
LL-SMART	link-layer	SMART-based selective ACKs
LL-SMART-TCP-AWARE	link-layer	SMART and dup ACK suppression
SPLIT	split-connection	none
SPLIT-SMART	split-connection	SMART-based wireless connection

Table 5.1 Summary of protocols investigated in this chapter.

5.2.1 Link-Layer Schemes

Unlike TCP for the transport layer, there is no de facto standard for reliable link-layer protocols. Our base link-layer protocol, called LL, uses cumulative ACKs to detect lost packets and then retransmit them from the base station to the mobile host. To minimize overhead, our implementation of LL uses TCP ACKs instead of generating its own. Timer-driven retransmissions are done based on a smoothed round-trip time estimate, with a minimum timeout granularity of 200 ms to limit the overhead of processing timer events. This permits LL to retransmit packets several times before a typical TCP Reno sender times out. LL is logically equivalent to the Snoop agent that does not suppress any duplicate ACKs and does not attempt in-order delivery of packets across the link when packets are lost.

While the use of TCP ACKs by our LL protocol renders it atypical of traditional ARQ protocols, it still preserves the key feature of such protocols: the ability to retransmit packets locally, independently of, and on a faster time scale than TCP. In practice, we find that LL is effective in performing quick local retransmissions of lost segments and preventing timeouts at the sender. Therefore, we expect the qualitative aspects of our results to be applicable to general ARQ-based link-layer protocols.

We also investigate a more sophisticated link-layer protocol (LL-SMART) that uses selective retransmissions to improve performance. The LL-SMART protocol uses a SMART-based selective ACK scheme (Section 2.3.1) at the link layer. Like the LL protocol, LL-SMART uses TCP ACKs instead of generating its own and limits its minimum timeout to 200 ms. LL-SMART is equivalent to the Snoop agent performing retransmissions based on SACKs but not suppressing duplicate ACKs at the base station.

We added TCP awareness to both the LL and LL-SMART protocols, resulting in the LL-TCP-AWARE and LL-SMART-TCP-AWARE schemes. The LL-TCP-AWARE protocol is identical to the Snoop protocol, while the LL-SMART-TCP-AWARE protocol uses SMART-based techniques for further optimization using selective retransmissions. LL-SMART-TCP-AWARE is the best link-layer protocol in our experiments (and in fact the best-performing protocol amongst those we considered): it performs local retransmissions based on selective ACKs and shields the sender from duplicate ACKs caused by wireless losses.

5.2.2 End-To-End Schemes

Although a wide variety of TCP variants are used in the Internet, TCP Reno is the current de facto standard for TCP implementations (Section 2.2). We call this the E2E protocol and use it as the standard basis for performance comparison.

The E2E-NEWRENO [55] protocol improves the performance of TCP-Reno after multiple packet losses in a window by remaining in fast recovery mode if the first new ACK received after a fast retransmission is "partial," i.e, is less than the value of the last byte transmitted when the fast retransmission was done. This protocol is described in Section 2.3.2.

The E2E-SMART and E2E-IETF-SACK protocols add SMART-based and IETF SACKs (Section 2.3.1) respectively to TCP Reno. This allows the sender to handle multiple losses within a window of outstanding data more efficiently than TCP Reno. However, the sender still assumes that losses are a result of congestion and invokes congestion control procedures, shrinking its congestion window size. This allows us to identify what fraction of the end-to-end performance degradation is associated with standard TCP's error detection and retransmission.

In SMART [76], the receiver sends information about which data segment caused the duplicate cumulative ACK to be generated. The sender uses this information to construct a map of which segments in the window are missing and perform the appropriate retransmissions. We used the SMART-based scheme only for the LAN experiments. This scheme is well-suited to situations where there is little reordering of packets, which is true for one-hop wireless systems such as ours. Unlike the scheme proposed in [76], we do not use any special techniques to detect the loss of a retransmission. The sender retransmits a packet when it receives a SMART ACK only if the same packet was not retransmitted within the last round-trip time. If no further SMART ACKs arrive, the sender reverts to the coarse timeout mechanism to recover from the loss. We experimented with the IETF SACK scheme in both LAN and WAN settings. Our implementation is based on the RFC and takes appropriate congestion control actions upon receiving SACK information [11].

The E2E-ELN protocol adds the Explicit Loss Notification (ELN) option to TCP ACKs. When a packet is dropped on the wireless link, future cumulative ACKs corresponding to the lost packet are marked to identify that a non-congestion related loss has occurred. Upon receiving this information with duplicate ACKs, the sender retransmits data without invoking the associated congestion control procedures. This protocol allows us to identify what fraction of the end-to-end performance degradation is associated with TCP's incorrect invocation of congestion control algorithms when it performs a fast retransmission of a packet lost on the wireless hop. The E2E-ELN-RXMT protocol is an enhancement of E2E-ELN, where the sender retransmits the packet on receiving the *first* duplicate ACK with the ELN option set (as opposed to the third duplicate ACK

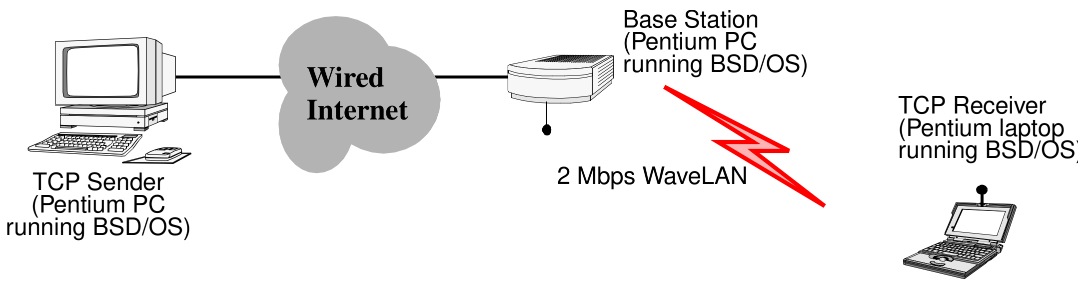

Figure 5.1 Experimental topology. A 10 Mbps Ethernet connected the sender to the base station in the LAN experiments, while there were 16 wired Internet hops between them in the WAN experiments.

in the case of TCP Reno), in addition to not shrinking its window size in response to wireless losses. At the TCP sender, this is similar to the protocol described in Section 4.4.2.

5.2.3 Split-Connection Schemes

We experiment with two split-connection schemes: SPLIT and SPLIT-SMART. The SPLIT scheme uses the base station to divide an end-to-end TCP connection into two separate TCP connections (Section 2.5.3). Our efficient implementation avoids data copying at the base station by passing the pointers to the same buffer between the two TCP connections. The SPLIT-SMART scheme is optimized for good performance over the wireless hop. It uses a SMART-based SACK scheme on the wireless connection to perform selective retransmissions. There is little chance of reordering of packets over the wireless connection since the base station is only one hop away from the final destination; thus all duplicate ACKs signify losses rather than reordering.

5.3 Experimental Methodology

We performed several experiments to compare the performance and efficiency of each of the protocols. The protocols were implemented as a set of modifications to the BSD/OS TCP/IP (Reno) network stack. To ensure a fair basis for comparison, none of the protocols implementations introduce any additional data copying at intermediate points from sender to receiver.

The network topology for our experiments is shown in Figure 5.1. The peak throughput for TCP bulk transfers is 1.5 Mbps in the local area testbed and 1.35 Mbps in the wide area testbed in the absence of congestion or wireless losses. These testbed topologies represent typical scenarios for mobile hosts connected over cellular wireless networks. We focus on data transfer to the mobile host, which is the common case for mobile applications (e.g., Web accesses).

To measure the performance of the protocols under controlled conditions, we emulate errors on the wireless link using an exponentially distributed bit-error model. The receiving entity on the error-prone link generates an exponential distribution for each bit-error rate and changes the IP header 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 ACKs are dropped too. The TCP data packet size in our experiments is 1400 bytes. We first measure and analyze the performance of the various protocols at an average error rate of one every 64 KB (this corresponds to a base bit-error rate of about 1.9×10^{-6}). Because the exponential distribution has a standard deviation equal to its mean, there are several occasions when multiple packets are lost in close succession in the same window. We then investigate the performance of many of these protocols across a range of error rates from one every 16 KB to one every 256 KB.

The choice of the exponentially 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 function of the exact error model, the relative performance is dependent on how the protocol behaves after one or more losses in a single TCP window. Thus, we expect our overall conclusions to be applicable under other patterns of wireless loss as well. Finally, we believe that though wireless errors are generated artificially in our experiments, the use of a real testbed is valuable because it introduces realistic effects such as wireless bandwidth limitation, media access contention, protocol processing delays, etc., that are hard to model realistically in simulation.

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

1.WAN experiments across the United States were performed between 10 pm and 4 am (PST) and we verified that no congestion losses occurred in the runs reported.

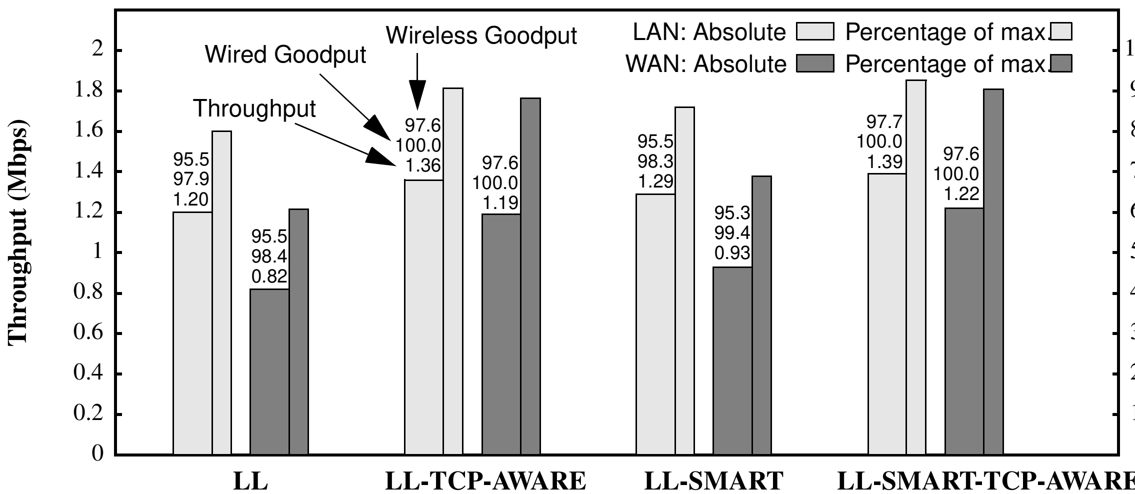

Figure 5.2 Performance of link-layer protocols: bit-error rate = 1.9x10^{-6} (1 error/64 KB), socket buffer size = 32 KB. For each case there are two bars: the thick one corresponds to the scale on the left and denotes the throughput in Mbps; the thin one corresponds to the scale on the right and shows the throughput as a percentage of the maximum, i.e. in the absence of wireless errors (1.5 Mbps in the LAN environment and 1.35 Mbps in the WAN environment).

Each run in the experiment consists of an 8 MByte transfer from the source to receiver across the wired net and the WaveLAN™ link. We chose this long transfer size to limit the impact of transient behavior at the start of a TCP connection. During each run, we measure the throughput at the receiver in Mbps and the wired and wireless goodputs as percentages. In addition, all packet transmissions on the wired network and WaveLAN are recorded for analysis using `tcpdump` [94], and the sender's TCP code instrumented to record events such as coarse timeouts, retransmission times, duplicate ACK arrivals, congestion window size changes, round-trip estimates, etc.

5.4 Results

In this section, we describe the results of our experiments. We focus on understanding and explaining the key reasons for measured performance in each case.

5.4.1 Link-Layer Protocols

Traditional link-layer protocols operate independently of the higher-layer protocol and consequently do not necessarily shield the sender from the error-prone link. In spite of local retransmissions, TCP performance is often sub-optimal due to adverse interactions between independently designed link and transport layers. Section 2.5.2 describes these interactions.

Unlike the analysis in [38], timer interactions are not the dominant effect when link-layer schemes such as LL are used with TCP Reno and its variants. Most 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 cause packets to reach the TCP receiver out-of-order. In response, the TCP receiver generates duplicate ACKs, which causes the sender to invoke fast retransmission and recovery. The result is degraded throughput and goodput, especially if the connection's delay-bandwidth product is larger than a few segments.

	LL	**LL-TCP-AWARE**	**LL-SMART**	**LL-SMART-TCP-AWARE**
LAN (8 KB)	1.20 (95.6%,97.9%)	1.29 (97.6%,100%)	1.29 (96.1%,98.9%)	1.37 (97.6%,100%)
LAN (32 KB)	1.20 (95.5%,97.9%)	1.36 (97.6%,100%)	1.29 (95.5%,98.3%)	1.39 (97.7%,100%)
WAN (32 KB)	0.82 (95.5%,98.4%)	1.19 (97.6%,100%)	0.93 (95.3%,99.4%)	1.22 (97.6%,100%)

Table 5.2 This table summarizes the results for the link-layer schemes for an average error rate of one every 64 KB of data. Each entry is of the form: throughput (wireless goodput,wired goodput). Throughput is expressed in Mbps.

Our results substantiate this claim, as can be seen by comparing the LL and LL-TCP-AWARE results in Figure 5.2 and Table 5.2. For a packet size of 1400 bytes, a bit error rate of 1.9×10^{-6} (1/64 KB) translates to a packet error rate of between 2.2% and 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 three duplicate ACKs. These duplicates eventu-

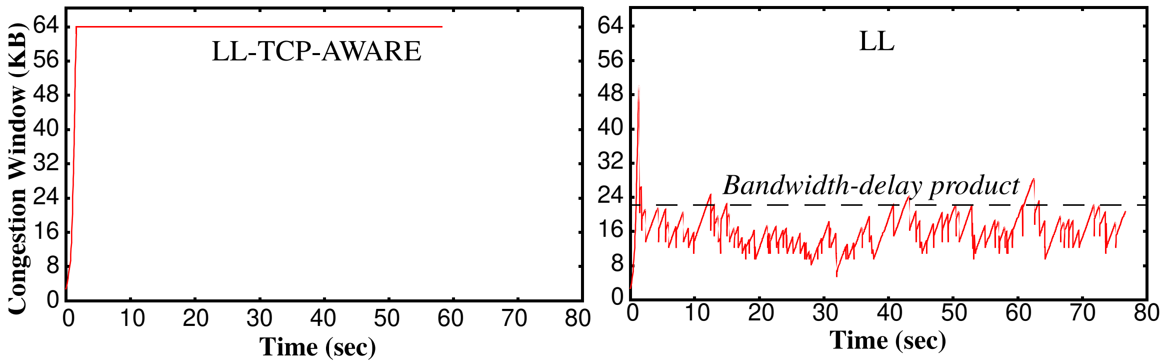

Figure 5.3 Congestion window size for link-layer protocols in the WAN experiments. The horizontal dashed line in the LL graph shows the 23 KB WAN bandwidth-delay product.

ally propagate to the sender and trigger a fast retransmission and the associated congestion control mechanisms. These redundant fast retransmissions degrade goodput; about 90% of the lost packets are retransmitted by both the TCP sender and the base station.

The effects of this interaction are much more pronounced in the wide-area experiments: the end-to-end difference in throughput between LL and LL-TCPAWARE is now about 30%. This pronounced deterioration compared to the LAN case is due to the higher bandwidth-delay product of the wide-area connection. The LL scheme often causes the sender to invoke congestion control due to duplicate ACKs and causes the average window size of the transmitter to be lower than for LL-TCP-AWARE. This is shown in Figure 5.3, which compares the congestion window size of LL and LL-TCP-AWARE as a function of time. Note that the number of outstanding data bytes in the network is the minimum of the congestion window and the receiver advertised window, bounded by the receiver's socket buffer size. In the congestion window graphs for each protocol, the receiver socket buffer is 32KB.

In the wide area, the bandwidth-delay product is about 23 KB (1.35 Mbps * 135 ms), and the congestion window drops below this value several times during each TCP transfer, and its time-averaged mean value is smaller than 23 KB. On the other hand, the LAN experiments do not suffer from such a large throughput degradation because LL's lower congestion-window size is usually still larger than the connection's delay-bandwidth product of about 1900 bytes (1.5 Mbps * 10 ms). Therefore, the LL scheme can maintain a nearly full "data pipe" between the sender and receiver in the local connection but not in the wide area one. The 10% LAN degradation is due to the excessive retransmissions over the wireless link and to the smaller average congestion window size

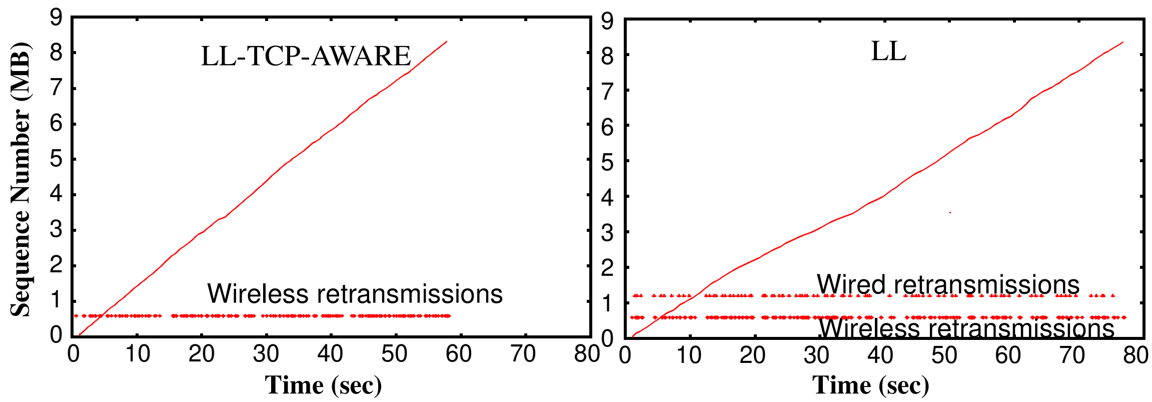

Figure 5.4 Packet sequence traces for LL-TCP-AWARE and LL. No coarse timeouts occur in either case. For LL-TCP-AWARE, the horizontal row of dots shows the times of wireless link retransmissions. For LL, the top row shows sender fast retransmissions and the bottom row shows both local wireless and sender retransmissions.

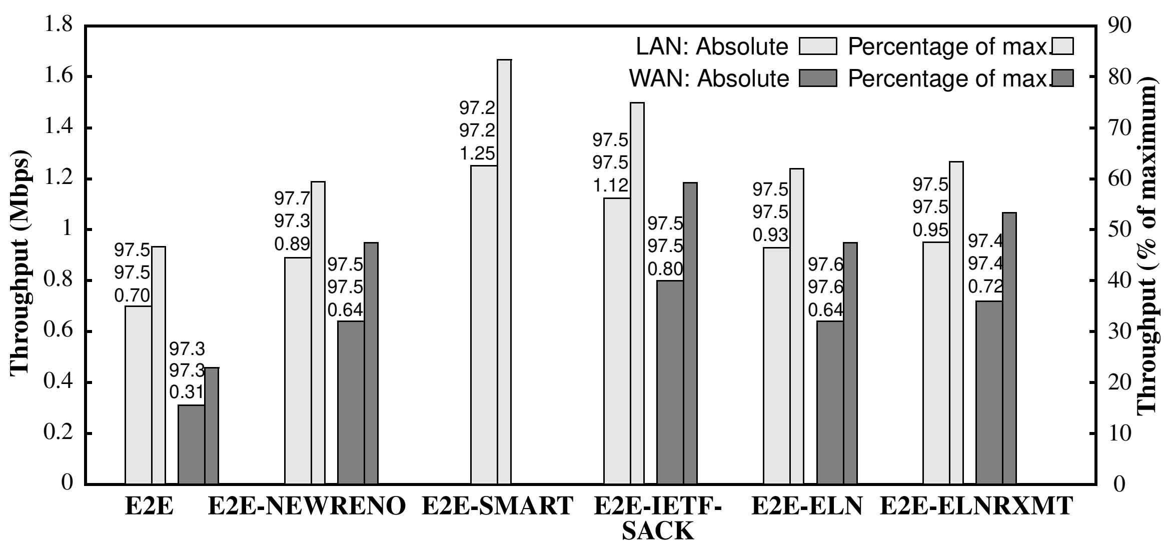

Figure 5.5 Performance of end-to-end protocols: bit error rate = 1.9×10^{-6} (1 error/64KB).

compared to LL-TCP-AWARE. Another important point to note is that LL achieves good local loss recovery and prevents coarse timeouts from occurring at the sender. Figure 5.4 shows the sequence traces of TCP transfers for LL-TCP-AWARE and LL; note that LL-TCP-AWARE has no wired retransmissions.

In summary, our results indicate that link-layer retransmission protocols that are designed independently of the transport layer are prone to adverse interactions with higher-layer mechanisms such as TCP fast retransmissions. These interactions degrade end-to-end performance. A transport-aware link-layer scheme that uses knowledge of TCP semantics to suppress duplicate ACKs caused by wireless losses and locally retransmits packets achieves significantly better performance. This combination of local loss recovery based on TCP ACK monitoring and duplicate ACK suppression is the main reason for the good performance of the Snoop protocol.

5.4.2 End-To-End Protocols

The performance of the various end-to-end protocols is summarized in Figure 5.5 and Table 5.3.

	E2E	E2E-NEWRENO	E2E-SMART	E2E-IETF-SACK	E2E-ELN	E2E-ELN-RXMT
LAN (8 KB)	0.55 (97.0,96.0)	0.66 (97.3,97.3)	1.12 (97.6,97.6)	0.68 (97.3,97.3)	0.69 (97.3,97.2)	0.86 (97.4,97.3)
LAN (32 KB)	0.70 (97.5,97.5)	0.89 (97.7,97.3)	1.25 (97.2,97.2)	1.12 (97.5,97.5)	0.93 (97.5,97.5)	0.95 (97.5,97.5)
WAN (32 KB)	0.31 (97.3,97.3)	0.64 (97.5,97.5)	N.A.	0.80 (97.5,97.5)	0.64 (97.6,97.6)	0.72 (97.4,97.4)

Table 5.3 This table summarizes the results for the end-to-end schemes for an average error rate of one every 64 KB of data. The numbers in the cells follow the same convention as in Table 5.2.

The performance of TCP Reno, the baseline E2E protocol, highlights the problems with TCP over error-prone links. At a 2.3% exponentially-distributed packet loss rate (as explained in Section 5.4.1), E2E achieves a throughput of less than 50% of the maximum (i.e., 50% of the throughput in the absence of wireless losses) in the LAN and less than 25% of the maximum in the WAN experiments. However, all the end-to-end protocols achieve goodputs close to the optimal value of 97.7%. The primary reason for the low throughput is the large number of timeouts that occur during the transfer (Figure 5.6, E2E graph). The resulting average window size during the transfer is small, preventing the “data pipe” from being kept full and greatly reducing the effectiveness of the fast retransmission mechanism (Figure 5.7, E2E graph). In turn, these small congestion windows increase the likelihood of timeouts.

The various end-to-end enhancements 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, E2E-SMART and E2E-IETF-SACK protocols each use new TCP options and more sophisticated ACK processing techniques to improve the speed and accuracy of identifying and retransmitting lost packets, as

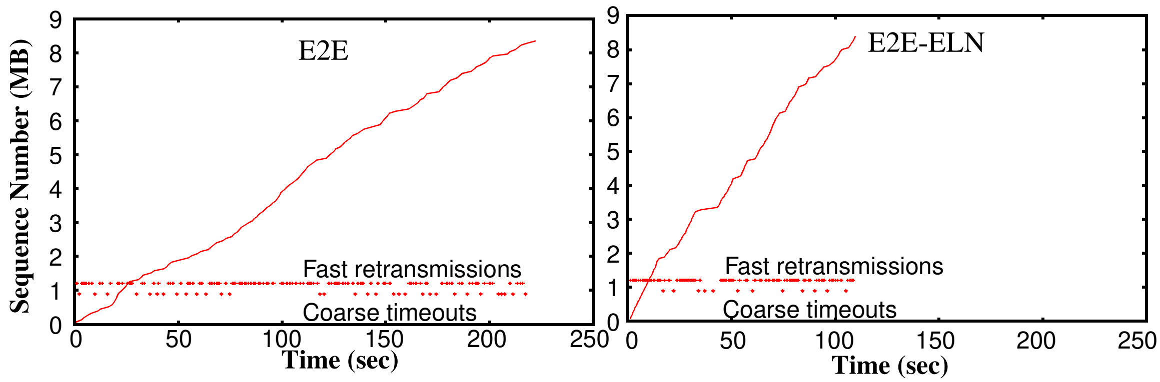

Figure 5.6 Packet sequence traces for E2E (TCP Reno) and E2E-ELN. The top row of horizontal dots shows the times when fast retransmissions occur; the bottom row shows the coarse timeouts.

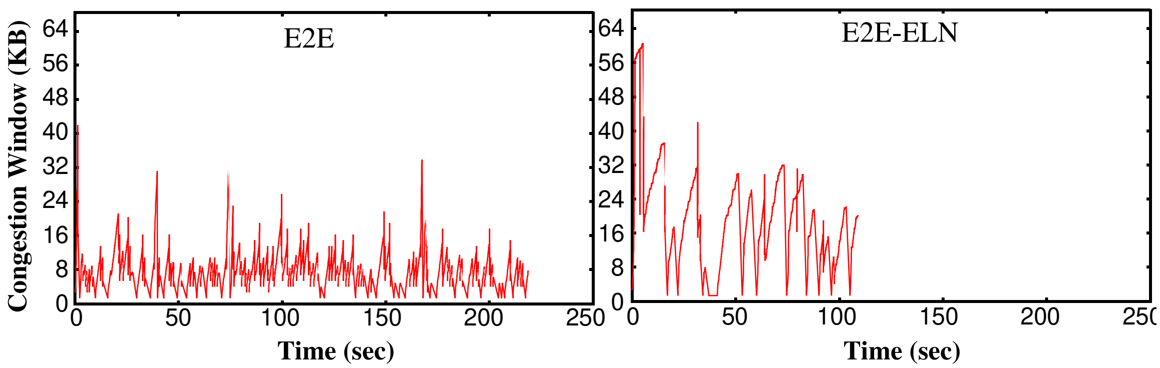

Figure 5.7 Congestion window size as a function of time for E2E (TCP Reno) and E2E-ELN. This figure clearly shows the utility of ELN in preventing rapid fluctuations, thereby maintaining a larger average congestion window size.

well as by recovering from multiple losses in a single transmission window without timing out. The remainder of this section discusses and quantifies the benefits of three techniques: *partial acknowledgments, explicit loss notifications* and *selective acknowledgments.*

Partial ACKs: E2E-NEWRENO, which uses partial ACK information to recover from multiple losses in a window at the rate of one segment per round-trip time, performs between 10% and 25% better than E2E over a LAN and about 100% better than E2E in the WAN experiments. Remaining in fast recovery when a partial ACK arrives reduces the occurrence of timeouts and directly translates to improved performance. The performance improvement is a function of the socket buffer size—the larger the buffer size, the better the relative performance. This is because in situations that E2E suffers a coarse timeout for a loss, the probability that E2E-NEWRENO does not increases with the number of outstanding packets in the network.

ELN: One way of reducing the number of coarse timeouts is to maintain as large a window size as possible. E2E-NEWRENO remains in fast recovery if the new ACK is only partial, but also performs congestion control and reduces the window size to half its original value upon the arrival of the first new ACK. The E2E-ELN and E2E-ELN-RXMT protocols use ELN 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 a factor of two better than E2E. This is because the sender's explicit awareness of the wireless link reduces the number of coarse timeouts (Figure 5.6) and maintains a larger average window (Figure 5.7).

The E2E-ELN-RXMT protocol performs only slightly better than E2E-ELN when the socket buffer size is 32 KB. 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 results for the 8 KB socket buffer size indicate (Table 5.3). This is because E2E-ELN-RXMT does not wait for three duplicate ACKs before retransmitting a packet, if it has ELN information for it. The maximum socket buffer size of 8 KB limits the number of unacknowledged segments (i.e., the window) to a small number at any point in time, which reduces the probability of three duplicate ACKs arriving after a loss and triggering a fast retransmission, and increases the likelihood of a timeout.

Despite explicit awareness of wireless losses, timeouts sometimes occur in the ELN-based protocols. This is a result of our implementation of the ELN protocol, which does not convey information about *multiple* wireless-related losses in the same window to the sender. Since it is coupled with only cumulative ACKs, the sender is unaware of the occurrence of multiple wireless-related losses in a window.

Selective ACKs: We experimented with two different SACK schemes. In the LAN case, we used a simple SACK scheme based on SMART (Section 5.2.2). 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 link-layer scheme, LL-SMART-TCP-AWARE, obtained a throughput of 1.39 Mbps).

In the WAN case, our SACK implementation [11] was based on RFC 2018. For the exponentially-distributed loss pattern we used, the throughput was about 0.8 Mbps, significantly higher than the 0.31 Mbps throughput of TCP Reno. However, this is still about 35% worse than LL-OPT. Even

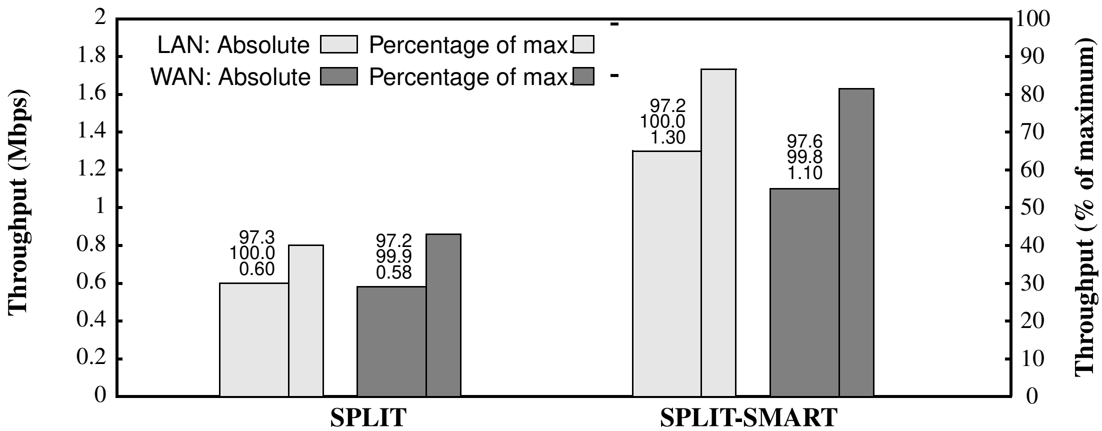

Figure 5.8 Performance of split-connection protocols: bit error rate = 1.9x10^{-6} (1 error/ 64 KB).

though SACKs allow the sender to often recover from multiple losses without timing out, the sender's congestion window decreases every time there is a packet dropped on the wireless link, causing it to remain small. The benefits of SACK are significant when windows are large and much smaller when they are small. SMART performs better because it assumes that no packet reordering occurs and retransmits segments early. We investigate in detail the problems to efficient loss recovery caused by small windows in Chapter 7.

In summary, E2E-NEWRENO is better than E2E, especially for large socket buffer sizes. Adding ELN to TCP significantly improves throughput 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 link-layer schemes in the LAN experiments, and over 35% worse in the WAN experiments for the particular error rate we used (1/64 KB).

5.4.3 Split-Connection Protocols

Split-connection protocols attempt to shield the sender from the vagaries of the wireless link by splitting the end-to-end connection in two at the base station. The hope is that the TCP sender (the wired connection) will be shielded by the wireless connection emanating from the base station.

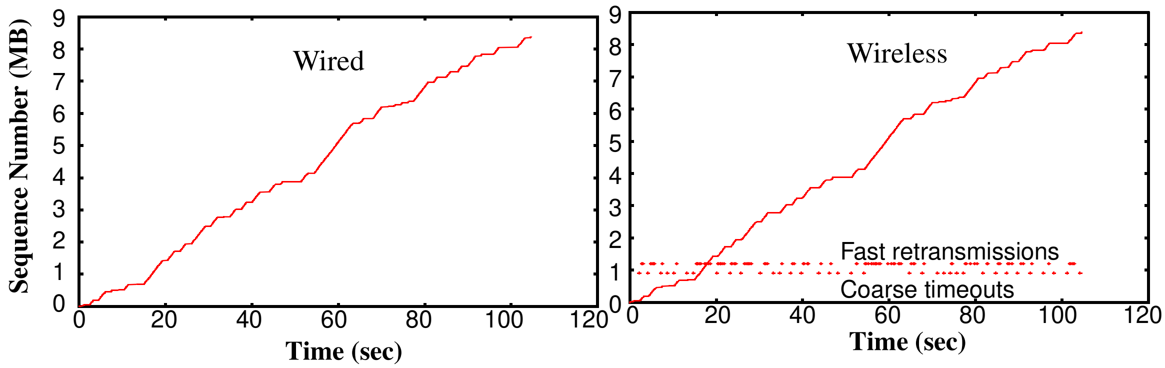

Figure 5.9 Packet sequence trace for the wired and wireless parts of the SPLIT protocol. The wireless part has two rows of horizontal dots: the top one shows the times of fast retransmissions and the bottom one the times of the timeout-based ones. The connection proceeds in fits and starts because the receiver-advertised window size from the base station to the TCP sender is often zero.

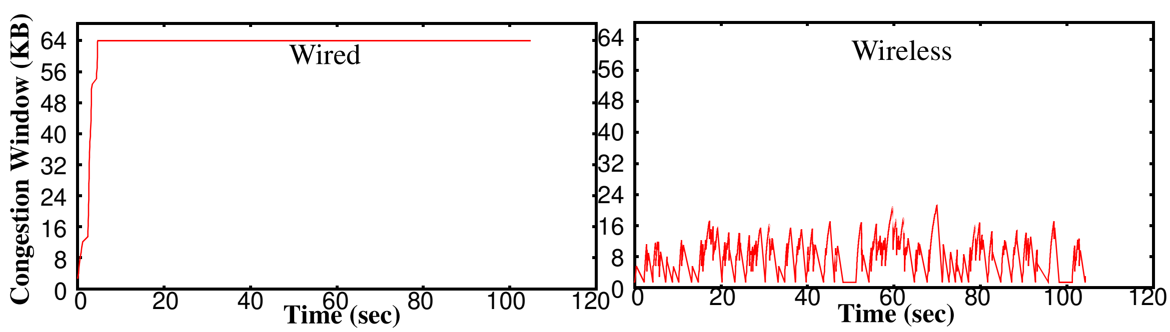

Figure 5.10 Congestion window sizes as a function of time for the wired and wireless parts of the split TCP connection. The wired sender sees no losses and maintains a 64 KB congestion window. However, the wireless TCP connection's congestion window fluctuates rapidly.

The sender of the wireless connection performs all the retransmissions in response to wireless losses.

Topology	SPLIT	SPLIT-SMART
LAN (8 KB)	0.54 (97.4%,100%)	1.30 (97.6%,100%)
LAN (32 KB)	0.60 (97.3%,100%)	1.30 (97.2%,100%)
WAN (32 KB)	0.58 (97.2%,100%)	1.10 (97.6%,100%)

Table 5.4 Summary of results for the split-connection schemes at an exponentially distributed error rate of 1 every 64 KB, on average.

Figure 5.8 and Table 5.4 show the throughput and goodput for the split-connection protocols in the LAN and WAN environments. We report the results for two cases: when the wireless connection uses TCP Reno (labeled SPLIT) and when it uses the SMART-based SACK scheme (labeled

SPLIT-SMART). In the LAN case, we see that the throughput achieved by SPLIT is quite low (0.6 Mbps), about the same as that for end-to-end TCP Reno (labeled E2E in Figure 5.5). The reason for this is apparent from Figures 5.9 and 5.10, which show the progress of the data transfer and the size of the congestion window for the wired and wireless connections. We see that the wired connection neither has any retransmissions nor any timeouts, resulting in a wired goodput of 100%. However, it periodically 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. Furthermore, a larger buffer at the base station will not improve steady-state performance for two reasons: (1) we measure performance in terms of receiver throughput, which is limited by the small congestion window size of the wireless connection, and (2) a long enough transfer will fill the buffer anyway. Thus, the performance of SPLIT is indistinguishable from TCP Reno, demonstrating that the split architecture per se does not yield any performance benefits.

In the WAN case the throughput of the SPLIT approach is about 0.58 Mbps, which is better than the 0.31 Mbps that the E2E approach achieves (Figure 5.5), but far inferior to the 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, the throughput for the SPLIT-SMART 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 ACK 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 a coarse timeout. This explains the 10-15% difference in throughput between the SPLIT-SMART scheme and the LL-SMART-TCP-AWARE scheme (Figure 5.2).

In summary, while the split-connection approach results in good throughput if the wireless connection uses special mechanisms, the performance is worse than that of a TCP-aware reliable link-layer protocol (LL-TCP-AWARE or LL-SMART-TCP-AWARE). The performance of the base SPLIT protocol is not significantly different compared to TCP Reno (E2E). Moreover, the link-layer protocols preserve the end-to-end semantics of TCP ACKs and do not incur the architectural

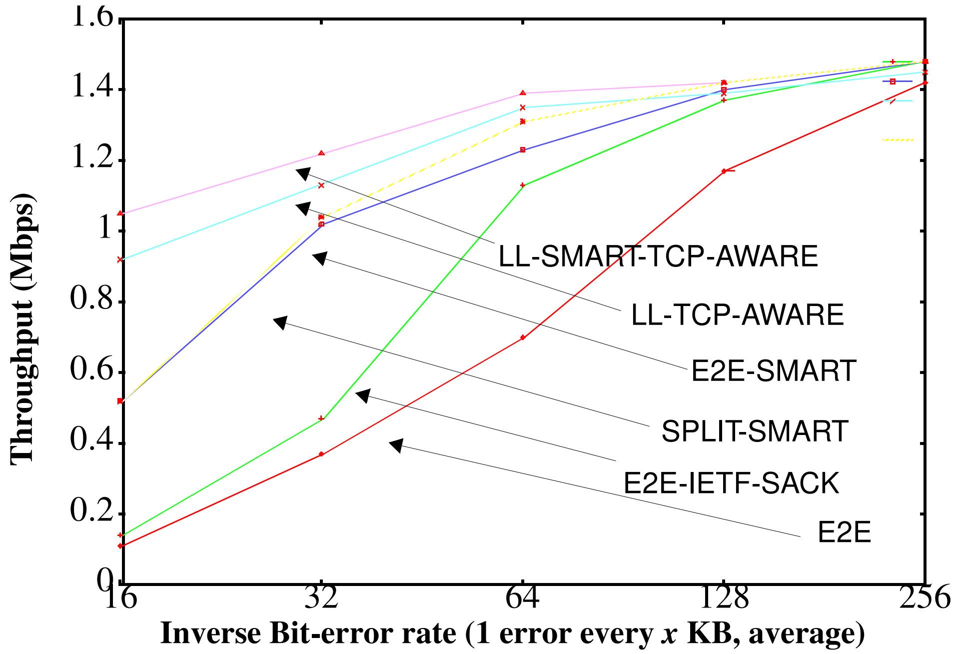

Figure 5.11 Performance of six protocols (LAN case) across a range of bit-error rates, ranging from 1 error every 16 KB to 1 every 256 KB shown on a log-scale.

drawbacks of the split-connection approaches. This demonstrates that the end-to-end connection need not be split at the base station to achieve good performance.

5.4.4 Performance at Different Error Rates

In this section, we present the results of several experiments performed across a range of bit-error rates for some of the protocols described earlier: E2E (the baseline case), LL-TCP-AWARE, LL-SMART-TCP-AWARE, E2E-SMART, E2E-IETF-SACK, and SPLIT-SMART. We choose the best performing protocols from each category, as well as some other protocols (e.g., E2E-IETF-SACK) to illustrate some interesting effects.

Figure 5.11 shows the performance of these protocols for an 8 MByte end-to-end transfer in a LAN environment for exponentially distributed error rates ranging from one error every 16 KB to one error every 256 KB, in increasing powers of two. We note that the overall qualitative results and conclusions are similar to those presented earlier for the 64 KB error rate. At low error rates (128 KB and 256 KB points in the graph), all the protocols shown perform almost equally well in improving TCP performance. At higher error rates, the performance of the TCP-aware link-layer schemes is about 1.75-2 times better than E2E-SMART and about 9 times better than TCP Reno. These graphs clearly show the benefits of TCP-aware link protocols like the Snoop protocol and the Snoop protocol enhanced with SACKs.

Another interesting point to note is the relative performance of E2E-IETF-SACK and E2E-SMART, especially at the high error rates. The congestion window does not grow larger than a few packets in the steady state at these error rates where there are multiple losses in many windows. E2E-IETF-SACK does not retransmit any packet using SACK information unless it receives three duplicate ACKs, to guard against potential packet reordering of packets on the path. This implies that no fast retransmissions are triggered if the number of segments in the window is less than four. The sender's congestion window is often smaller than this, leading to timeouts and degraded performance. In contrast, our implementation of E2E-SMART assumes no reordering of packets (which is justified on a LAN) and retransmits the lost packet when the first duplicate ACK arrives. This reduces the number of timeouts and leads to better end-to-end performance. We note that this approach only points out the benefits of avoiding timeouts and is not the final solution to this problem, because it unrealistically assumes that no packet reordering can occur in the Internet. In Chapter 7 we present a robust solution to this problem caused by small transmission windows.

5.5 Enhancements to the Snoop Protocol

In this section, we discuss two enhancements to the Snoop protocol. The first improves performance when there are cellular wireless transit links on the path that bridge wired clouds. The second exploits selective ACK information to perform better loss recovery when wireless losses occur in bursts.

5.5.1 Wireless Transit Links

The ELN scheme presented in Section 4.4.2 for improving end-to-end performance from a mobile TCP sender also generalizes to cellular wireless transit links where the TCP ACKs traverse the same base stations as the data packets on the forward path [12]. Consider a connection over one cellular wireless transit link and other wired links, as shown in Figure 5.12. Suppose a loss happened due to corruption over the wireless link. In this case, the agent at base station A would have seen the packet, while B would not, so the packet gets added to B's list of holes using the same

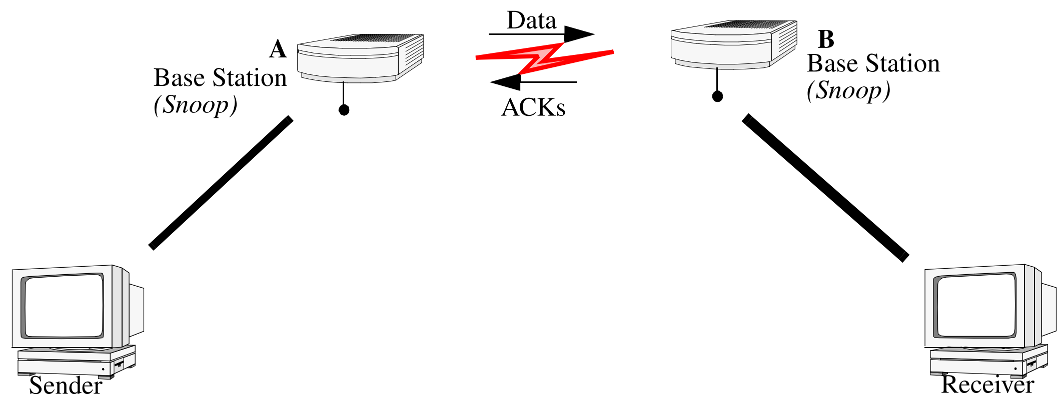

Figure 5.12 Topology for data transfer over a cellular wireless transit link. Both congestion and corruption could occur, necessitating active involvement by the base stations in demarcating and distinguishing the two types of losses to the TCP sender.

algorithm as in Section 4.4.2. When duplicate ACKs arrive for the missing packet at B, its agent sets the ELN information bit and propagates it towards A. Since A originally saw the packet (it is not in its list of holes), it infers that the packet was lost over the wireless link and lets the ACK go through to the data sender with the ELN information set on it, for the sender to perform a retransmission without invoking congestion control. Of course, it is possible to deploy a Snoop agent that caches and locally retransmits packets at A in this topology. In this case, the agent also suppresses duplicate ACKs for corrupted packets, as described earlier. It is important to note that local retransmissions from the Snoop agent at A now happen only upon duplicate ACKs with ELN set.

Now suppose that the packet was lost due to *congestion* elsewhere on the path. If it was lost before A, then the agent at A, upon seeing an ELN ACK via B, *resets* this bit because this packet did not reach A to start with. The sender reacts to this loss in the conventional way and performs congestion control. Similarly, if the packet was lost due to congestion after B, no agent in the network sets the ELN information on any duplicate ACK (because both A and B would have seen the packet), leading to correct congestion control. It is easy to see how this approach generalizes to the case when there is more than one single-hop wireless link on the path. Thus, we use ELN to differentiate between congestion and corruption in these topologies.

Our performance experiments show that in the first case (TCP sender retransmissions), observed performance is identical to the ELN case shown in Figure 4.9. In the second case (Snoop agent at A), performance closely tracks that of the last-hop Snoop agent, presented in Section 4.7.

5.5.2 Handling Burst Losses

In this section, we report the results of experiments that illustrate the benefit of selective ACKs in handling burst losses. We consider two of the best performing local protocols: LL-TCP-AWARE (Snoop) and LL-SMART-TCP-AWARE (Snoop with SMART-based selective ACKs). LL-TCP-AWARE recovers from a single loss by retransmitting the lost packet when a duplicate ACK arrives for it. It also keeps track of the number of expected duplicate ACKs and the next expected new ACK after this local retransmission. If this loss is part of a burst, the first new ACK to arrive after the duplicates will be less than the next expected new one, which causes an immediate retransmission of the lost segment. Thus, LL-TCP-AWARE attempts to recover from burst losses in a prompt manner, but resorts to a timeout if it is unsuccessful. LL-SMART-TCP-AWARE uses the additional useful information provided by the SMART-based selective ACKs—the sequence number of the segment that caused the duplicate ACK—to accurately determine losses and recover from them.

Burst Length	LL-TCP-AWARE (Mbps)	LL-SMART-TCP-AWARE (Mbps)
2	1.25	1.28
4	1.02	1.20
6	0.84	1.10

Table 5.5 Throughputs of Snoop and LL-SMART-TCP-AWARE at different burst lengths. This illustrates the benefits of SACKs, even for a high-performance, TCP-aware link protocol.

Table 5.5 shows the performance of the two protocols for bursts of lengths 2, 4, and 6 packets. These errors are generated at an average rate of one every 64 KB of data, and 2, 4, or 6 packets are destroyed in each case. Selective ACKs improve the performance of LL-SMART-TCP-AWARE over LL-TCP-AWARE by up to 30% in the presence of burst errors. While this is a simplistic burst-error model, it does illustrate the effectiveness of the TCP-aware schemes and selective ACKs in recovering from the loss of multiple packets in close succession.

5.6 Discussion

This section reflects on some key lessons we have learned from our experience with reliable transport over error-prone wireless networks.

- **Cross-layer protocol optimizations:** For data transfer to a mobile host, the Snoop protocol uses a *transport-aware* link protocol between the base station and mobile host to achieve good end-to-end performance for bulk and Web-like transfers. For data transfer from a mobile host, it uses ELN to make the transport protocol *link-aware*. This approach violates the conventional layered model used in network protocol design, where there is no explicit control information transferred between higher and lower layers of the protocol stack. The benefits of our solutions in both simulation and real deployments show that better protocol adaptation is enabled by making different layers of the protocol stack more aware of what the other layers are doing and also of what the underlying channel is doing. This approach achieves superior end-to-end performance because it *explicitly* prevents adverse interactions that could occur between independent protocols at different layers of the stack.

- **The end-to-end argument:** The design of the Snoop agent and protocol has been strongly influenced and guided by the end-to-end argument and design principle propounded by Saltzer, Reed and Clark [127]. The end-to-end argument is a design principle that guides the placement of services in a system. It states that a service should be implemented in a subsystem only if the service can be completely implemented in that subsystem, or if by partially implementing the service in the subsystem, it substantially improves end-to-end performance. This is precisely the philosophy that guided the design of the Snoop protocol and its approach toward local recovery. In contrast to split-connection approaches, the Snoop protocol does not violate the end-to-end semantics of TCP and is not vulnerable to base station failures. The good performance of some of the split connection approaches derives not from their spoofing ACKs and breaking the connection, but from their running a wireless-optimized protocol over the noisy link [14]. And, it is possible to obtain the same or higher benefit using the Snoop approach, without any change in end-to-end semantics.

- **Soft-state agents in the network infrastructure:** The performance benefits of deploying a soft-state Snoop agent at the base station are significant. In general, we believe that this is an efficacious way in which network heterogeneity and associated performance problems can be solved, using agents in the network infrastructure. The Snoop agent state is soft, because it is not crucial to the correct end-to-end functioning of the system and because it is periodically and automatically refreshed upon the arrival of data and ACKs. Any errors in the state do not compromise the correctness of the connection; incorrect state is automatically rectified when new messages arrive. This soft-state approach also enables low handoff latencies with little connection disruption, since the contents of the Snoop cache can be mirrored using multicast from the home agent (Section 4.5.1).

- **Data-driven retransmissions:** Reliable transport protocols can recover from losses in two ways: using data-driven retransmissions, which infer that an earlier packet is missing from the receipt of later data, and timeouts, which are triggered when the sender does not receive an ACK within a certain period of time. Data-driven retransmissions recover from losses with little disruption in the stream, while timeouts necessarily have to be conservative to ensure that data that is delayed but still in transit does not get prematurely retransmitted [150]. In the Snoop protocol, we reduce the dependence on local timers by performing data-driven retransmissions as often as possible.

- **Higher priority local retransmissions:** Retransmitting packets at a higher priority improves performance at all error rates. This enables retransmitted packets to reach the mobile host sooner, reducing the number of duplicate ACKs and leading to improved throughput. In the common case when the wireless link is the connection's bandwidth bottleneck, the first new ACK generated after a higher-priority retransmission is usually less than what it would be without such prioritization. This implies that the number of segments transmitted by the sender in a burst is smaller in the former case, because the sliding window shifts by a smaller amount. This in turn reduces the degree of congestion in the network by reducing burst transmissions, leading to improved overall performance.

  Higher priority local retransmissions also allow us to avoid unnecessary local timeouts in the following way. When a retransmission is scheduled, the Snoop agent precisely knows what the last in-sequence transmission was. Hence, it knows what the next expected new ACK should be, assuming no further losses happen. Now, if the next new ACK is smaller than the next expected one, a retransmission occurs without a timeout. This helps in recovering from multiple losses in a single window without a timeout.

- **Ineffectiveness of end-to-end TCP SACK:** An interesting observation concerns the lack of effectiveness of purely end-to-end TCP SACK in the wireless environment, despite the sender's ability to recover quickly from multiple losses in a window when SACK information is available. Initially, this was surprising to us because of SACK's benefits in burst loss situations.

However, upon closer analysis [14], we found that even with SACK, the sender incurred numerous coarse timeouts. With a large enough window, burst losses over the wireless link are indeed recovered quickly, but the sender still performs congestion control and reduces its window. The end result is a small average window size, with only a small number of outstanding segments. As a result, there is not enough information for the sender to distinguish between reordered and lost packets, necessitating coarse timeouts. We present a robust solution to this problem based on enhancing TCP's loss recovery algorithm in Chapter 7.

- **Sharing loss recovery information:** When there are multiple connections outstanding to the same mobile host, the Snoop agent can perform loss recovery in an integrated way by effectively sharing ACK information between them. We do this by treating the different connections as one integrated stream and by keeping track of the relative transmission order between packets belonging to different connections. Then, when an ACK arrives on a connection that acknowledges a data item that was transmitted *after* a currently unacknowledged segment belonging to another connection, we treat this as a *later* ACK indication for the latter connection. When the number of *later* ACKs reaches a threshold, we retransmit the earliest missing segment, independent of the connection(s) that generated this later ACK information. Results of an end-to-end integrated loss recovery scheme in the context of TCP are presented in [15, 107].

- **End-to-end security:** An important issue concerning the viability of approaches like the Snoop protocol and split connection approaches is their interaction with end-to-end IP Security. If a connection has end-to-end security enabled, its entire IP payload, including the transport header, would be encrypted and therefore unreadable by network agents like the Snoop agent. One possible solution is for the source to not encrypt part of the TCP header—this would not compromise the security model provided by IP Security and would overcome the problem. Another (less attractive) possibility is to include the base station as part of the mobile host's trust infrastructure and allow it to decrypt packets destined for the mobile host. Developing a standardized way to solve this problem without compromising end-to-end security is currently gaining momentum in the Internet Engineering Task Force (IETF).

5.7 Summary

In this chapter, we presented a comparative analysis of several techniques to improve the end-to-end performance of TCP over error-prone wireless links. We categorized these as end-to-end, link-layer or split-connection schemes. We used end-to-end throughput and wired and wireless goodputs as metrics for comparison.

Our results lead to the following conclusions:

1. A reliable link-layer protocol that uses knowledge of TCP (LL-TCP-AWARE) to shield the sender from duplicate ACKs arising from wireless losses gives a 10-30% higher throughput than one (LL) that operates independently of TCP and does not attempt in-order delivery of packets. Also, the former avoids redundant retransmissions by both the sender and the base station, resulting in a higher goodput. Of the schemes we investigated, the TCP-aware link-layer protocol with SACKs performs the best. Thus, our conclusion is that such cross-layer optimizations as in the Snoop protocol are beneficial in obtaining good performance in the face of wireless bit-errors.

2. End-to-end schemes, while not as effective as local techniques in handling wireless losses, are promising since significant performance gains can be achieved without extensive support from intermediate nodes in the network. The ELN scheme resulted in a throughput improvement of more than a factor of two over TCP-Reno with comparable goodput. This is the link-aware transport algorithm used by the Snoop protocol for transfers from the mobile host.

3. The split-connection approach with TCP Reno used over the wireless link does not perform well. The sender periodically stalls due to timeouts on the wireless connection, leading to poor end-to-end throughput. A wireless-optimized SMART-based SACK mechanism for the wireless hop yields better throughput. However, it is still less than that of a transport-aware link-layer protocol that does not split the connection. This demonstrates that splitting the end-to-end connection is not essential for good performance in wireless systems and that the good performance of SPLIT-SACK is not a consequence of the split-connection architecture.

4. The SMART-based SACK scheme is quite effective at handling the high packet loss rate when deployed over the wireless hop or by the sender in a LAN environment. In the WAN experiments, the SACK scheme based on RFC 2018 improves end-to-end performance, although its

performance is not as good as the link-layer schemes. From our results we conclude that SACKs are very useful in the presence of error-prone links, especially when losses occur in bursts. However, the performance benefits of *end-to-end* SACK protocols are not impressive because average windows remain small, and the sender does not have sufficient information to distinguish genuine losses from potentially reordered packets.

5. Using ELN and soft-state network agents, we developed an enhancement to the Snoop protocol that performs well when error-prone wireless transit links are part of the connection.

In summary, our results show that cross-layer protocol optimizations significantly improve transport performance over error-prone wireless links in a variety of topologies. We demonstrated this using a transport-aware link protocol and a link-aware transport protocol that uses the ELN mechanism to distinguish congestion from corruption.

The next chapter discusses the problems caused by asymmetric effects on TCP performance and presents our solutions to them.