In Proc. 1st ACM Int'l Conf. on Mobile Computing and Networking (Mobicom), November 95.
This paper won the best student paper award.

Improving TCP/IP Performance over Wireless Networks[1]

Hari Balakrishnan, Srinivasan Seshan, Elan Amir and Randy H. Katz
{hari,ss,elan,randy}@CS.Berkeley.EDU
Computer Science Division
University of California at Berkeley

Abstract

TCP is a reliable transport protocol tuned to perform well in traditional networks made up of links with low bit-error rates. Networks with higher bit-error rates, such as those with wireless links and mobile hosts, violate many of the assumptions made by TCP, causing degraded end-to-end performance. In this paper, we describe the design and implementation of a simple protocol, called the *snoop* protocol, that improves TCP performance in wireless networks. The protocol modifies network-layer software mainly at a base station and preserves end-to-end TCP semantics. The main idea of the protocol is to cache packets at the base station and perform local retransmissions across the wireless link. We have implemented the snoop protocol on a wireless testbed consisting of IBM ThinkPad laptops and *i*486 base stations communicating over an AT&T Wavelan. Our experiments show that it is significantly more robust at dealing with unreliable wireless links as compared to normal TCP; we have achieved throughput speedups of up to 20 times over regular TCP in our experiments with the protocol.

1. Introduction

Recent activity in mobile computing and wireless networks strongly indicates that mobile computers and their wireless communication links will be an integral part of future internetworks. Communication over wireless links is characterized by limited bandwidth, high latencies, high bit-error rates and temporary disconnections that must be dealt with by network protocols and applications. In addition, protocols and applications have to handle user mobility and the handoffs that occur as users move from cell to cell in cellular wireless networks. These handoffs involve transfer of communication state (typically network-level state) from one base station (a router between a wired and wireless network) to another, and typically last anywhere between a few tens to a few hundreds of milliseconds.

Reliable transport protocols such as TCP [Pos81, Ste94, Bra89] have been tuned for traditional networks made up of wired links and stationary hosts. TCP performs very well on such networks by adapting to end-to-end delays and packet losses caused by congestion. TCP provides reliability by maintaining a running average of estimated round-trip delay and mean deviation, and by retransmitting any packet whose acknowledgment is not received within four times the deviation from the average. Due to the relatively low bit-error rates over wired networks, all packet losses are correctly assumed to be because of congestion.

In the presence of the high error rates and 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 [Jac88]) and resets its retransmission timer (Karn's Algorithm [KP87]). These measures result in an 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 [CI94].

In this paper, we describe the design and implementation of a simple protocol to alleviate this degradation and present the results of several experiments using this protocol. Our aim is to improve the end-to-end performance on networks with wireless links without changing existing TCP implementations at hosts in the fixed network and without recompiling or relinking existing applications. We achieve this by a simple set of modifications to the network-layer (IP) software at the base station. These modifications consist mainly of caching packets and performing local retransmissions across the wireless link by monitoring the acknowledgments to TCP packets generated by the receiver. Our experiments show speedups of up to 20 times over regular TCP in the presence of bit errors on the wireless link. We have also found that our protocol is significantly more robust at dealing with multiple packet losses in a single window as compared to regular TCP.

The rest of this paper is organized as follows. In Section 2, we describe and evaluate some design alternatives and

1. This work was supported by ARPA Contract J-FBI-93-153.

related work that addresses this problem. In Section 3, we describe the details and dynamics of the protocol. We describe our implementation and the modifications to the router software at the base station in Section 4 and the results of several of our experiments in Section 5. Section 6 compares our protocol with some of the other alternatives published in the literature. We discuss our future plans in Section 7 and conclude with a summary in Section 8.

2. Design Alternatives and Related Work

Is TCP an appropriate protocol model for wireless networks? We believe it is. Since many network applications are built on top of TCP, and will continue to be in the foreseeable future, it is important to improve its performance in wireless networks *without any modifications to the fixed hosts*. This is the only way by which mobile devices communicating on wireless links can seamlessly integrate with the rest of the Internet.

Recently, several reliable transport-layer protocols for networks with wireless links have been proposed [BB94, BB95, CI94, YB94] to alleviate the poor end-to-end performance of unmodified TCP in the wireless medium. We summarize these protocols in this section and point out the advantages and disadvantages of each method. In Section 6. we present a more detailed comparison of these schemes with our protocol.

- **The Split Connection Approach:** The Indirect-TCP (I-TCP) protocol [BB94, BB95] was one of the first protocols to use this method. It involves splitting a TCP connection between a fixed and mobile host into two separate connections at the base station -- one TCP connection between the fixed host and the base station, and the other between the base station and the mobile host. Since the second connection is over a one-hop wireless link, there is no need to use TCP on this link. Rather, a more optimized wireless link-specific protocol tuned for better performance can be used [YB94]. The advantage of the split connection approach is that it achieves a separation of flow and congestion control of the wireless link from that of the fixed network and hence results in good bandwidth at the sender. However, there are some drawbacks of this approach, including:

  1. *Semantics:* I-TCP acknowledgments and semantics are not end-to-end. Since the TCP connection is explicitly split into two distinct ones, acknowledgments of TCP packets can arrive at the sender even before the packet actually reaches the intended recipient. I-TCP derives its good performance from this splitting of connections. However, as we shall show, there is no need to sacrifice the semantics of acknowledgments in order to achieve good performance.
  2. *Application relinking:* Applications running on the mobile host have to be relinked with the I-TCP library and need to use special I-TCP socket system calls in the current implementation.
  3. *Software overhead:* Every packet needs to go through the TCP protocol stack and incur the associated overhead *four* times -- once at the sender, twice at the base station, and once at the receiver. This also involves copying data at the base station to move the packet from the incoming TCP connection to the outgoing one. This overhead is lessened if a more lightweight, wireless-specific reliable protocol is used on the last link.

- **The Fast-Retransmit Approach [CI94]:** This approach addresses the issue of TCP performance when communication resumes after a handoff. Unmodified TCP at the sender interprets the delay caused by a handoff process to be due to congestion (since TCP assumes that all delays are caused by congestion) and when a timeout occurs, reduces its window size and retransmits unacknowledged packets. Often, handoffs complete relatively quickly (between a few tens to a couple of hundred milliseconds), and long waits are required by the mobile host before timeouts occur at the sender and packets start getting retransmitted. This is because of coarse retransmit timeout granularities (on the order of 500 ms) in most TCP implementations. The fast retransmit approach mitigates this problem by having the mobile host send a certain threshold number of duplicate acknowledgments to the sender. This causes TCP at the sender to immediately reduce its window size and retransmit packets starting from the first missing one (for which the duplicate acknowledgment was sent). The main drawback of this approach is that it only addresses handoffs and not the error characteristics of the wireless link.

- **Link-level Retransmissions [PAL+95]:** In this approach, the wireless link implements a retransmission protocol coupled with forward error correction at the data-link level. The advantage of this approach is that it improves the reliability of communication independent of the higher-level protocol. However, TCP implements its own end-to-end retransmission protocol. Studies have shown that independent retransmission protocols such as these can lead to degraded performance, especially as error rates become significant [DCY93]. A tight coupling of transport- and link-level retransmission timeouts and policies is necessary for good performance. In particular, information needs to be passed down to the data link layer about timeout values and policies reasonable for co-existence with the higher transport layer policy.

In summary, several schemes have been proposed to improve the performance of TCP in wireless networks.

However, they have the disadvantages described above. We feel that it is possible to design a protocol to solve this problem without these drawbacks. The rest of the paper describes the design, implementation, and performance of such a protocol.

3. The Snoop Protocol

Most current network applications that require reliable transmission use TCP. Therefore, it is desirable to achieve our goal of improving its performance in our network without changing existing TCP implementations in the fixed network. The only components of the network we can expect to have administrative control over are the base stations and the mobile hosts. For transfer of data from a fixed host to a mobile host, we make modifications only to the routing code at the base station. These modifications include caching unacknowledged TCP data and performing local retransmissions based on a few policies dealing with acknowledgments (from the mobile host) and timeouts. By using duplicate acknowledgments to identify packet loss performing local retransmissions as soon as this loss is detected, the protocol shields the sender from the vagaries of the wireless link. In particular, transient situations of very low communication quality and temporary disconnectivity are hidden from the sender. This results in significantly improved performance of the connection, without sacrificing any of the end-to-end semantics of TCP, modifying host TCP code in the fixed network or relinking existing applications. This combination of improved performance, preserved protocol semantics and full compatibility with existing applications is the main contribution of our work.

A preliminary design of a protocol based on these ideas appeared in [ABSK95]. Simulations of the protocol indicated that it was capable achieving the same throughput as unmodified TCP at 10 times higher bit-error rates. These promising results indicated that an implementation would be worthwhile. The simulated protocol was used as the basis of the initial implementation. Several parts of the protocol were changed based on measurements and our experience with it.

3.1 Data Transfer from a Fixed Host

We first describe the protocol for transfer of data from a fixed host (FH) to a mobile host (MH) through a base station (BS). The base station routing code is modified by adding a module, called the *snoop*, that monitors every packet that passes through the connection in either direction. No transport layer code runs at the base station. The snoop module maintains a cache of TCP packets sent from the FH that haven't yet been acknowledged by the MH. This is easy to do since TCP has a cumulative acknowledgment policy for received packets. When a new packet arrives from the FH, snoop adds it to its cache and passes the packet on to

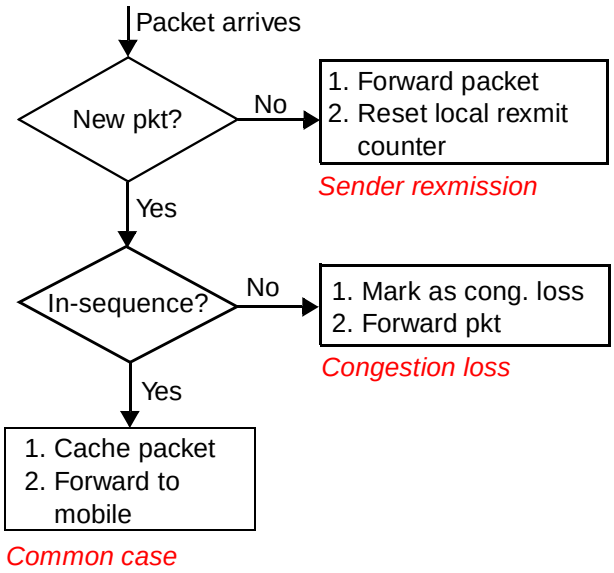

Figure 1. Flowchart for snoop_data().

the routing code which performs the normal routing functions. The snoop module also keeps track of all the acknowledgments sent from the mobile host. When a packet loss is detected (either by the arrival of a duplicate acknowledgment or by a local timeout), it retransmits the lost packet to the MH if it has the packet cached. Thus, the base station (snoop) hides the packet loss from the FH by not propagating duplicate acknowledgments, thereby preventing unnecessary congestion control mechanism invocations.

The snoop module has two linked procedures, snoop_data() and snoop_ack(). Snoop_data() processes and caches packets intended for the MH while snoop_ack() processes acknowledgments (ACKs) coming from the MH and drives local retransmissions from the base station to the mobile host. The flowcharts summarizing the algorithms for snoop_data() and snoop_ack() are shown in Figures 3 and 2 and are described below.

3.1.1 Snoop_data().

Snoop_data() processes packets from the fixed host. TCP implements a sliding window scheme to transmit packets based on its congestion window (estimated from local computations at the sender) and the flow control window (advertised by the receiver). TCP is a byte stream protocol and each byte of data has an associated sequence number. A TCP packet (or segment) is identified uniquely by the sequence number of its first byte of data and its size. At the BS, snoop keeps track of the last sequence number seen for the connection. One of several kinds of packets can arrive at the BS from the FH, and snoop_data() processes them in different ways:

1. *A new packet in the normal TCP sequence:* This is the common case, when a new packet in the normal increasing sequence arrives at the BS. In this case the packet is added to the snoop cache and forwarded on to the MH. We do not perform any extra copying of data while doing this. We also place a timestamp on one packet per transmitted window in order to estimate the round-trip time of the wireless link. The details of these steps are described in Section 4.

2. *An out-of-sequence packet that has been cached earlier:* This is a less common case, but it happens when dropped packets cause timeouts at the sender. It could also happen when a stream of data following a TCP sender fast retransmission arrives at the base station. Different actions are taken depending on whether this packet is greater or less than the last acknowledged packet seen so far. If the sequence number is greater than the last acknowledgment seen, it is very likely that this packet didn't reach the MH earlier, and so it is forwarded on. If, on the other hand, the sequence number is less than the last acknowledgment, this packet has already been received by the MH. At this point, one possibility would be to discard this packet and continue, but this is not always the best thing to do. The reason for this is that the original ACK with the same sequence number could have been lost due to congestion while going back to the FH. In order to facilitate the sender getting to the current state of the connection as fast as possible, a TCP acknowledgment corresponding to the last ACK seen at the BS is generated by the snoop module (with the source address and port corresponding to the MH) and sent to the FH.

3. *An out-of-sequence packet that has not been cached earlier:* In this case the packet was either lost earlier due to congestion on the wired network or has been delivered out of order by the network. The former is more likely, especially if the sequence number of the packet (i.e, the sequence number of its first data byte) is more than one or two packets away from the last one seen so far by the snoop module. This packet is forwarded to the MH, and also marked as having been retransmitted by the sender. Snoop_ack() uses this information to process acknowledgments (for this packet) from the MH.

3.1.2 Snoop_ack().

Snoop_ack() monitors and processes the acknowledgments (ACKs) sent back by the MH and performs various operations depending on the type and number of acknowledgments it receives. These ACKs fall into one of three categories:

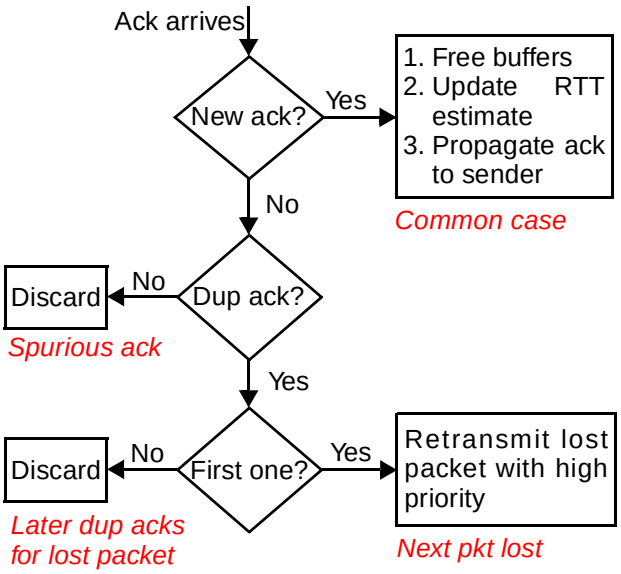

Figure 2. Flowchart for snoop_ack().

1. A *new* ACK: This is the common case (when the connection is fairly error-free and there is little user movement), and signifies an increase in the packet sequence received at the MH. This acknowledgment initiates the cleaning of the snoop cache and all acknowledged packets are freed. The round-trip time estimate for the wireless link is also updated at this time. This estimate is not done for every packet, but only for one packet in each window of transmission, and only if no retransmissions happened in that window. The last condition is needed because it is impossible in general to determine if the arrival of an acknowledgment for a retransmitted packet was for the original packet or for the retransmission [KP87]. Finally, the acknowledgment is forwarded to the FH.

2. A *spurious* ACK: This is an acknowledgment less than the last acknowledgment seen by the snoop module and is a situation that rarely happens. It is discarded and the protocol continues.

3. A *duplicate* ACK (DUPACK): This is an ACK that is identical to a previously received one. In particular, it is the same as the last ACK seen so far. In this case the next packet in sequence from the DUPACK has not been received by the MH. However, some subsequent packets in the sequence have been received, since the MH generates a DUPACK for each TCP segment received out of sequence. One of several actions is taken depending on the type of duplicate acknowledgment and the current state of snoop:
   - The first case occurs when we receive a DUPACK for a packet that is either not in the snoop cache or has been marked as having been retransmitted by

the sender. If the packet is not in the cache, it needs to be resent from the FH, perhaps after invoking the necessary congestion control mechanisms at the sender. If the packet was marked as a sender-retransmitted packet, the DUPACK needs to be routed to the FH because the TCP stack there maintains state based on the number of duplicate acknowledgments it receives when it retransmits a packet. Therefore, both these situations require the DUPACK to be routed to the FH.

- The second case occurs when snoop gets a DUPACK that it doesn't expect to receive for the packet. This typically happens when the first DUPACK arrives for the packet, after a subsequent packet in the stream reaches the MH. The arrival of each successive packet in the window causes a DUPACK to be generated for the lost packet. In order to make the number of such DUPACKs as small as possible, the lost packet is retransmitted as soon as the loss is detected, and at a higher priority than normal packets. This is done by maintaining two queues at the link layer for high and normal priority packets. In addition, snoop also estimates the maximum number of duplicate acknowledgments that can arrive for this packet. This is done by counting the number of packets that were transmitted after the lost packet prior to its retransmission.
- The third case occurs when an "expected" DUPACK arrives, based on the above maximum estimate. The missing packet would have already been retransmitted when the first DUPACK arrived (and the estimate was zero), so this acknowledgment is discarded. In practice, the retransmitted packet reaches the MH before most of the later packets do and the BS sees an increase in the ACK sequence before all the expected DUPACKs arrive.

Retransmitting packets at a higher priority using a fast queue improves performance at all error rates. The benefits of this approach are most visible at low to medium bit-error rates. This is a consequence of the average queue lengths in the retransmission queue. At high bit-error rates, most packets need to be retransmitted, and there is no significant advantage to be derived from maintaining two queues. However, at low and medium error rates, the fast queue enables retransmitted packets to reach the mobile host sooner than if there were only one queue, leading to improved throughput.

Snoop keeps track of the number of local retransmissions for a packet, but resets this number to zero if the packet arrives again from the sender following a timeout or a fast retransmission. In addition to retransmitting packets depending on the number and type of acknowledgments, the snoop protocol also performs retransmissions driven by timeouts. This is described in more detail in the section on Implementation (Section 4).

3.2 Data Transfer from a Mobile Host

It is unclear that a protocol with modifications made only at the base station can substantially improve 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 acknowledgments. For example, simply caching packets at the base station and retransmitting them as necessary will not be very useful, since the bulk of the packet losses are likely to be from the mobile host to the base station. 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. Since TCP performs retransmissions on the basis of round-trip time estimates for the connection, sender timeouts for packets lost on the (first) wireless link will happen much later than they should.

Our design involves a slight modification to the TCP code at the mobile host. At the base station, we keep track of the packets that were lost in any transmitted window, and generate negative acknowledgments (NACKs) for those packets back to the mobile. 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 NACKs 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 NACKs as a bit vector can ensure that the relative fraction of the sparse wireless bandwidth consumed by NACKs is relatively low.

Our implementation of NACKs is based on using the Selective Acknowledgment (SACK) option in TCP [JB88]. Selective acknowledgments, currently unsupported in most TCP implementations, were introduced to improve TCP performance for connections on "long fat networks", or LFNs. These are networks where the capacity of the network (the product of bandwidth and round-trip time) is large. SACKs were proposed to handle multiple dropped packets in a window, but the current TCP specification (JBB92) does not include this feature. The basic idea here is that in addition to the normal cumulative ACKs the receiver can inform the sender which specific packets it didn't receive. The snoop protocol uses SACKs to cause 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 will be to enable SACK processing. No changes of any sort are required in any of the fixed hosts.

We have implemented the ability to generate SACKs at the base station and process them at the mobile hosts to retransmit lost packets and are currently measuring the perfor-

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.