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 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.

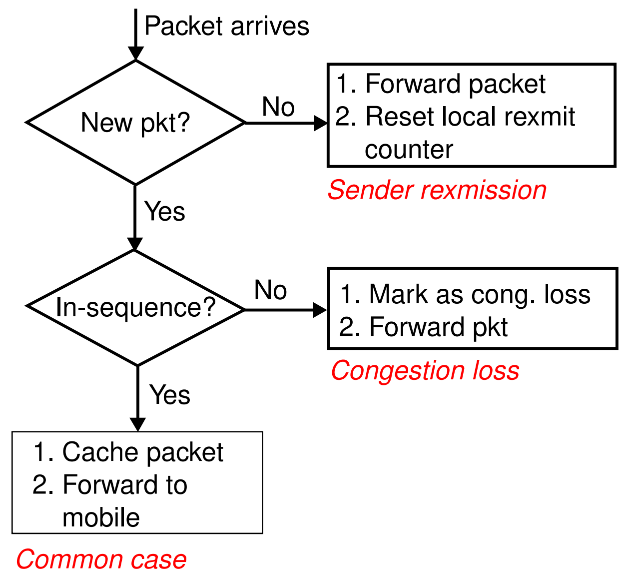

Figure 1. Flowchart for snoop_data().

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:

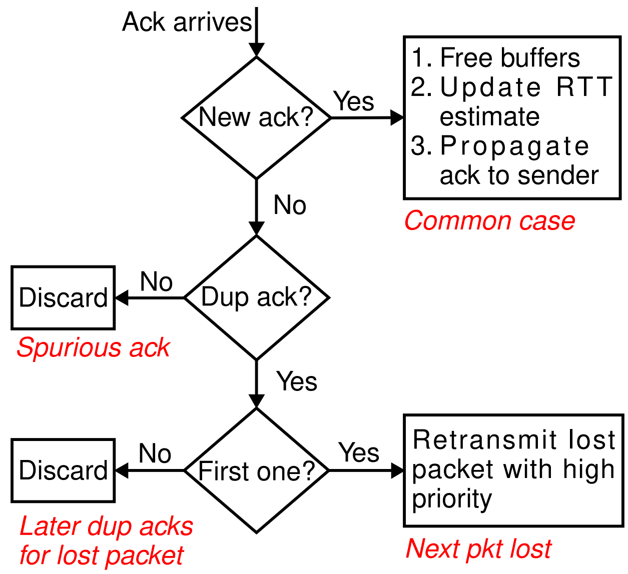

Figure 2. Flowchart for snoop_ack().

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-

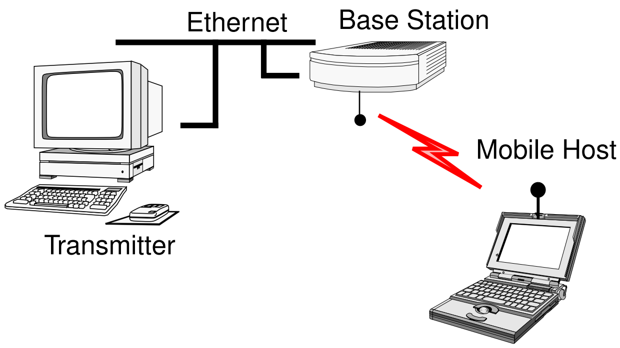

Figure 3. Network topology for experiments.

mance of transfers from the mobile host.

3.3 Mobility Handling

Handoffs in our system are based on multicast [Dee91]. Most often, they are mobile-initiated and occur when the mobile host discovers a base station with a stronger signal than the current one. When a handoff is requested by the mobile host or anticipated by the base station, the nearby base stations also begin receiving packets destined for the mobile host. This allows them to begin building up their snoop caches for this mobile host. However, during this period these nearby "buffering" base stations cannot snoop on any acknowledgments sent from the mobile host. Once the handoff occurs information to synchronize the snoop cache is sent to the base station that the mobile host has transferred to. This scheme has the advantage that the duration of a handoff is quite short (because the snoop cache has been primed in advance) and the sender can continue sending packets without experiencing much delay.

We have implemented the algorithms for handoff and integrated it with the snoop protocol. Preliminary measurements indicate that handoffs are completed between 10 and 25 ms.

4. Implementation

We have implemented the snoop protocol on a testbed consisting of IBM ThinkPad laptops and *i*486 base stations running BSD/OS 2.0 from BSDI, communicating over an AT&T Wavelan. The maximum raw bandwidth of the Wavelan is about 2Mb/s per mobile-host. The implementation currently supports bulk transfers to and from mobile hosts and supports smooth handoffs. The network topology is shown in Figure 3.

The state maintained by snoop is soft state and can easily be reconstructed from scratch by snooping on a few packets and acknowledgments. The snoop cache is maintained as a circular buffer of packets, consisting mainly of pointers to kernel mbufs [LMKQ89] and some other associated information that includes the packet sequence number, its size, the number of local retransmissions, and a flag set if the packet was retransmitted by the sender. In general, the size of the cache needs to be large enough to handle the maximum transmission window size. In practice, we set a "high-water mark" on the cache: the only packets accepted into the cache after this point is reached are those that are out of order and earlier in sequence than the last one seen. Other packets are forwarded to the mobile host without being cached. This is because it is more important for the older, rather than newer, packets to be cached and retransmitted, since they will cause sender timeouts earlier.

Several studies have shown that one of the predominant costs of TCP is the copying of data [CJRS89, KP93]. We use the reference counting mechanism present in kernel mbufs to avoid data copying in the snoop protocol. Thus, we do not incur any extra overhead associated with copying at the base station. When error rates are relatively low, the protocol overhead is small -- an incoming packet is added to the cache without copying it, and it is forwarded on to the mobile host. A small number of state variables (e.g., the last sequence number seen) are updated. When a new acknowledgment arrives at the base station, we forward it on to the fixed host and clean the snoop cache by freeing the packets corresponding to packets already acknowledged by the mobile. The last link round-trip time estimate is updated once per transmission window.

In addition to retransmitting packets depending on the number and type of acknowledgments received, the snoop protocol also performs retransmissions driven by timeouts. There are two types of timer interrupts in the protocol, the *round-trip* timer and the *persist* timer. The round-trip timer is based on the estimate of the smoothed round-trip time (srtt) of the last link. We compute this using the standard adaptive technique, $srtt=(1-\alpha)*old\_srtt+\alpha *curr\_rtt$, with α set to 0.25, so that integer shift operations can be used. The packet is retransmitted if an acknowledgment hasn't been received in twice this time. In order to limit the amount of time spent processing timer interrupts, we don't timeout more frequently than a threshold time, currently set to 40ms. Additionally, we trigger this timeout only after the first retransmission of a packet from the snoop cache, caused by the arrival of a duplicate acknowledgment. This also ensures that a negligible number of (unnecessary) retransmissions occur for packets that have already reached the mobile host.

The persist timer triggers a retransmission if there are unacknowledged packets in the cache, and if there has been no activity either from the sender or receiver for 200ms. This timer also sets the number of expected DUPACKs to zero and the next expected acknowledgment to one more than the last ACK seen so far. These timers and their associated retransmissions are critical when packet losses are high (e.g., due to interference or movement), since they increase the number of transmission attempts and thereby increase the likelihood of the packet getting through sooner to the

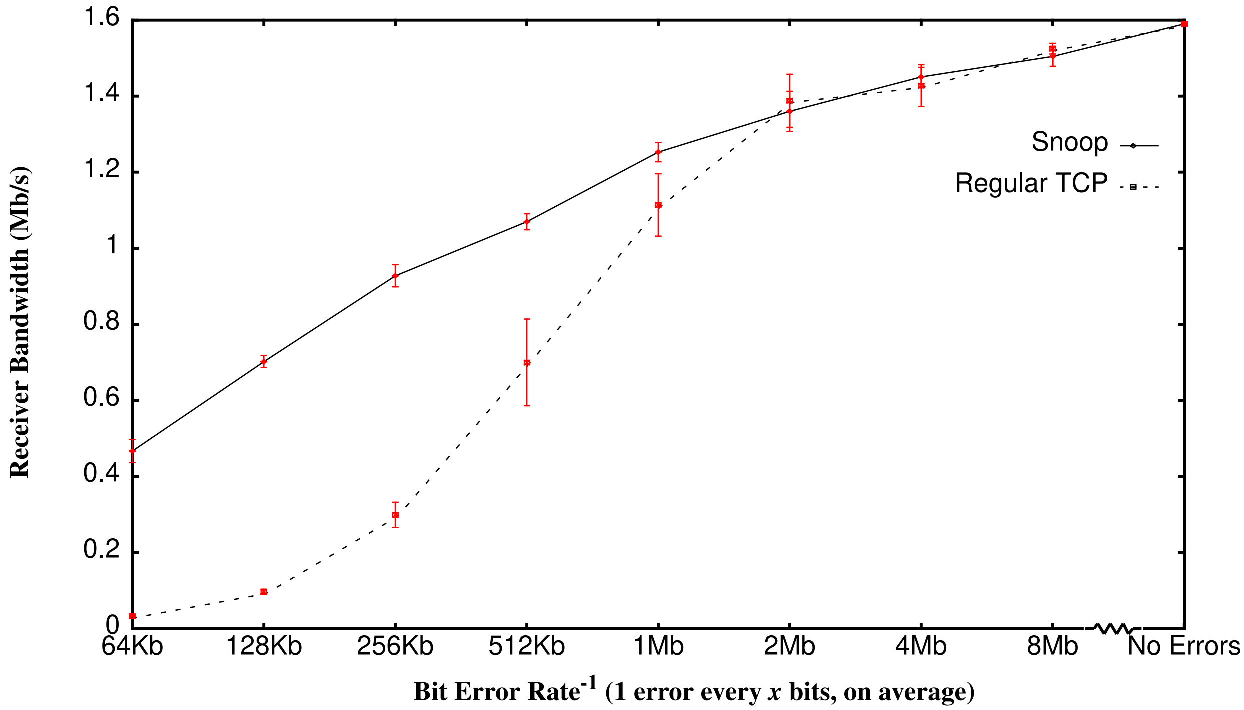

Figure 4. Throughput received by the mobile host at different bit-error rates ($\log_2$ scale). The vertical error bars show the standard deviations of the receiver throughput.

mobile host.

5. Performance

We performed several experiments with the snoop protocol on our wireless testbed and compared the resulting performance with unmodified TCP. We present the results of these experiments in this section. In the presence of no packet losses, the maximum throughput achieved by a TCP connection over the wireless link was about 1.6 Mbits/s. The rated maximum raw bandwidth of the wireless link was 2 Mbits/s. We present the results of data transfer from a fixed sender to a mobile receiver. These were obtained using the network configuration shown in Figure 3. The sender TCP stack was based on TCP Reno, an implementation that supported fast retransmissions upon the arrival of three duplicate acknowledgments. The maximum possible window size for the connection was 64 KBytes and the maximum TCP segment size was 1460 bytes.

In order to measure the performance of the implementation under controlled conditions, we used a Poisson-distributed bit error model. We generated a Poisson distribution for each bit-error rate and changed the TCP checksum of the packet at the base station if the error generator determined that the packet should be dropped at the receiver, before forwarding the packet over the wireless link. The same operation was done for packets (acknowledgments) from the mobile host. Each run involved a 10 MByte transfer and this was repeated ten times at each error rate. Figure 4 compares the throughput of a connection using the snoop protocol with that of a connection using an unmodified TCP implementation, for various Poisson-distributed bit-error rates shown on a log scale. The vertical error bars in the figure show the standard deviation of the receiver throughput.

We see that for error rates of over 5×10^{-7} (close to the 2 Mb point on the x-axis of the graph) the snoop protocol performs significantly better than unmodified TCP, achieving a bandwidth improvement factor of 1 to 20 depending on the bit error rate. In fact, the snoop protocol was robust and completed the run even when every other packet was being dropped over the last link, while the regular TCP connection didn't make any progress. Under conditions of very low bit error rates ($< 5 \times 10^{-7}$), we see little difference between the snoop protocol and unmodified TCP. At such low bit errors there is typically less than one error per transmitted window and unmodified TCP is quite robust at handling these. At these low error rates, snoop behaves as is it were not present and this ensures no degradation in performance.

A more detailed picture of the behavior of the connection can be seen can be seen in Figure 5, which plots the sequence numbers of the received TCP packets versus time for one of the experiments. These values were obtained using the tcpdump [MJ93] network monitoring tool. The figure shows the comparison of sequence number progres-

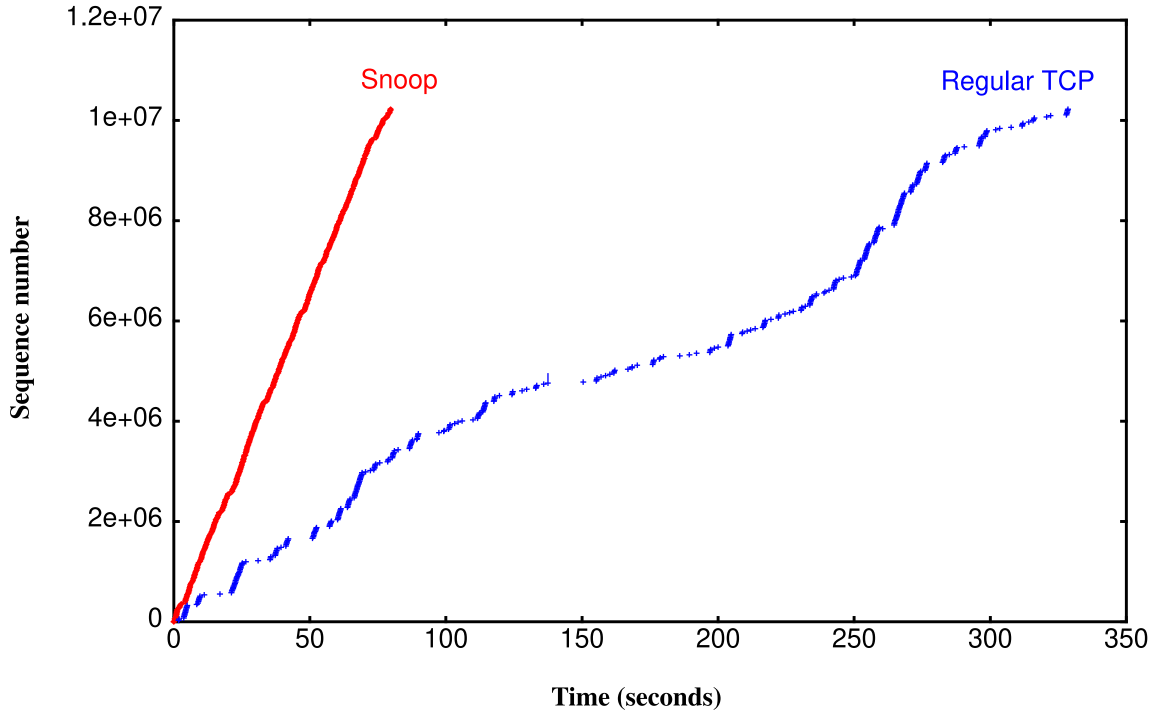

Figure 5. Sequence numbers for transfer to mobile host over channel with $3.8x10^{-6}$ (1/256 Kbits) BER.

sion in a connection using the snoop protocol and a connection using unmodified TCP for a Poisson-distributed bit error rate of $3.9x10^{-6}$ (a bit error every 256 Kbits on average). We see that the snoop protocol maintains a high and consistent throughput. On the other hand, regular TCP unnecessarily invokes congestion control procedures several times during the duration of the connection. This phenomenon appears as the flat and empty regions of the curve and degrades the throughput significantly. For this particular run, the aggregate bandwidth with the snoop protocol was about 1 Mbit/s, while it was only about 0.25 Mbit/s for regular TCP.

In summary, the results for moderate to high error rates are very encouraging. For bit error rates greater than $5x10^{-7}$ we see an increase in throughput by a factor of up to 20 times compared to regular TCP depending on the bit error rate. For error rates that are lower than this, there is little difference between the performance of snoop and regular TCP, showing that the overhead caused by snoop is negligible.

6. Comparisons with Other Approaches

In this section, we compare the snoop protocol to the other protocols described in Section 2.. All these protocols seek to improve end-to-end performance by minimizing the number of sender timeouts and retransmissions. In addition, the time for which the connection remains idle after a handoff is completed must be made as small as possible [CI94].

As mentioned in Section 2., the main drawback of the split connection approach is that the semantics of TCP acknowledgments are violated. In contrast, the snoop protocol maintains the end-to-end semantics of the TCP connection between the fixed and mobile hosts by not generating any artificial acknowledgments. Handoffs in this approach require the transfer of a significant amount of state. For example, I-TCP handoff times vary between 265 and 1400 ms depending on the amount of data in the socket buffers that need to be transferred from one base station to another [BB95]. The snoop protocol performs handoffs based on multicast as described in Section 3.3 and typical completion times are between 10 and 25 ms.

Caceres and Iftode [CI94] show that show that a mechanism based on fast retransmissions is quite successful in reducing delays after a handoff. However, one drawback of their approach is that it doesn't handle errors well. Another problem is that the sender usually reduces the transmission window size before starting fast retransmissions. Furthermore, several TCP implementations don't support fast retransmissions. In contrast, the snoop mechanism has the advantage that the connection will not be idle for much time after a handoff since the new base station will forward cached packets as soon as the mobile host is ready to receive them. One other advantage of this approach is that it results in low-latency handoffs for non-TCP streams as well, especially continuous media streams.

The snoop protocol is similar to link-level retransmissions over the wireless links in that both schemes perform retransmissions locally to the mobile host. However, the snoop protocol is closely coupled to TCP, and so does not perform many redundant (and possibly competing) retransmissions (i.e, few packets are retransmitted both locally and by the sender because of timeouts). Packets retransmitted by the sender that arrive at a base station are already cached there. This happens most often because the sender often transmits half a window's worth of data and several of these packets are already in the cache. In our experiments, a very small percentage of these packets actually arrived because of sender timeouts. In order for link-level retransmissions to perform well, they need to be closely coupled with the higher-level protocols, especially if those also provide reliable service via retransmissions. Also, there are several higher-level protocols that don't require reliable transfer, but those packets may also be retransmitted multiple times on the wireless link. This is not necessary, since packets arriving late are useless for several applications, and retransmissions at the link-level are not required for them.

7. Future Work

We are currently in the process of measuring and optimizing the performance of the snoop protocol under various situations. These include wide-area connections to a mobile host, data transfers from a mobile host, and multicast-based handoffs. We are also working on characterizing the behavior of TCP connections and the snoop protocol in the presence of real-life sources of interference.

In addition to this, we have started working on improving the TCP performance of the Metricom system, a metropolitan-area packet relay network. This system has multiple wireless hops from the base station to a mobile host and operates at bandwidths of about 100 Kbits/s. Although there are several differences between this and the Wavelan, we believe that with minor modifications the snoop protocol will result in improved performance in this environment.

Wireless networks of the future are likely to be heterogeneous where each host will simultaneously be connected to different wireless interfaces, that may interfere with each other. An example of this is an in-building Wavelan network and a campus-wide packet relay network, that also extends inside buildings. The problems of improving TCP performance, routing and handoff in such heterogeneous networks, characterizing the impact of interference on connection quality, and support for network-characteristic-aware applications are challenging ones with significant practical value [Kat94].

8. Summary

We have presented a protocol to improve the performance of TCP in networks with wireless links and mobile hosts. This protocol works by modifying the network-layer software at the base station, and involves no other changes to any of the fixed hosts elsewhere in the network. The main idea is the caching of packets intended for the mobile host at the base station and performing local retransmissions across the wireless link. We have implemented this protocol on a wireless testbed consisting of IBM ThinkPad laptops and *i*486 base stations running BSD/OS 2.0 communicating over a 2 Mbits/s AT&T Wavelan. Experiments show that the protocol is significantly more robust than regular TCP at dealing with unreliable links and multiple errors in a window; we have achieved performance improvements of up to 20 times over normal TCP/IP for data transfer from a fixed to a mobile host across a wide range of bit error rates.

9. Acknowledgments

We thank Kimberly Keeton, Bruce Mah, Steve McCanne and the anonymous reviewers for several comments and suggestions that greatly improved the quality of this paper. We also thank Jim O'Toole for pointing out the problems of counting acknowledgments on a noisy channel that led us to investigate other approaches and retransmit packets at a higher priority.

10. References

[ABSK95] E. Amir, H. Balakrishnan, S. Seshan, and R. H. Katz. Efficient TCP over Networks with Wireless Links. In *Proc. HotOS-V*, May 1995.

[BB94] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. Technical Report DCS-TR-314, Rutgers University, October 1994.

[BB95] A. Bakre and B. R. Badrinath. Handoff and System Support for Indirect TCP/IP. In *Proc. Second Usenix Symp. on Mobile and Location-Independent Computing*, April 1995.

[Bra89] R. T. Braden. *Requirements for Internet Hosts – Communication Layers*, October 1989. RFC-1323.

[CI94] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE JSAC*, 13(5), June 1994.

[CJRS89] D. C. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *IEEE Communication Magazine*, June 1989.

[DCY93] A. DeSimone, M. C. Chuah, and O. C. Yue. Throughput Performance of Transport-Layer Protocols over Wireless LANs. In *Proc. Globecom '93*, December 1993.

[Dee91] S. E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, December 1991.

[Jac88] V. Jacobson. Congestion avoidance and control. In *SIGCOMM 88*, August 1988.

[JB88] V. Jacobson and R. T. Braden. *TCP Extensions for Long Delay Paths*. RFC, Oct 1988. RFC-1072.

[JBB92] V. Jacobson, R. T. Braden, and D. A. Borman. *TCP Extensions for High Performance*. RFC, May 1992. RFC-1323.

[Kat94] R. H. Katz. Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Communications*, 1(1), 1994.

[KP87] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. In *SIGCOMM 87*, August 1987.

[KP93] J. Kay and J. Pasquale. The Importance of Non-Data Touching Processing Overheads in TCP/IP. In *Proc. SIGCOMM '93 Conf.*, San Francisco, CA, Sep 1993.

[LMKQ89] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.3 BSD UNIX Operating System*. Addison-Wesley, Reading, MA, Nov 1989.

[MJ93] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In *Proc. Winter '93 USENIX Conference*, San Diego, CA, January 1993.

[PAL+95] S. Paul, E. Ayanoglu, T. F. LaPorta, K. H. Chen, K. K. Sabnani, and R. D. Gitlin. An Asymmetric Link-Layer Protocol for Digital Cellular Communications. In *Proc. InfoComm '95*, 1995.

[Pos81] J. B. Postel. *Transmission Control Protocol*. RFC, SRI International, Menlo Park, CA, September 1981. RFC-793.

[Ste94] W. R. Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley, Reading, MA, Nov 1994.

[YB94] R. Yavatkar and N. Bhagwat. Improving End-to-End Performance of TCP over Mobile Internetworks. In *Mobile 94 Workshop on Mobile Computing Systems and Applications*, December 1994.