In Proc. IEEE Globecom 1998, Internet Mini-Conference, Sydney, Australia.

Explicit Loss Notification and Wireless Web Performance

Hari Balakrishnan and Randy H. Katz
{hari,randy}@cs.berkeley.edu
Computer Science Division, Department of EECS,
University of California at Berkeley

Abstract

This paper describes our experiences with improving TCP and Web performance over a WaveLAN-based wireless network. In previous work, we analyzed the problems to TCP performance in error-prone wireless networks and designed the Berkeley Snoop protocol to significantly improve performance over error-prone wireless links [7, 5].

Most efforts to date have focused on improving performance for transfers *to* a mobile host. We present a novel protocol based on Explicit Loss Notification (ELN) to improve performance when the mobile host is the TCP sender, a situation that is becoming increasingly common. Then, we use experimental packet traces of wireless errors from a production wireless network to derive an empirical model of channel errors. We use this to evaluate the performance of TCP Reno, TCP Selective Acknowledgments [17] and the Snoop protocol for Web workloads to mobile hosts. We also discuss the scaling behavior of the Snoop protocol and reflect on some general lessons we have learned about efficient protocol design for reliable wireless transport.

1. Introduction

Wireless technologies are playing an increasingly prominent role in the global Internet infrastructure. They are ideal as Internet access technologies, providing a convenient and cheap solution to the "last mile" problem. However, reliable transport protocols such as the Transmission Control Protocol (TCP) [22, 25], used by popular applications like the World Wide Web, file transfer, electronic mail, and interactive remote terminal applications, show greatly degraded performance over wireless networks. Over the past many years, TCP has been tuned for traditional networks comprising wired links and stationary hosts. It assumes *congestion* in the network to be the primary cause for packet losses and unusual delays, and adapts to it. The TCP receiver sends cumulative acknowledgments (ACKs) for successfully received segments, which the sender uses to determine which segments have been successfully received. The sender identifies the loss of a packet either by the arrival of several duplicate cumulative ACKs, triggering a fast retransmission, or by the absence of an ACK for a timeout interval equal to the sum of the smoothed round-trip delay and four times its mean deviation. TCP reacts to packet losses by retransmitting missing data, and simultaneously invoking congestion control by reducing its transmission (congestion) window size and backing off its retransmission timer. These measures reduce the level of congestion on the intermediate links.

Unfortunately, when packets are lost for reasons other than congestion, these measures result in an unnecessary reduction in end-to-end throughput and hence, in sub-optimal performance. Communication over wireless links is often characterized by high bit-error rates due to channel fading, noise or interference, and intermittent connectivity due to handoffs. TCP performance in such networks suffers from significant throughput degradation and very high interactive delays because the sender misinterprets corruption for congestion [see e.g., 8].

Recently, several schemes have been proposed to alleviate the effects of non-congestion-related losses on TCP performance over error-prone networks [3, 7, 26]. In previous work [5], we compared many existing schemes and argued for *transport-aware* link protocols to improve performance. The end result of this work was a more adaptive TCP/IP protocol architecture that adapted not only to network congestion, but also to error-prone wireless links.

However, most design and measurement efforts to date have focused on the (common) case of data transfers *to* a mobile host. The case of a mobile host transmitting data over a first-hop wireless link to hosts on the wired Internet has largely been ignored. Such access scenarios are becoming increasingly common in the Internet; for example, researchers at Daimler Benz are working on a prototype of a Mercedes car that runs a Web server, which disseminates information over wireless and wired links about the status of the vehicle to users on the Internet [13].

In this paper, we present a novel protocol based on Explicit Loss Notification (ELN) to improve transport performance when the mobile host is the TCP sender, a situation that is becoming increasingly common in many wireless access networks (Section 3). Then, we obtain experimental packet traces of wireless errors from a production outdoor wireless network deployed as part of the Reinas environmental monitoring project at UC Santa Cruz [9] and derive an empirical model of channel errors based on this data (Section 4). We use this to evaluate the performance of TCP Reno, TCP SACK [17] and the Snoop protocol for an empirically derived Web workload to mobile hosts (Section 5). Finally,

we reflect on some lessons learned about improving wireless transport performance in the face of wireless bit-errors (Section 6) and conclude with a summary and directions for future work (Section 7).

2. Background

In this section, we summarize some protocols and transport enhancements that have been proposed to improve the performance of TCP over wireless links.

- **Link-layer protocols**: There have been several proposals for reliable link-layer protocols to improve wireless performance [1, 14, 19]. These typically use forward error correction (FEC) or retransmissions (ARQ) to improve performance.

  The main advantage of employing a link-layer protocol for loss recovery is that it fits naturally into the layered structure of network protocols and achieve local reliability. However, there are several occasions when transport performance does not improve when such protocols are used. There are three chief modes of adverse interaction that could arise between independent reliable transport and link layers:

  1. *Timer interactions:* Independently set timers at both layers could trigger at the same time, leading to redundant retransmissions at both layers and degraded performance [10]. While this is a concern in general, TCP does not suffer from this problem because of the coarse and conservative timeout intervals in practice.

  2. *Fast retransmission interactions:* This arises when a link layer protocol achieves reliability by local retransmissions, but does not preserve the in-order sequential delivery of TCP segments to the receiver. Then, although local recovery occurs, the receipt of later segments causes duplicate ACKs from the receiver as well, leading to redundant sender fast retransmissions, sender window reduction, and reduced throughput [5].

  3. *Large round-trip variations:* If a link layer protocol is designed to be overly robust and attempt, for example, to provide a TCP-like service, it often result in long latencies and large round-trip time deviations at the TCP sender. This leads to long and conservative timeouts when congestion-related losses occur on the path. Other problems that arise include a large number of redundant retransmissions if a sender timeout occurs for a wireless loss [15], a problem observed in GSM networks running the RLP protocol [18].

- **Split connection protocols [3, 26]:** Split connection protocols split each TCP connection between a sender and receiver into two separate connections at the base station — one TCP connection between the sender and the base station, and the other between the base station and the receiver. Over the wireless hop, a specialized protocol tuned to the wireless environment may be used. In [26], the authors propose two protocols — one in which the wireless hop uses TCP, and another in which the wireless hop uses a selective repeat protocol (SRP) on top of UDP. They study the impact of handoffs on performance and conclude that they obtain no significant advantage by using SRP instead of TCP over the wireless connection in their experiments. However, our experiments demonstrate benefits in using a simple SACK scheme with TCP over the wireless connection.

  Indirect-TCP [3] is a split-connection solution that uses standard TCP for its connection over the wireless link. Like other split-connection proposals, it attempts to separate loss recovery over the wireless link from that across the wireline network, thereby shielding the original TCP sender from the wireless link. However, as our experiments indicate, the choice of TCP over the wireless link results in several performance problems. Since TCP is not well-tuned for the lossy link, the TCP sender of the wireless connection often times out, causing the original sender to stall. In addition, every packet incurs the overhead of going through TCP protocol processing twice at the base station (as compared to zero times for a non-split-connection approach), although extra copies are avoided by an efficient kernel implementation. Another disadvantage of split connections is that the end-to-end semantics of TCP ACKs is violated, since ACKs to packets can now reach the source even before the packets actually reach the mobile host. This makes the system vulnerable to base station crashes. Also, since split-connection protocols maintain a significant amount of state at the base station per TCP connection, handoff procedures tend to be complicated and slow [2].

- **Snoop Protocol [7]:** The snoop protocol introduces a module, called the *snoop agent*, at the base station. The agent monitors every packet that passes through the TCP connection in both directions and maintains a cache of TCP segments sent across the link that have not yet been acknowledged by the receiver. A packet loss is detected by the arrival of a small number of duplicate ACKs from the receiver or by a local timeout. The snoop agent retransmits the lost packet if it has it cached and suppresses the duplicate ACKs. Thus, the snoop protocol is a transport-aware link protocol.

  The main advantage of this approach is that it suppresses duplicate ACKs for TCP segments lost and retransmitted locally, thereby avoiding unnecessary fast retransmissions and congestion control invocations by the sender. The per-connection state maintained by the snoop agent at the base station is *soft*, and is not essential for correctness. A detailed description of the Snoop protocol appears in [7].

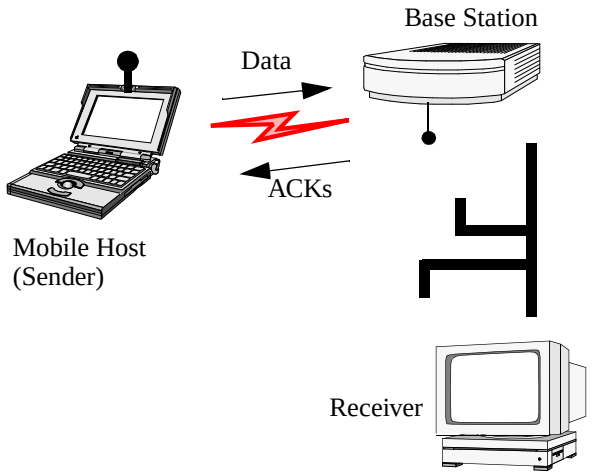

Figure 1. Topology for data transfer from a mobile host.

- **Selective Acknowledgments**: Since TCP Reno uses a cumulative ACK scheme, it often does not provide the sender with sufficient information to recover quickly from multiple packet losses within a single transmission window. The SACK RFC [17] proposes that each ACK contain information about up to three non-contiguous blocks of data that have been received successfully by the receiver. Each block of data is described by its starting and ending sequence number. Due to the limited number of blocks, it is best to inform the sender about the most recent blocks received. The RFC does not specify the sender behavior, except to require that standard TCP congestion control actions be performed when losses occur. SACKs help greatly when window sizes are large and multiple losses occur in a single transmission window [11].

3. Transfers from a Mobile Host

We now proceed to discuss the case of data transfer from a mobile host. While this has conventionally been a less common scenario for wireless information access, it is rapidly gaining in importance. There is an increasing number of Web and other information servers that serve data across first-hop wireless links [13, 9]. A representative topology of this scenario is shown in Figure 1.

It might be tempting to come to the conclusion that a protocol very similar to the snoop protocol described in Section 2 and [7] for data transfer to the mobile host can be used in this case as well. In particular, one of the important features of that case was that it required changes only to the base station and not to any other entities in the network. However, it is unlikely, if not impossible, that a protocol with modifications made *only* at the base station can substantially improve the end-to-end performance of reliable bulk data transfers from the mobile host to other hosts on the network, while preserving the precise semantics of TCP ACKs. For example, caching packets at the base station and retransmitting them as necessary is not useful, since most of the packet losses will be from the mobile host to the base station. Running a Snoop agent at the mobile host will be inappropriate because the same duplicate ACKs signify both corruption and congestion losses. There is no way for the mobile sender to know if the loss of a packet happened on the wireless link or elsewhere in the network due to congestion.

There are at least two ways in which the basic Snoop scheme can be enhanced to achieve good performance for this case — the first involves the use of negative ACKs, and the second the use of Explicit Loss Notifications (ELN). We first describe the first approach, and then the second, which we believe to be an elegant and simple solution to the problem. We also show how the algorithm naturally generalizes to the case when a cellular wireless link is the transit link bridging two or more wired networks.

3.1 Using Negative ACKs

An agent at the base station keeps track of the packets that were lost in any transmitted window and generates negative acknowledgments (NAKs) for those packets back to the mobile sender. This is especially useful if several packets are lost in a single transmission window, a situation that happens often under high interference or in fades where the strength and quality of the signal are low. These NAKs are sent when either a threshold number of packets (from a single window) have reached the base station or when a certain amount of time has expired without any new packets from the mobile. Encoding these NAKs as a bit vector can ensure that the fraction of the sparse wireless bandwidth consumed by NAKs is relatively low. The mobile host used these NAKs to selectively retransmit lost packets.

NAKs can be implemented using the TCP SACK option. The agent could use SACKs to enable the mobile host to quickly (relative to the round-trip time of the connection) retransmit missing packets. The only change required at the mobile host is to enable SACK processing. No changes of any sort are required in any of the fixed hosts. Also, SACKs are generated by the snoop agent when it detects a gap corresponding to missing packets; we emphasize again that no transport-layer code runs at the base station to do this.

There is a danger of a possible violation of end-to-end semantics in this scheme, because the snoop agent would send SACK information about segments it receives, but they may not yet have been received by the intended receiver. This will lead to problems if the corresponding segments get lost later in the path. However, this is not strictly true because of the semantics of TCP SACKs: they are *advisory* in nature, and only cumulative ACKs are binding. What this means is that even if a TCP sender receives SACKs for certain data blocks, it must not clean those segments from it's transmission buffer or assume successful reception, until cumulative ACKs arrive for them.

However, this approach based on SACKs is inelegant because it relies on a relatively obscure feature of the SACK specification. Of course, we could implement an explicit NAK scheme, but that would require complex modifications at the sender, which would anyway implement SACKs in the future. In addition, we would like to avoid explicit protocol messaging as far as possible, and both SACKs and NAKs generated from the base station do not provide this. All these issues led us to investigate alternate protocols; what follows is one such scheme.

3.2 Using Explicit Loss Notifications

Explicit Loss Notification (ELN) is a mechanism by which the reason for the loss of a packet can be communicated to the TCP sender. In particular, it provides a way by which senders can be informed that a loss happened because of reasons unrelated to network congestion (e.g., due to wireless bit errors), so that sender retransmissions can be decoupled from congestion control. If the receiver or a base station knows for sure that the loss of a segment was not due to congestion, it sets the ELN bit in the TCP header and propagate it to the source. In the situation at hand, this ELN message is sent as part of the same connection (and not in a separate way, using ICMP for instance). This simplifies the sender implementation as it receives messages in-band. ELN is a general concept that has applications in a wide variety of wireless topologies. In this section, we describe it in the context of data transfer from a mobile host connected to the rest of the Internet via a cellular wireless link.

The snoop agent running at the base station monitors all TCP segments that arrive over the wireless link. However, it does not cache any TCP segments since it does not perform any retransmissions. Rather, it keeps track of *holes* in the sequence space as it receives data segments, where a hole is a missing interval in the sequence space. These holes correspond to segments that have been lost over the wireless link. However, it could also be the case that the packet was lost due to congestion at the base station. To avoid against wrongly marking a congestion hole as having been due to a wireless loss, it only adds a hole to the list of holes when the number of packets queued on the base station's input interface is not close to the maximum queue length.

When ACKs, especially duplicate ACKs, arrive from the receiver, the agent at the base station consults its list of holes. It sets the ELN bit on the ACK if it corresponds to a segment in the list before forwarding it to the data sender. It also cleans up all holes with sequence numbers smaller than the current ACK, since they correspond to segments that have been successfully received by the receiver. When the sender receives an ACK with ELN information in it, it retransmits the next segment, but does not take any congestion control actions. The sender also makes sure that each segment is retransmitted at most once during the course of a single round-trip, as the snoop agent would flag an ELN for *each* duplicate ACK following a loss.

3.3 ELN Implementation

We need to make modifications to both the base station (BS) and the mobile host (MH), which is the TCP sender. Fixed hosts in the rest of the Internet are unchanged.

Data structures: At the BS, the snoop agent detects holes in the data transmission from the MH. The basic data structure used for this purpose is a list of blocks, or maximal contiguous segments of data specified by a starting sequence number and size. Gaps between blocks correspond to missing segments in the transmission sequence. At the MH, a new variable is added to the connection's TCP control block to keep track of the last ELN-triggered retransmission.

Data path: When new data arrives from the MH, the agent attempts to coalesce it with an existing block and checks if it bridges a previous hole. A new data segment that is out of the normal increasing sequence creates a hole, because segments in between have been lost. We ensure that every hole was the consequence of a corruption-induced loss, and not caused by congestion. Congestion-induced losses can occur in two ways — due to the base station's input queue overflowing, or due to the mobile host's output queue overflowing. Piggybacked on each packet from the MH is its instantaneous output queue length, which the base station can use to classify each loss it detects. When a new, out-of-sequence packet arrives at the BS, the snoop agent checks the size of its input queue and gets the size of the MH's output queue from the packet. If either of the instantaneous queue sizes is above a certain threshold of the maximum (set to 75% in our implementation), then this loss is not considered as one due to corruption. Because the agent should only flag ELN information for those losses that were unrelated to congestion, the implicitly maintained list of holes should only include packets lost due to corruption.

ACK processing: Whenever an ACK arrives at the BS, the agent updates the list of blocks by cleaning up all blocks (or parts of blocks) that have been acknowledged so far. It also checks if the ACK corresponds to a hole, i.e., if the next segment in sequence was lost due to corruption on the wireless link. It does so by comparing the ACK to the sequence number of the earliest block currently in its list for that connection. If the value of the ACK is smaller, it sets the ELN bit in the TCP header, modifies the TCP checksum, and forwards the ACK on to the MH. Because there is currently no specific bit in the TCP header for ELN, we use one of the unreserved bits for this purpose. Note that while ELN marking happens most often for duplicate ACKs, it can (and does) also happen for new ones. However, it does not happen when the list of blocks is empty, because there is no way the agent at the BS can know if any further data has been transmitted by the mobile host.

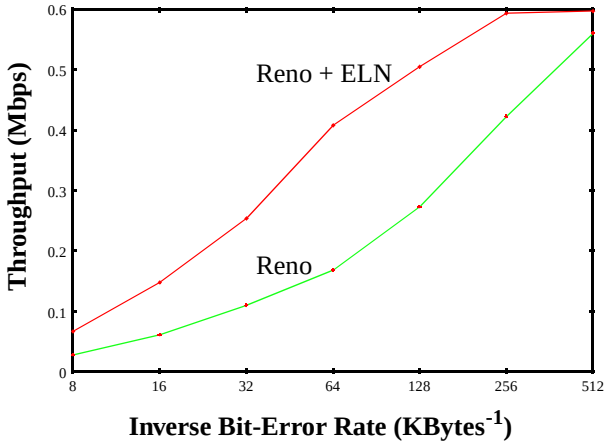

Figure 2. Throughput of TCP Reno and Reno enhanced with ELN across a range of exponentially distributed bit-error rates for transfers from a mobile host.

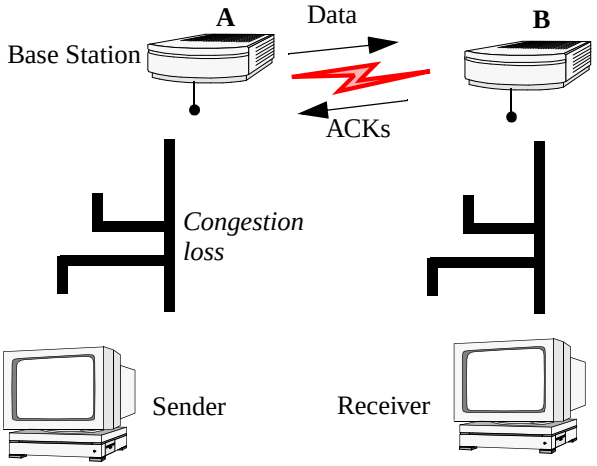

Figure 3. Topology for data transfer over a cellular wireless transit link.

When the MH receives an ACK with the ELN bit set, it retransmits the missing packet and updates a variable (eln_last_rxmit) that keeps track of the last ELN-induced retransmission. This ensures that the MH does not retransmit the same packet on every ACK with ELN that it receives via the base station; thus, the retransmission policy is left to the end-host, and the base station only conveys relevant information to it. The TCP stack at the MH uses a configurable variable, eln_rexmt_thresh, to determine when to retransmit a segment upon receiving an ACK with ELN set. Upon receiving eln_rexmt_thresh ACKs with ELN, it retransmits the missing segment. In our implementation and experiments, we set its value to 2.

When the MH retransmits a segment based on ELN information, it does not reduce its congestion window and perform congestion control. It also bypasses the fast recovery code that inflates the congestion window for every duplicate ACK. Finally, we note that these actions are taken only for duplicate ACKs with ELN, and not for other duplicate ACKs that signify network congestion.

3.4 Performance

We performed several experiments to measure the performance of data transfer from the MH to the FH. These results, measured across a range of exponentially-distributed bit-error rates, are shown in Figure 2. As before, there are significant performance benefits of using the snoop protocol coupled with the ELN mechanism in this situation. These measurements were made for wide-area transfers between UC Berkeley and IBM Watson, across one wireless WaveLAN hop and 16 Internet hops. At medium to high error rates, the performance improvement due to ELN is roughly a factor of 2. At lower error rates, TCP Reno performs quite well as expected, and the benefits of ELN are not as pronounced. The main advantage of ELN is that it helps maintain a large TCP congestion window even when wireless error rates are high, reacting only to congestion.

While the relative performance of ELN is about 100% better than Reno at high error rates, it is not as high as the relative improvement for the FH to MH case using the Snoop protocol. The reason for this is that the Snoop protocol performs quick local recovery in addition to shielding the sender from congestion. On the other hand, loss recovery with ELN is due to TCP retransmissions alone. When multiple losses occur in a window TCP with ELN incurs a coarse timeout, leading to “only” a factor of two improvement in throughput.

3.5 Cellular Wireless Transit Links

The ELN-based approach presented above for improving end-to-end performance also generalizes to cellular wireless transit links where the TCP ACKs traverse the same base stations as the data packets on the forward path. Consider a connection over one cellular wireless transit link and other wired links, as shown in Figure 3. 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 algorithms as in Section 3.2. 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 obviously lost over the wireless link and simply lets the ACK go through to the data sender with the ELN information set on it. 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.