In ACM Wireless Networks Journal (WINET), 1(4), December 1995.

Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks[1]

Hari Balakrishnan, Srinivasan Seshan, and Randy H. Katz
`{hari,ss,randy}@CS.Berkeley.EDU`
Computer Science Division, Department of Electrical Engineering and Computer Science,
University of California at Berkeley, Berkeley, CA 94720-1776.

Abstract

TCP is a reliable transport protocol tuned to perform well in traditional networks where congestion is the primary cause of packet loss. However, networks with wireless links and mobile hosts incur significant losses due to bit-errors and handoff. This environment violates many of the assumptions made by TCP, causing degraded end-to-end performance. In this paper, we describe the additions and modifications to the standard Internet protocol stack (TCP/IP) to improve end-to-end reliable transport performance in mobile environments. The protocol changes are made to network-layer software at the base station and mobile host, and preserve the end-to-end semantics of TCP. One part of the modifications, called the snoop module, caches packets at the base station and performs local retransmissions across the wireless link to alleviate the problems caused by high bit-error rates. The second part is a routing protocol that enables low-latency handoff to occur with negligible data loss. We have implemented this new protocol stack on a wireless testbed. Our experiments show that this system is significantly more robust at dealing with unreliable wireless links than normal TCP; we have achieved throughput speedups of up to 20 times over regular TCP and handoff latencies over 10 times shorter than other mobile routing protocols.

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, sporadic high bit-error rates and temporary disconnections that network protocols and applications must deal with. 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 often result in either packet loss or variation in packet delays. Handoffs typically last anywhere between a few tens to a few hundreds of milliseconds in most systems.

Reliable transport protocols such as TCP [22, 23, 5] 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. It 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 caused by congestion.

In the presence of the high bit-error rates, intermittent connectivity and handoffs characteristic of wireless environments, 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 [11]) and

1. This work was supported by ARPA Contract J-FBI-93-153.
This paper was in part presented at the ACM Mobile Computing and Networking Conference (Mobicom '95), Berkeley, California, 14-15 November 1995.

resets its retransmission timer (Karn's Algorithm [14]). 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 [6].

In this paper, we describe the design and implementation of a protocol stack 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 standard Internet network-layer (IP) software at the base station and mobile host. Two different changes deal with the wireless transmission losses and handoff-related losses. The snoop module deals with bit-error losses while the routing protocol eliminates the losses during handoff. The snoop modifications consist 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 the snoop module makes the protocol significantly more robust at dealing with multiple packet losses in a single window as compared to regular TCP. The routing protocol multicasts packets to several base stations in the target mobile host's area. This multicast combined with intelligent buffering at these base stations allows mobile hosts to roam without incurring packet losses due to handoff. Our measurements show that handoff completes in 5 to 70 ms and causes negligible degradation of end-to-end TCP performance. The snoop module and routing protocol combine to alleviate the TCP performance problems present in typical wireless networks.

The rest of this paper is organized as follows. In Section 2, we describe and evaluate some design alternatives and related work that addresses this problem. In Section 3 and Section 4, we describe algorithms for our snoop module and routing protocols respectively. Section 5 describes the interaction between the snoop module and the handoff protocol. We describe our implementation and the modifications to the router software at the base station and mobile host in Section 6 and the results of several of our experiments in Section 7. Section 8 compares our protocols with some of the other alternatives published in the literature. We discuss our future plans in Section 9 and conclude with a summary in Section 10.

2. Design Alternatives and Related Work

TCP performance is an important factor in determining the current and future usability of wireless networks, since several popular applications such as ftp, telnet, and WWW (http) are built upon it. It is desirable to improve its performance in wireless networks *without any modifications to the fixed hosts*, since this will enable seamless integration of mobile devices communicating over wireless links with the rest of the Internet.

Recently, several reliable transport-layer protocols for networks with wireless links have been proposed [3, 4, 6, 24] 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 8 we present a more detailed comparison of these schemes with our protocol.

- **The Split Connection Approach:** The Indirect-TCP (I-TCP) protocol [3, 4] 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 [24]. During a handoff, the I-TCP connections must be moved from the base station currently forwarding data to another one. This is done by transferring the state associated with the two connections to the new base station.

  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. *Loss of 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.

4. *Handoff Latency:* The state maintained at a base station in I-TCP consists mainly of a set of socket buffers. Since this state must be moved to another base station during handoff, the latency of handoff is proportional to the size of these socket buffers. The I-TCP handoffs range from 265 ms for empty socket buffers to 1430 ms for 32KByte socket buffers [4].

- **The Fast-Retransmit Approach [6]:** 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 [20]:** 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 [9]. In such systems, a tight coupling of transport- and link-level retransmission timeouts and policies is necessary to avoid redundant retransmissions.

In summary, several schemes have been proposed to improve the performance of TCP in wireless networks. However, they have the disadvantages described above. Furthermore, while a variety of schemes for low-latency, low data loss handoffs have been proposed [1, 10], none of these address the problems of reliable protocols in wireless networks. 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 stack.

3. The Snoop Module

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 and performing local retransmissions as soon as this loss is detected, the module 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.

For transfer of data from a mobile host to a fixed host, we detect missing packets at the base station and generate negative acknowledgments for them. These negative acknowledgments are sent to the mobile host (the sender), which then processes them and retransmits the corresponding missing packets. This requires modifications to both the fixed and mobile hosts.

These mechanisms together improve the performance of the connection in both directions, 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 [2]. 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, and we used the simulated protocol as the basis of our 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 changes to 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 *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, the snoop module 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.

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 1 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:

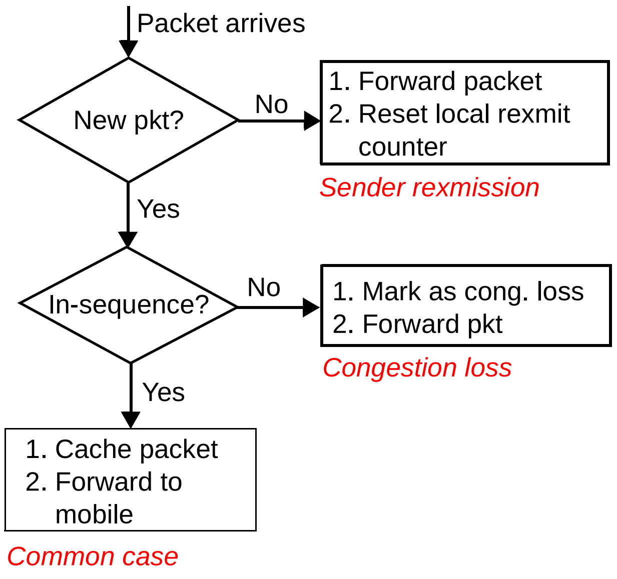

Figure 1. Flowchart for snoop_data().

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 6.

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 duplicate acknowledgments that arrive 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:

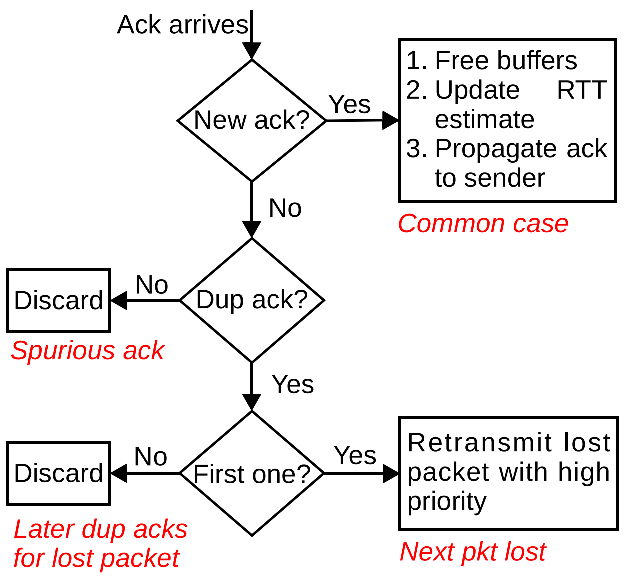

Figure 2. Flowchart for snoop_ack().

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 ACK 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 [14]. 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 packet processing 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 highest cumulative 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 the DUPACK is 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 the snoop module 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, following a packet loss. 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 (because it was retransmitted at a higher priority) and the BS sees an increase in the ACK sequence before all the expected DUPACKs arrive.

Retransmitting packets at a higher priority improves performance at all error rates. This enables retransmitted packets to reach the mobile host sooner, reducing the number of duplicate ACKs and leading to improved throughput.

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 module also performs retransmissions driven by timeouts. This is described in more detail in the section on implementation (Section 6).

3.2 Data Transfer from a Mobile Host

It is unlikely 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 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 to improve this situation 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. The mobile host used these NACKs to selectively retransmit lost packets.

Our implementation of NACKs is based on using the Selective Acknowledgment (SACK) option in TCP [12]. Selective acknowledgments, currently unsupported in most TCP implementations, were introduced to improve TCP performance for connections on “long fat networks” (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 [13] 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 module 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 is to enable SACK processing. No changes of any sort are required in any of the fixed hosts. Also, SACKs are generated by snoop 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.

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 performance of transfers from the mobile host.

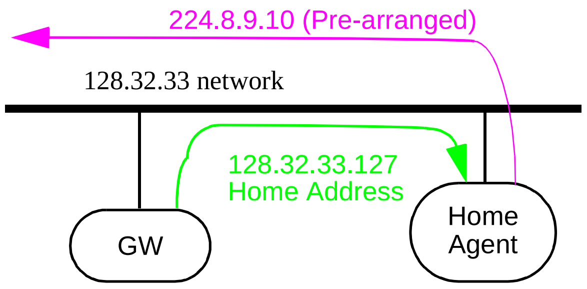

Figure 3. Home Agent Encapsulation.

4. The Routing Protocol

In this section, we describe our approach to solving the problems associated with user mobility in cellular wireless networks. The undesirable effects of user movement include packet losses, disruptions in connectivity and increased latencies. When a mobile host moves between the cells of a wireless system, the route taken by data between it and the fixed host must be updated. This update of routing information constitutes handoff. In systems such as Mobile IP [21], packets traversing the network during a handoff are either lost or experience unusually long delays. The change in network behavior adversely affects end-to-end TCP performance in the system.

Handoffs in our system use multicast [8] and intelligent buffering in nearby base stations to eliminate data loss and provide consistent performance [17]. The basic routing strategy is similar to the Mobile IP protocol. This strategy provides a mechanism to deliver packets from fixed hosts to mobile hosts. However, our routing protocol differs from Mobile IP in order to support low-latency handoff and to reduce packet loss and delay variation during handoff. In our scheme, packets from mobile hosts to fixed hosts utilize the normal IP routing support. There are three basic parts to our routing of packets to a mobile host. The first part deals with delivering the packet to a machine that understands mobility. This uses the home agent concept from Mobile IP. The second responsibility of the routing system is to determine the physical location of the mobile host. Finally, the routing system must support the delivery of packets from the home agent to the mobile host.

Each MH is assigned a long-term IP address (home address) associated with its home location. The MH's home agent (HA) intercepts any packets transmitted to this home address using proxy ARP. Each MH is also assigned a temporary IP multicast address. The home agent encapsulates packets destined for the MH and forwards them to its associated multicast group. The members of this multicast include the base stations in the vicinity of the mobile host, but the mobile host itself does not join the group. An example of this interception and encapsulation is shown in Figure 3. Standard IP multicast routing is used to deliver the packet to the MH's current base station.

The second task of the routing system is to determine the current location of the MH. Each BS periodically broadcasts a beacon message to all the MHs in range of it. Each MH keeps track of all the recent beacons it has received to approximate its current location and motion. The MH uses statistics such as the received signal strength of the beacons and communication quality to identify which BSs are nearby. The MH also determines which wireless network cell it should join as well as which cells it is likely to handoff to in the near future. Based on this determination, the MH configures the routing from the between the HA and the various BSs.

The delivery of packets from the HA to the BS utilizes the dynamic routing provided by IP multicast. The BS responsible for the cell containing the MH joins the IP multicast group. Each packet transmitted from the HA on the multicast group is forwarded by this BS, the primary, to the MH. At any instant of time, there is at most one primary base station in the system for a given mobile host. In addition, BSs that are identified as likely handoff targets are asked to join the multicast group. These BSs do not forward the packets from the multicast group to the wireless network. Instead, each of these BSs buffer the last several packets transmitted from the HA. Typically, handoffs will be to cells whose BSs have been primed for a MH. When a MH enters such a cell, the new primary BS

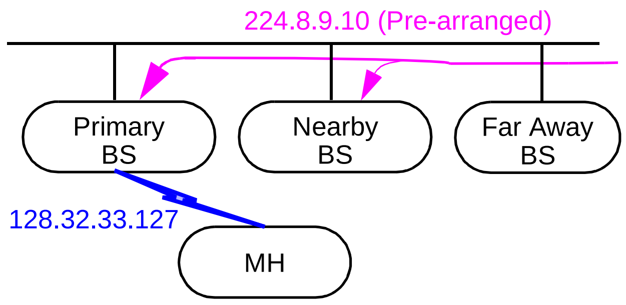

Figure 4. Home Agent to Mobile Host Routing.

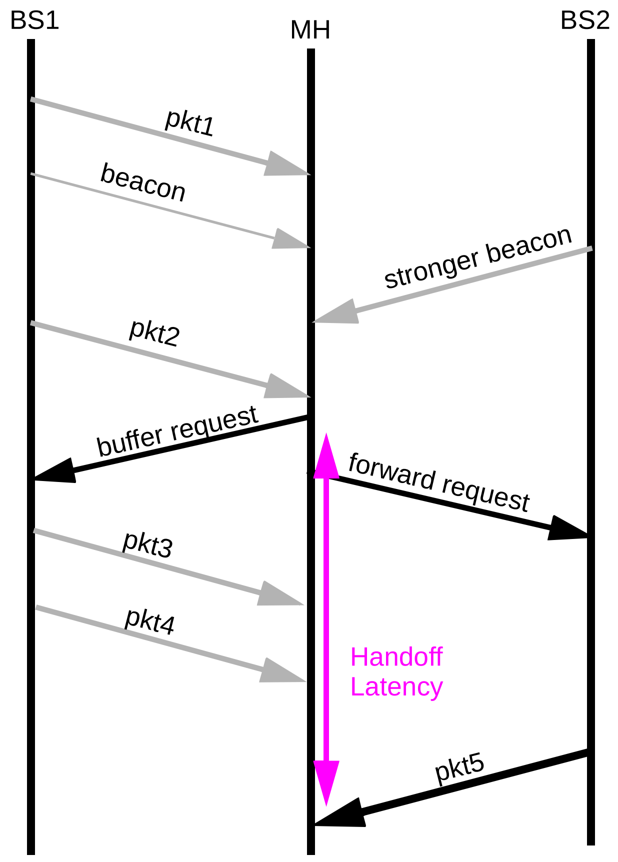

Figure 5. Typical handoff messaging.

begins transmitting packets from its buffer of packets. Since the data “in-flight” from the FH is delivered directly from the new BS without having to forward it from the previous BS, this handoff scheme has minimal data loss during handoff and incurs no delays due to forwarding. This routing of data between the HA and the MH is shown in Figure 4.

Handoffs are mobile-initiated and occur when the mobile host discovers a base station with a stronger signal than the current one. The sequence of events typical of a handoff are shown in Figure 5. The figure shows what happens when a mobile host moves from BS1’s cell to BS2’s cell. The handoff begins when the MH receives a new beacon measurement. Based on its beacon measurements and some hysteresis, the MH computes that BS1 should be buffering (rather than forwarding) packets and that BS2 should be forwarding packets. Using hysteresis ensures that very frequent and unnecessary handoffs don’t occur. After comparing the desired state for each of the BSs in the area with their current state, the MH transmits a set of control messages to the various BSs. These control messages request each BS to either begin or end forwarding and buffering of packets. While these requests are being delivered and processed, packets continue to arrive from BS1. The control message to activate forwarding on BS2 also includes a list of unique identifiers for the last several packets received by the MH. This allows the new primary BS, BS2, to determine which packets in its buffer have already been delivered to the MH by the previous forwarding BS. After synchronizing its buffer, the BS2 begins forwarding packets from the buffer and the multicast group to the MH.

By setting up the routing in advance, the actual duration and amount of data loss of handoff is greatly reduced. This allows handoff to complete without affecting the performance of data transfers using TCP or other protocols. However, in addition to routing there is some amount of state associated with the snoop module at the base station that either needs to be explicitly transferred to the new forwarding base station or be recreated there. In order to meet the goal of low-latency handoffs, we do not explicitly transfer any state during handoff. This process is explained in more detail in the next section.

5. The Interaction Between the Snoop Module and Routing Protocol

The snoop module relies on a cache of unacknowledged packets to improve end-to-end TCP performance. Typically, the size of this cache is proportional to the TCP window size. After a handoff, the new base station must have the current set of unacknowledged packets in its cache to provide improved performance based on local retransmissions.

When a handoff is requested by the mobile host or anticipated by the base station, the nearby base stations join a multicast group to update the routing. They receive all packets destined for the MH and the snoop modules here attempt to mirror the state present in the primary base station for the different TCP connections. Using these packets, the snoop module builds up its cache for various connections to the mobile host. However, during this period packets and acknowledgments *from* the MH continue to pass only through the primary BS. Therefore, the nearby buffering base stations cannot snoop on any acknowledgments for the caches they are mirroring. This prevents the snoop module from freeing up packets that have safely reached the MH. As a result, packets are only freed when snoop runs out of buffer space using a FIFO scheme. In normal operation, the buffering BSs have a superset of the packets contained at the forwarding BS.

Once the handoff occurs, acknowledgments begin to pass through the new forwarding BS. The first acknowledgment the BS receives identifies which packets have been received by the MH on a connection. The BS uses this information to clean and synchronize the contents of the snoop cache to the last state of the previous primary BS.

In reality, the state of the new and previous BSs are not likely to be identical after the synchronization. The new primary BS may be missing several packets from its snoop cache. This is because it may have either missed several packets while it was in buffering mode (due to congestion) or buffered packets for only a short period of time before handoff. Since the snoop module does not change any of the end-to-end semantics of TCP, it is resistant to these gaps in its state. As the connection progresses the holes in the cache will either be filled or ignored. In the worst case, these holes will result in a slight performance degradation for a short period after handoff. Our experience has been that it takes only a few packets before the snoop module at the new base station reaches the current state.

The duration of a handoff is short since no state is explicitly transferred between BSs. The state transfer is avoided because the new primary BS has attempted to mirror the state of the previous primary BS prior to handoff. This handoff scheme attains low-latency for TCP streams since multicast provides a simple, lightweight way to mirror state changes at the BSs and because the snoop module is resistant to gaps in its state.

6. Implementation

We have implemented the snoop module and routing 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

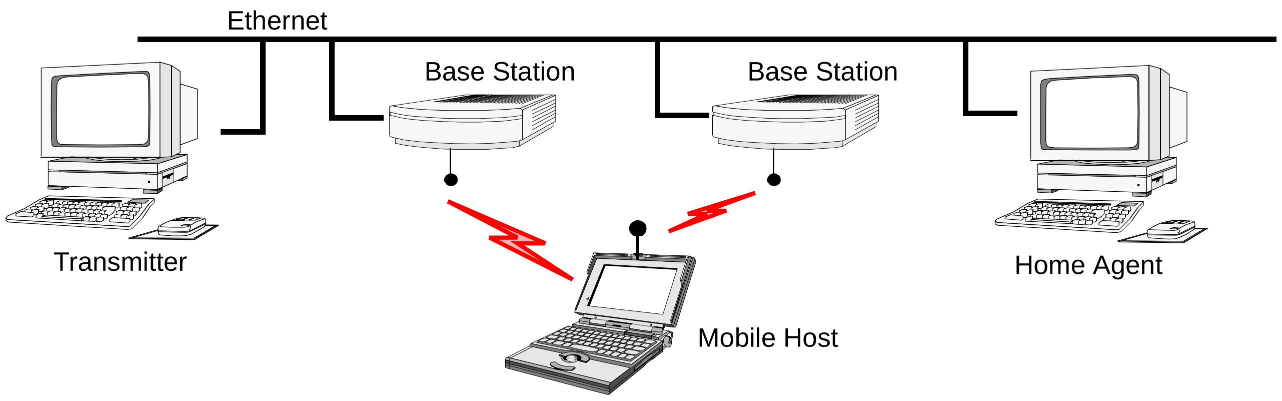

Figure 6. Network topology for experiments.

aggregate bandwidth of the Wavelan is about 2 Mbits/s. The implementation currently supports bulk transfers to and from mobile hosts and supports smooth handoffs. The network topology for our experiments is shown in Figure 6.

The state maintained by snoop is easily 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 [18] 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 at a forwarding (primary) base station needs to be large enough to handle the maximum transmission window size. In practice, we set a "highwater 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 [7, 16]. We use the reference counting mechanism present in kernel mbufs to avoid data copying in the snoop module. 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 module also performs retransmissions driven by timeouts. There are two types of timer interrupts in the module, 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 do not timeout more frequently than a threshold time, currently set to 40 ms. 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 200 ms. 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 mobile host.

As mentioned in the design of the routing protocol, there are three major components of the mobile routing system -- the packet encapsulation at the home agent, the decapsulation at the base stations and the beaconing system. The encapsulation and decapsulation modules uses data structures similar to those used by the snoop module. In-kernel data structures are used to maintain mappings between IP multicast addresses and home addresses. These tables are configured by user-level daemons. The home agent uses this table to perform the encapsulation in the IP forwarding code. This same table is used in the forwarding and buffering BSs to identify which multicast packets are destined to which mobile hosts.

Each of the buffering base stations store the packets transmitted to a mobile host in a circular buffer. The maximum number of packets to store is set to prevent data loss during handoff. In our current implementation, we store at most 12 packets, which corresponds to between 24 and 72 ms of transfer time across the Wavelan link. During a handoff, the mobile host transmits to the new primary base station unique identifiers (IP IDs) of the last three packets it has received. The base station searches for these IDs in its circular buffer and frees all packets that arrived before them. The remaining TCP packets in the circular buffer are transmitted to the mobile host. If the IP IDs are not found, the entire circular buffer is sent.

The beaconing system was implemented in three parts -- a user level process at the base stations, a user level process at the mobile host and a kernel module at the mobile host. The process on the base station transmits a broadcast packet on the wireless network once per second. The beacon message contains the IP address of the base station and a timestamp. Each mobile host has a user-level beacon analysis process that listens for new beacons on the wireless network. When the process receives a beacon from a base station it has not heard before, it requests the kernel to add the base station to the list of active beacon sources. When a beacon from a listed base station arrives a the mobile host, the kernel samples the signal strength of the wireless transmitter. The beacon analyzer process reads the signal strength samples and uses them to determine when handoff should occur.

7. Performance

We performed several experiments with the snoop module and routing 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 6. The sender TCP stack was based on TCP Reno, an implementation supporting 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. We also experimented with using a two-state Markov error generator that more accurately modeled the wireless channel. The two states corresponded to periods of good connectivity and periods of poor connectivity. Poisson-distributed errors were generated at different rates in each state. Unfortunately, throughput measurements converged very slowly when using this error model and it was difficult to interpret the implications of the results. As a result, we used the Poisson error model across a wide range of bit-error rates to understand how the snoop module would perform in either the good or bad channel state.

Each run involved a 10 MByte transfer and this was repeated ten times at each error rate. Figure 7 compares the throughput of a connection using the snoop module with that of a connection using an unmodified TCP implemen-

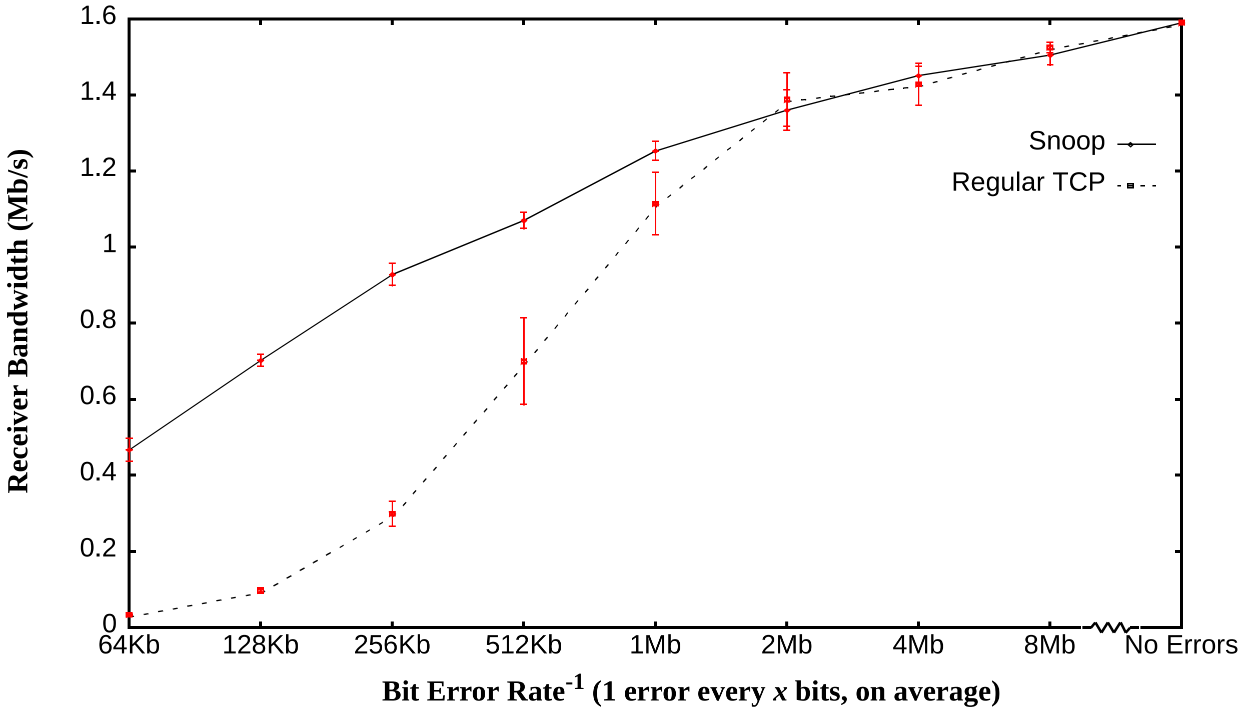

Figure 7. Throughput received by the mobile host at different bit-error rates ($\log_2$ scale). The vertical error bars show the standard deviations of the receiver throughput.

tation, 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 $5x10^{-7}$ (close to the 2 Mb point on the x-axis of the graph) the snoop protocol performs significantly better than unmodified TCP, achieving a throughput 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, whereas the regular TCP connection didn't make any progress. Under conditions of very low bit error rates ($< 5x10^{-7}$), 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 8, which plots the sequence numbers of the received TCP packets versus time for one of the experiments. These values were obtained using the tcpdump [19] network monitoring tool. The figure shows the comparison of sequence number progression 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 Mbits/s for regular TCP.

To isolate and measure the impact of handoff on TCP performance, we examined the performance of a TCP connection to the mobile host with regularly spaced handoffs between two base stations. As mentioned in Section 6, each base station sends out a beacon signal once per second. The presence of these beacons reduces the peak TCP throughput in the absence of errors and handoffs to 1.45 Mbits/s. In this experiment, the arrival of a beacon at the mobile host does not trigger an analysis of the signal strengths of the different base stations; instead, the mobile host uses the time elapsed since the last handoff to determine if a handoff should occur. In order to stress the perfor-

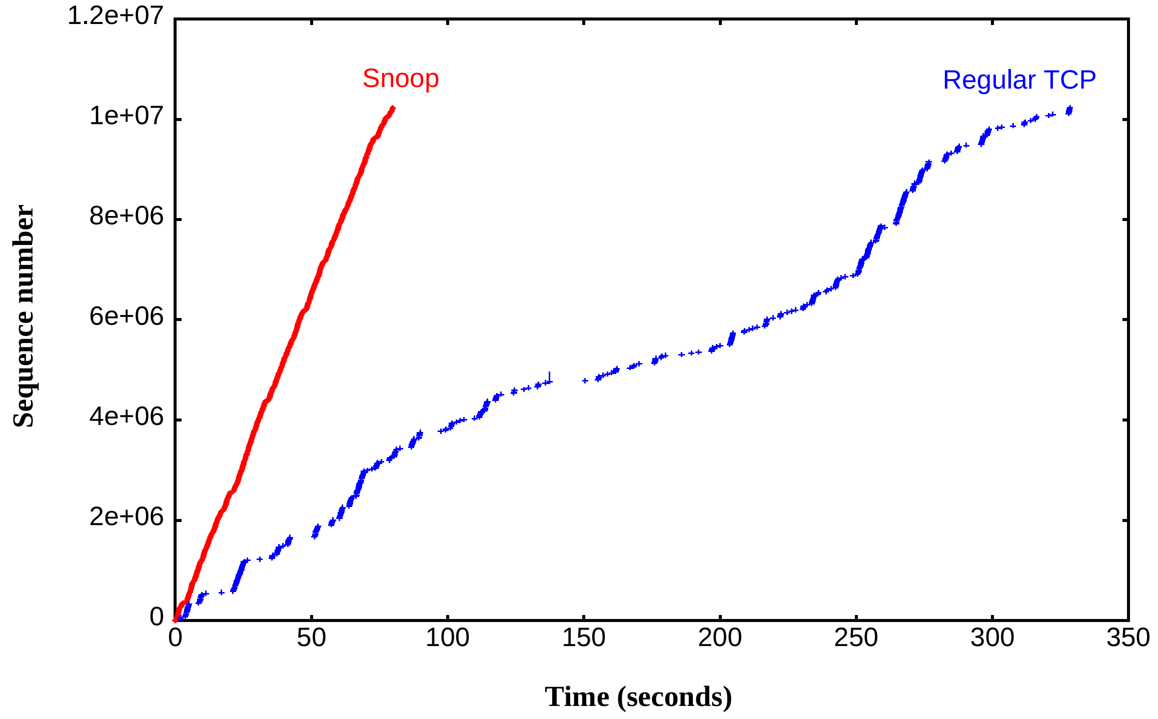

Figure 8. Sequence numbers for transfer to mobile host over channel with $3.8x10^{-6}$ (1/256 Kbits) BER.

mance impact of the handoff scheme on end-to-end performance, we performed several tests, varying the time between handoffs from 1 to 10 seconds. The results for different handoff rates is shown in Table 1. These measure-

Time Between Handoffs (sec)	Throughput (Mbits/s)	Standard Deviation (Mbit/s)
1	1.42	.011
2	1.43	.016
3	1.43	.012
5	1.43	.014
8	1.44	.012
10	1.43	.012
∞	1.45	.011

TABLE 1. Throughput received by the mobile host at different handoff frequencies.

ments show that even frequent handoffs have very little impact on performance. In a real environment, handoffs are likely to occur much less frequently than once per second. The behavior of a connection experiencing handoff is shown in Figure 9. The figure plots the sequence numbers of a TCP connection to a mobile host with handoffs occurring every 10 seconds. We see that the data transfer progresses without any significant interruptions despite the presence of the handoffs. The throughput during this transfer was about 1.4 Mbits/s.

Figure 10 shows the time chart of various events that occur during a typical handoff. The horizontal axis shows the observed range of times between events. The timings were obtained from analyzing tcpdump traces of handoffs during a run. The variation in delays are mainly due to network queueing at the MH and BS and delays in the MAC layer of the Wavelan network. The run analyzed consisted of 10 handoffs occurring before, during and after a 10 Mbyte transfer. Handoffs occurred every 10 seconds during the 100 second long run. During idle periods, handoffs

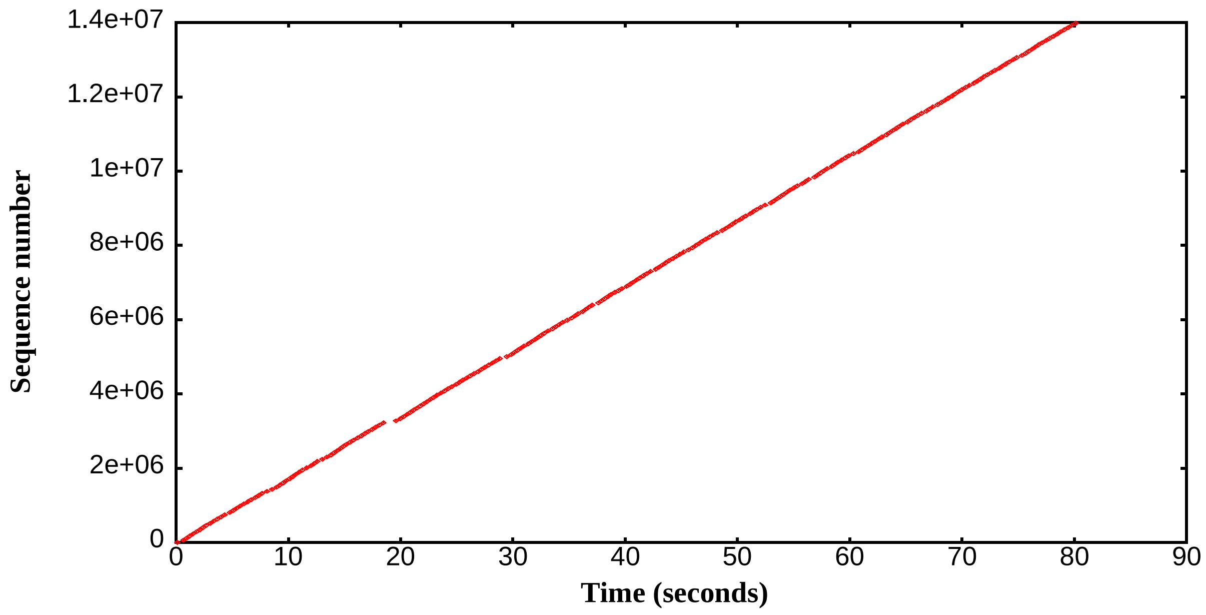

Figure 9. Sequence numbers for transfer to mobile host over channel with handoffs every 10 seconds.

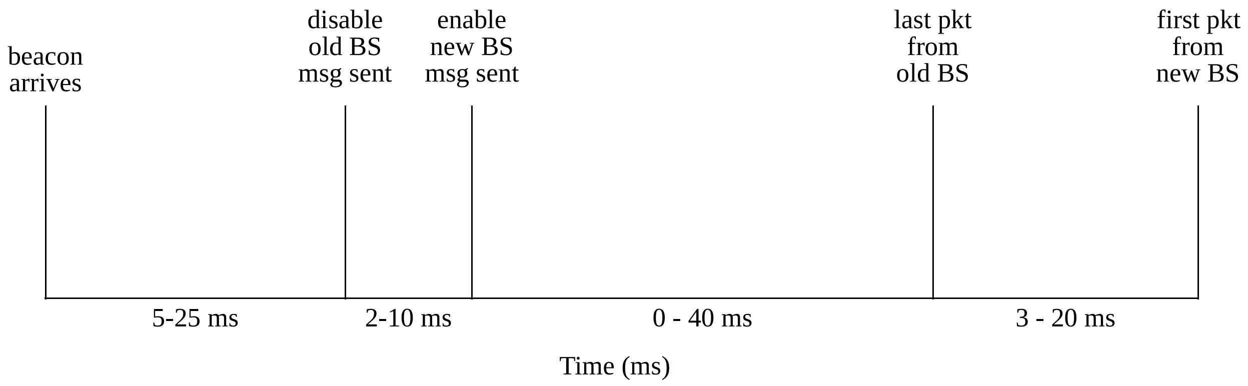

Figure 10. Timing of handoff events

complete on the low range of the times listed, with handoff latencies on the order of 5 to 10 ms. The handoff latency grows to between 25 and 70 ms during the TCP transfer.

Figure 11 shows the performance obtained from the combination of the snoop module and routing protocol. The figure plots the receiver throughput seen over a 10 Mbyte transfer. The connections experienced different error rates, shown on the x-axis, and handoffs at a fixed rate, one every 5 seconds. At low bit-error rates, < $1x10^{-6}$ (one error every 1 Mbits of data), we obtain close to the peak performance available in the presence of beacons, 1.45 Mbit/s. At higher bit-error rates, we see performance comparable to transfers through snoop to a mobile host not experiencing handoff. This indicates that the handoffs do not adversely affect the snoop processing and that the snoop module state was effectively mirrored at the buffering base stations.

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. The routing protocol allows user mobility without data loss or highly variable delays. Handoffs complete in 5-70 ms and this results in a less than 1% degradation in snoop throughput.

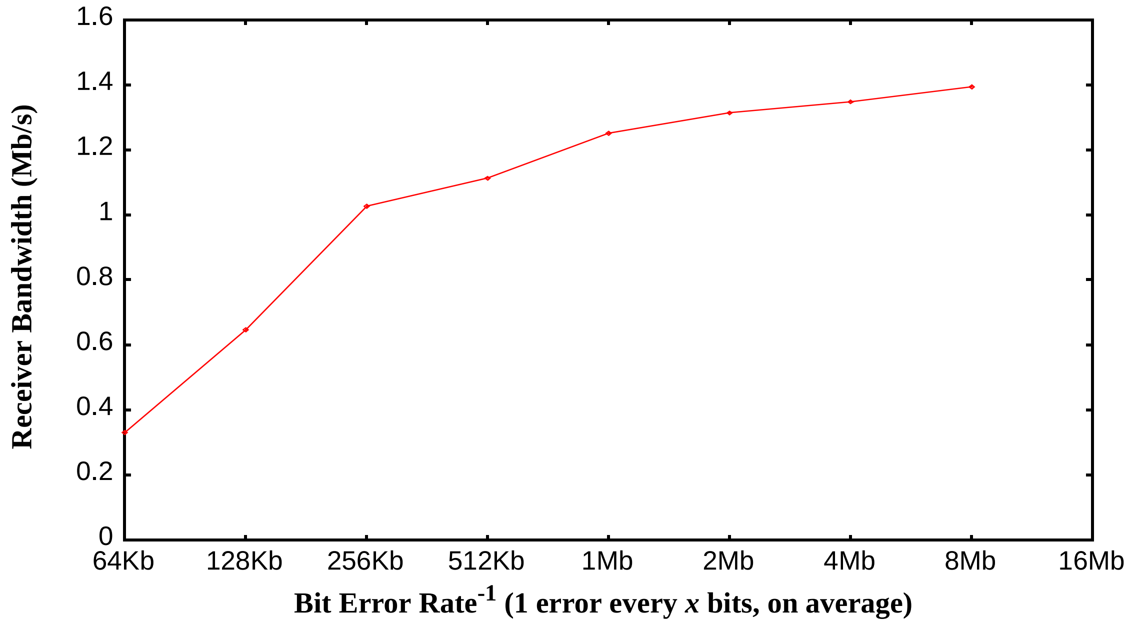

Figure 11. Throughput received by the mobile host at different bit-error rates ($\log_2$ scale). Handoffs occurring every 5 seconds

8. Comparisons with Other Implementations

In this section, we compare the snoop and routing protocols 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.

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 [4]. In contrast, handoffs in our system are based on multicast as described in Section 4 and typical completion times are between 5 and 70 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. Also, the retransmissions begin from the last packet the mobile host has received and the set of packets in flight during the handoff will be retransmitted from the source. 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 our 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.

9. 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 and data transfers from a mobile host. We are also working on characterizing the behavior of TCP connections and the snoop module 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 [15].

10. Summary

We have presented a set of additions and modifications to the standard Internet protocol stack 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 mobile host, and involves no other changes to any of the fixed hosts elsewhere in the network. The two main ideas of the new protocol address the problems of high bit-error rates on wireless links and data loss caused by handoffs. Our solution to the problem of bit-errors is to cache packets intended for the mobile host at the base station and perform local retransmissions across the wireless link. We eliminate losses caused by mobility by using a low latency, multicast-based handoff algorithm. We have implemented the new protocol stack 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 this protocol stack is significantly more robust than regular TCP in the presence of unreliable links, multiple errors in a window and user mobility. 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, reduced handoff latency to between 5 and 70 ms and eliminated data loss during handoff.

Acknowledgments

Elan Amir actively participated in the initial design and simulations of the snoop protocol. We are grateful to him for his contributions. We thank Kimberly Keeton, Bruce Mah, and Steven McCanne for several comments and suggestions on earlier drafts 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.

References

[1] A. S. Acampora and M. Naghshineh, An architecture and methodology for mobile-executed handoff in cellular ATM, *IEEE Journal on Selected Areas in Communications*, 12(8) (October 1994) 1365–1375.

[2] E. Amir, H. Balakrishnan, S. Seshan, and R. H. Katz, Efficient TCP over networks with wireless links, in *Proc. Fifth IEEE Workshop of Hot Topics in Operating Systems* (May 1995).

[3] A. Bakre and B. R. Badrinath, I-TCP: Indirect TCP for mobile hosts, Technical Report DCS-TR-314, Rutgers University (October 1994).

[4] 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).

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

[6] R. Caceres and L. Iftode, Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments, *IEEE Journal on Selected Areas in Communications*, 13(5) (June 1994) 850-857.

[7] D. C. Clark, V. Jacobson, J. Romkey, and H. Salwen, An Analysis of TCP Processing Overhead, *IEEE Communication Magazine* (June 1989) 23-29.

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

[9] A. DeSimone, M. C. Chuah, and O. C. Yue, Throughput Performance of Transport-Layer Protocols over Wireless LANs, in *Proc. Globecom ’93* (December 1993).

[10] R. Ghai and S. Singh, An Architecture and Communications Protocol for Picocellular Networks, *IEEE Personal Communications Magazine*, 1(3) (1994) 36–46.

[11] V. Jacobson, Congestion avoidance and control, in Proc. *SIGCOMM 88* (August 1988).

[12] V. Jacobson and R. T. Braden, *TCP Extensions for Long Delay Paths*, RFC-1072 (October 1988).

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

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

[15] R. H. Katz, Adaptation and mobility in wireless information systems, *IEEE Personal Communications*, 1(1) (1994).

[16] J. Kay and J. Pasquale, The importance of non-data touching processing overheads in TCP/IP, in *Proc. SIGCOMM ’93* (September 1993).

[17] K. Keeton, B.A. Mah, S. Seshan, R.H. Katz, and D. Ferrari, Providing connection-oriented service to mobile hosts, in *Proc. First USENIX Symp. on Mobile and Location-Independent Computing* (August 1993).

[18] 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, November 1989).

[19] S. McCanne and V. Jacobson, The BSD packet filter: A new architecture for user-level packet capture, in *Proc. Winter '93 USENIX Conference* (January 1993).

[20] 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).

[21] C. Perkins, IP mobility support, IETF Mobile-IP Draft (1995).

[22] J. B. Postel, *Transmission Control Protocol*. RFC 1793 (September 1981).

[23] W. R. Stevens, *TCP/IP Illustrated, Volume 1* (Addison-Wesley, Reading, MA, November 1994).

[24] R. Yavatkar and N. Bhagwat, Improving end-to-end performance of TCP over mobile internetworks, in *Workshop on Mobile Computing Systems and Applications* (December 1994).

Hari Balakrishnan (ACM S 1995, IEEE S 95) is a Ph.D. candidate in Computer Science at the University of California at Berkeley. His research interests are in the areas of computer networks, mobile computing, and distributed computing and communication systems. He received a B. Tech. degree in Computer Science and Engineering from the Indian Institute of Technology (Madras) in 1993.

On the WWW, his URL is: http://www.cs.berkeley.edu/~hari. His e-mail address is: hari@cs.berkeley.edu.

Srinivasan Seshan (ACM S 1993, IEEE S 93) is a Ph.D. candidate in Computer Science at the University of California at Berkeley. His main area of research interest is computer networks especially network protocols to support mobile computing. He is also interested in other areas of communication systems, mobile computing and distributed computing. He received a B.S. in Electrical Engineering and M.S. in Computer Science from the University of California at Berkeley in 1990 and 1993 respectively. He is a member of member of the Eta Kappa Nu and Tau Beta Pi honor societies.
E-mail: ss@CS.Berkeley.EDU
WWW URL: http://www.CS.Berkeley.EDU/~ss

Professor Randy H. Katz is a leading researcher in computer system design and implementation. His research experience has spanned numerous disciplines. He has written over 120 technical publications on CAD, database management, multiprocessor architectures, high performance storage systems, and video server architectures. He was responsible for developing the concept of Redundant Arrays of Inexpensive Disks (RAID), now a $3 billion industry segment. Katz's recent research has focused on wireless communications and mobile computing applications.

From January 1993 through December 1994, Katz was a program manager and deputy director of the Computing Systems Technology Office of ARPA. He was responsible for “wiring the White House” to the Internet, and also assisted the Clinton Administration in formulating policies related to the National Information Infrastructure and wireless technologies.

He is a member of the ACM and a senior member of the IEEE.