M-TCP: TCP for Mobile Cellular Networks*

Kevin Brown and Suresh Singh
Department of Computer Science
University of South Carolina
Columbia, SC 29205
{*kbrown,singh*}*@cs.sc.edu*
Tel: 803-777-2596

July 29, 1997

Abstract

Transport connections set up over wireless links are frequently plagued by problems such as – high bit error rate (BER), frequent disconnections of the mobile user, and low wireless bandwidth that may change dynamically. In this paper, we study the effects of frequent disconnections and low variable bandwidth on TCP throughput and propose a protocol that addresses this problem. We discuss the implementation (in NetBSD) of our protocol called M-TCP and compare its performance against other mobile TCP implementations. We show that M-TCP has two significant advantages over other solutions: (1) it maintains *end-to-end* TCP semantics and, (2) it delivers excellent performance for environments where the mobile encounters periods of disconnection.

1 Introduction

Mobile computing is an emerging new computing paradigm that will allow users to access electronic data and services "anywhere, anytime". Thus, users away from home will be able to transparently access their data as if they had never left. We can identify at least two types of *mobile users* based on observed patterns of user behavior – one type of user model is where the user travels away from home and accesses her data from a remote *stationary* location (say a hotel room), and the second model is one where the user accesses her data while *on the move* (e.g., while in a car). In this paper we will restrict ourselves to the latter type of mobile data access only.

A *cellular network* infrastructure is typically used to connect mobile users to the Internet. A geographical region, such as a highway or campus, is divided into *cells* each of which contains a *base station* that provides a connection end-point for roaming mobiles. The base stations are connected to the wired infrastructure to provide access to the Internet. A cell may be as small as tens of meters in diameter (in the case of in-building picocellular networks [15]) or as large as a mile in diameter (macrocellular network). Smaller cells are useful if we need to provide higher bandwidth to the users (this is possible since the number of users per cell is smaller) but results in smaller *cell latencies* (i.e., time a mobile spends in a cell) that, in turn, cause frequent *handoffs* (when the mobile roams into a new cell, the new base station takes over the management of all connections – this is called a handoff).

In order to maintain connections for mobile users, it is necessary to keep track of their *location*, ensure that the network can *route* packets to the user's current location and, depending on the type of connection, ensure some Quality of Service (e.g., best-effort UDP, reliable data transfer, etc.). Many solutions have been proposed for all of these problems. For instance, [13, 9, 20] discuss several aspects of routing in a mobile environment and compare different solutions including Mobile-IP. Other authors have addressed the problem of location management, see for instance [8]. A solution for improving the performance of UDP over wireless links was proposed by us, [11]. In this paper, however, we are only interested in studying the behavior of the TCP protocol over wireless links and we will therefore only provide a comprehensive literature review of research relevant to this problem.

Many papers have been written proposing methods for improving TCP performance over a wireless link, [3, 21, 6, 7, 12]. All these papers have, however, concentrated on only one problem associated with wireless links – a perceived high bit-error rate (BER) over the wireless link. While a high BER has significant implications for protocol performance, other limitations of the wireless environment are equally or more important than high BER. In this

*This work was supported by the NSF under grant number NCR-9410357 and by a equipment grant from NCR Corporation.

paper we study the effects of *periodic disconnections* on TCP performance and develop a protocol (M-TCP) that successfully deals with this problem. The protocol is also capable of on-the-fly data compression in the event that the wireless bandwidth available is very small. Finally, unlike many other solutions, our protocol maintains *end-to-end* TCP semantics.

1.1 The Problem with TCP

TCP is a connection-oriented transport layer protocol that provides reliable, in-order delivery of data to the TCP receiver. Since mobile hosts will expect the same services that are offered to fixed hosts, it is necessary to implement TCP for the mobile domain. If we use TCP without any modification in mobile networks, we experience a serious drop in the throughput of the connection. There are several reasons for such a drastic drop in TCP throughput and in this section we examine these reasons in some detail.

The Effect of a High Bit Error Rate (BER)

It has been suggested that the wireless link suffers from a high bit error rate. While this may be true in some cases, recent studies [14] indicate that the bit error may be no worse then 10^{-5} which can be reduced by one or two orders of magnitude by the use of appropriate FEC (Forward Error Correcting) codes and retransmission schemes at the link layer, see [2]. However, let us assume for the sake of this discussion that the wireless link *is* susceptible to inordinately high bit error rates.

Bit errors cause packets to get corrupted which result in lost TCP data segments or acknowledgements. When acknowledgements do not arrive at the TCP sender within a short amount of time, (the retransmit timeout or RTO which is a multiple of half a second), the sender retransmits the segment, exponentially backs off its retransmit timer for the next retransmission, and *closes its congestion window to one segment*. Repeated errors will ensure that the congestion window at the sender remains small resulting in low throughput (see [5]). It is important to note that FEC may be used to combat high BER, but it will waste valuable wireless bandwidth when correction is not necessary.

The Effect of Disconnections

In a mobile environment, as a user moves between cells, there is a brief blackout period (or disconnection) while the mobile performs a handoff with the new MSS. Disconnections may also be caused by physical obstacles in the environment that block radio signals, such as buildings. If a cell contains many users, some connections (of newly arriving mobiles) may not receive any bandwidth for large time periods (call blocking) – this can also be considered a disconnection event. Disconnection periods can be of the order of several seconds causing packet loss or delay in the transmission of acknowledgements of received packets. In the case of Metricom's Ricochet network [1], for instance, a disconnection may last as long as 1 minute! These disconnections result in lost data segments and lost ACKs which, in turn, result in the TCP sender timing out and closing its congestion window, thus greatly reducing the efficiency of the connection. Since these disconnections tend to be fairly lengthy, forward error correction schemes are ineffective.

The Effect of Frequent Disconnections

It is likely that in order to provide high-bandwidth wireless connections, cell sizes will have to be reduced (one example where cell sizes are small is the Ricochet network developed by Metricom, Inc. – a typical cell covers a one or two block area). Small cell sizes unfortunately result in small *cell latencies* that, in turn, cause frequent disconnections as a user roams. All the problems that result from disconnections, as we discussed above, occur more often here.

Another problem caused by small cell latencies and frequent disconnections is that of *serial* timeouts at the TCP sender. A serial timeout is a condition wherein multiple consecutive retransmissions of the same segment are transmitted to the mobile while it is disconnected. All these retransmissions are thus lost (see Figure 1). Since the retransmission timer at the sender is doubled with each unsuccessful retransmission attempt (until it reaches 64 sec), several consecutive failures can lead to inactivity lasting several *minutes*. Thus, even when the mobile is reconnected no data is successfully transmitted for as long as 1 minute! We have found that serial timeouts at the TCP sender can prove to be even more harmful to overall throughput than losses due to bit errors or small congestion windows.

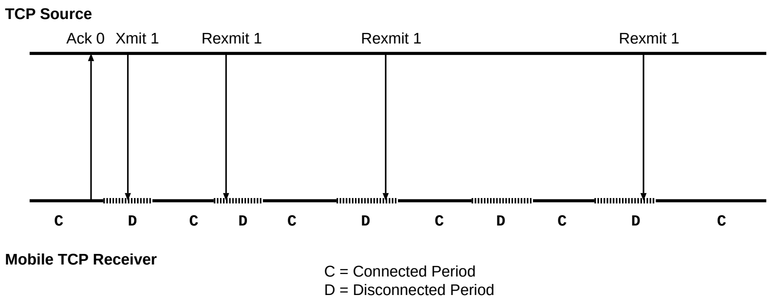

Figure 1: Serial timeouts at the TCP sender.

1.2 Overview of the Paper

Our M-TCP protocol was designed to work well in the presence of frequent disconnection events and over low bit-rate wireless links subject to dynamically changing bandwidth. In addition, it maintains end-to-end TCP semantics. In the following section we discuss the related work on this problem. In section 3 we present our network architecture in the context of which we developed M-TCP. Section 4 presents the design of our M-TCP protocol and section 5 discusses the implementation details. Performance results are presented in section 6 and section 7 concludes the paper.

2 Approaches for Improving TCP Throughput in Mobile Environments

In order to ensure that the TCP connection to a mobile is efficient, it is necessary to prevent the sender from shrinking its congestion window when packets are lost either due to bit-error or due to disconnection. Furthermore, it is important to ensure that, when the mobile is reconnected, it begins receiving data immediately (rather than having to wait for the sender to timeout and retransmit). As we saw from our earlier discussion, the sender may shrink its congestion window in response to a timeout (caused by lost packets – either due to a high BER or due to disconnections) or in response to duplicate ACKs. Most of the solutions we discuss below attempt to keep the sender's congestion window open by introducing a host in the fixed network who 'spoofs' the sender into thinking everything is fine on the wireless link. Unfortunately, however, these solutions do not work under all scenarios of mobility. Specifically, they all perform poorly when faced with frequent or lengthy disconnections. Furthermore, some solutions fail to maintain end-to-end TCP semantics.

The remainder of this section is divided into three parts. In section 2.1 we present the existing solutions to the problem. In section 2.2 we discuss the relative performance of these solutions and outline their drawbacks. Finally, other related work is presented in section 2.3.

2.1 Existing Solutions

One proposed solution for losses caused by high BER is the Berkeley Snoop Module [6]. The snoop module resides at an intermediate host near the mobile user (typically the base station). It inspects the TCP header of TCP data packets and acknowledgements which pass through and buffers copies of the data packets. Using the information from the headers, the snoop module detects lost packets (a packet is assumed to be lost when duplicate acknowledgements are received) and performs local retransmissions to the mobile. The module also implements its own retransmission timer, similar to the TCP retransmission timeout, and performs retransmissions when an acknowledgement is not received within this interval. An improved version of the snoop module [7] adds selective retransmissions from the intermediate node to the mobile.

Another solution to the problem caused by high BER is the I-TCP [3] protocol (Indirect-TCP). In the I-TCP protocol a TCP connection between a fixed host and a Mobile Host (MH) is **split** in two at the Mobile Support Station (MSS) or base station. Data sent to the MH is received and ACK'ed by the MSS before being delivered to the MH. Note that on the connection between the MSS and MH it is not necessary to use TCP, rather some protocol optimized for the wireless link could be used. The MTCP protocol of [21] is very similar to I-TCP. It also splits a TCP connection in two – one from the MH to MSS and another from the MSS to the FH (i.e., the Fixed Host or sender). The MH to MSS connection passes through MHP, a session layer protocol. Two implementations for

MHP are proposed. In the first, MHP uses TCP over the wireless link to the MSS. In the second implementation, a selective repeat protocol, SRP, is used over the wireless link. SRP lies below the socket layer but above TCP.

Both I-TCP and MTCP achieve better throughputs than standard TCP for the following reason: the node where the connection is split is, hopefully, one or two hops away from the MH's cell and can adapt more quickly to the dynamic mobile environment because the round trip time (RTT) is very small. This means that even if the MSS sender (the node where the connection is split) reduces its congestion window to one because of lost acknowledgements from the mobile, the window will quickly build up to the maximum again when new ACKs arrive from the mobile because of the short RTT. MTCP/SRP also increases throughput by using selective repeat which reduces duplicate sends.

[12] presents a solution that addresses the problem caused by short disconnections (where one or two segments only are lost). It, however, does not split the TCP connection. The solution is based on the following observation: during a handoff, since the MH cannot receive packets, unmodified TCP at the sender will think a congestion has occurred and will begin congestion control (reduce window size and retransmit) after a timeout. The timeout period is long and even though the MH may have completed the handoff it will have to wait for the full timeout period before it begins receiving packets from the sender. The fast retransmit idea presented in [12] forces the MH to retransmit, in triplicate, the last old ACK as soon as it finishes a handoff. This forces the sender to reduce the congestion window to a half and retransmit one segment immediately.

2.2 Drawbacks and Strengths of Existing Solutions

In this section we examine how each of the existing solutions discussed above fare when confronted with high BER, different types of disconnections, and their ability to maintain TCP semantics.

Let us first consider the two *split* connection protocols, I-TCP and MTCP (note that [7] discusses these and other minor variants of the split connection idea as well). It is easy to see that the sender is shielded from any knowledge of the wireless link including disconnections and high BER because the MSS ACKs all data even before it is ACK'ed by the MH. However, as [7] showed, it is necessary to implement SACK (Selective ACK) over the MSS–MH part of the connection in order to recover quickly in high bit-error environments.

If the mobile environment is subject to frequent disconnections, it is easy to see that *serial timeouts* may occur at the base station resulting in long idle periods. We have found that the standard split connection approach, which uses TCP on both halves of the connection, does, in fact, respond poorly to lengthy disconnections. In experiments using a cell latency of 5 seconds, and fade intervals ranging from 0.5 to 4.5 seconds, we have observed serial timeouts that sometimes cause periods of inactivity lasting as long as 5 minutes or more. Thus, a split connection using TCP on both sides only localizes the problem, it does not correct it.

It is also important to observe that, since the MSS ACK's segments to the sender before delivering them to the MH, it can easily run out of buffer space (imagine several users browsing the web simultaneously and downloading GIFs) – *this happens because there is no linkage between the wireless throughput and the receive buffer size at the MSS*. There is thus a need to develop smart buffer management strategies at the base station. Another problem arising from the use of large buffer space is an increased handoff latency when the MH roams from one cell into another. When the mobile moves from one cell to another, the old MSS will need to handoff all the I-TCP state. This state can be fairly large and will cause a significant handoff delay (the MH cannot receive data until the state has been handed off).

Finally, it is clear that both I-TCP and MTCP *do not* preserve end-to-end TCP semantics. The sender may believe a segment is delivered correctly to the MH since the MSS ACK'ed it even if the MH disconnected before receiving this segment. This is clearly a serious problem for many applications.

The snoop module [6, 7] was designed to combat high BER environments by providing local retransmissions of data and local filtering of ACKs to prevent the sender from invoking congestion control. The snoop module implements a round-trip timer, much like TCP and does retransmissions of unacknowledged segments based on this timer. In addition snoop maintains a persist timer that goes off once every 200 ms, when there is unacknowledged data at the snoop module, and there has been no activity from either the sender or the receiver in that time (where the retransmissions above do not count as activity).

It is easy to see that the snoop module (in particular the version that incorporates SACK [7]) performs extremely well in high BER environments because the RTO estimate between the MSS and MH is small resulting in quick retransmissions of lost data. In addition, the persist timer ensures that if there has been a bursty loss (2-6 packets, see [7]), the snoop module will retransmit some data within 200 msec (this interval is smaller than the minimum granularity of TCP timers which is 500 msec) thus ensuring that the sender does not invoke congestion control due

	Fast Retransmit	Snoop	I-TCP & MTCP	Our M-TCP
End-to-end TCP semantics	$\surd$	$\surd$		$\surd$
Handle High BER		$\surd$	$\surd$	$\surd$
Long Disconnections			May run out of buffers	$\surd$
Frequent Disconnections			High handoff cost	$\surd$

Table 1: Comparison of different mobile TCP solutions.

to such a loss. It is also noteworthy that snoop *maintains* end-to-end TCP semantics because it simply forwards ACKs generated by the MH.

Unfortunately, snoop does not perform as well either in the presence of lengthy disconnections or in environments where there are frequent disconnections. If the mobile is disconnected for a lengthy period of time, the sender will automatically invoke congestion control because it will not have received ACKs for some segments. The snoop module will generate persist packets every 200 msec but, since the mobile is disconnected, these packets will serve no purpose.

A related problem with snoop is its behavior in the presence of frequent handoffs. When a MH moves into a new cell, the new MSS starts up a copy of the snoop module on behalf of this MH. This snoop module starts out with an empty cache and slowly builds the cache up. Thus, while the snoop module is building up its cache, the MH will see poor TCP throughput since snoop cannot yet perform any useful retransmission or filtering functions. If we assume that cell sizes are small (as they will be in picocellular and microcellular environments), we see that the performance degradation can be serious. A solution proposed to this problem is to use 'soft handoffs' where the MH continues receiving data from the old MSS while the new MSS builds up its cache. A problem with this solution is the assumption that the mobile will move into a specific new cell when it moves out of its current cell. It is quite likely that the mobile may move into one of many adjacent cells. Thus, in order for the above solution to work, the MSSs in *all* adjacent cells will need to begin running the snoop module *in anticipation* of the MH wandering in. In effect, each MSS runs a snoop module for MHs located in its own cell as well as in all adjacent cells! The processing and buffering overhead is clearly too high for this solution to be practical.

The fast retransmit solution [12] was designed specifically to combat the effects of short disconnections on TCP throughput. If we assume that one or two packets were lost due to a disconnection and if the RTO at the sender is large, it is clear that this solution works very well since the sender is forced to retransmit one packet as soon as the MH finishes handoff. Unfortunately, if the MH was disconnected for a long time, the sender would already have invoked congestion control and shrunk its window to one segment. Thus when the MH sends three duplicate ACKs, the sender will only retransmit the one segment in its window. The congestion window will have to grow slowly – in effect the behavior of this protocol is no different from unmodified TCP. Similarly, if disconnections are frequent or if the wireless environment is lossy, this scheme will do little to improve throughput because the sender's congestion window will repeatedly get shrunk to half its previous size.

We summarize the performance of all these protocol in Table 1. The last column refers to the M-TCP protocol presented in this paper and we will justify its columnar entries later in the paper.

2.3 Other Solutions

Most other solutions that have been proposed to improve wireless throughput focus on the link layer. Some schemes attempt to alleviate the problem of losses over the wireless link by using FEC codes or by employing ARQ (automatic repeat request) messages [17] – the MH requests retransmissions of specific messages that were corrupted. The AIRMAIL [2] protocol is one good example of an approach that uses both of these techniques. The drawback to using link layer solutions is that they *cannot account for disconnections*. Thus, even though link layer solutions succeed in reducing the observed BER, they can do little to prevent TCP from timing out when an ACK does not arrive on time (because the MH was disconnected). We believe that link layer solutions, in combination with our own M-TCP protocol, provide the best solution for improving TCP throughput – the link layer solutions reduce the loss rate due to bit-error while our protocol ensures that the sender does not react negatively to disconnections and low (or variable) wireless bandwidth.

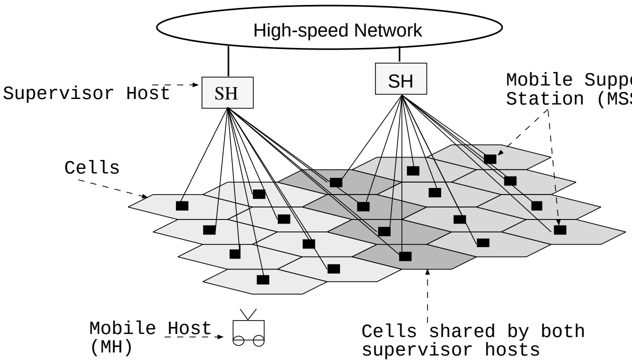

Figure 2: Proposed Architecture.

3 Underlying Architecture

The implementation of M-TCP was influenced, to some degree, by the mobile network architecture we have been developing at the University of South Carolina. The detailed architecture is presented in [10] and in this section we briefly summarize the main points. Our architecture may be viewed as a three-level hierarchy (see Figure 2). At the lowest level are the mobile hosts (MH) who communicate with MSS nodes in each cell. Several MSSs are controlled by a machine called the Supervisor Host (SH). The SH is connected to the wired network and it handles most of the routing and other protocol details for the mobile users. In addition it maintains connections for mobile users, handles flow-control and is responsible for maintaining the negotiated quality of service. These SHs thus serve the function of *gateways*. Why did we opt for a 3-layer architecture? Some of the reasons are:

1. The MSSs are simple cheap devices that only serve as connection end-points. The I-TCP, MTCP and snoop protocols, on the other hand, require a sophisticated MSS – this increases the cost of the cellular infrastructure particularly if we have small cell sizes as is the case for in-building picocellular networks.

2. When a MH roams from one cell into another, the two MSSs need not perform any state transfer if the two cells are controlled by the same SH. Observe that, if a SH controls a region containing n cells, there may only be $O(\sqrt{n})$ significant handoffs per unit time (i.e., when a mobile leaves a cell controlled by one SH and enters one controlled by another SH – the old SH will have to transfer the state of all the mobile's connections). Contrast this with the case where the MSSs manage all transport connections – here there will be $O(n)$ state handoffs per unit time (see [18]).

3. Since several MSSs are controlled by a single SH, the roaming MH remains within the domain of the same SH for long time periods[1]. This makes it easier to manage the MH's connections and ensures that any MH state maintained at the SH need only be moved infrequently.

One drawback of this architecture is the complexity of the SH. In particular, if one SH manages several cells, we need to ensure that it can handle all connections that may originate from mobiles located in these cells. In section 6 we examine this issue in some more detail.

3.1 Transport Layer Design

The design of our transport layer was influenced by the following constraints unique to the mobile environment:

- Available bandwidth within a cell may change dynamically (because it is impossible to control the number of users per cell). This leads to difficulties in guaranteeing QoS parameters such as delay bounds and bandwidth guarantees.

- Mobile hosts frequently encounter extended periods of disconnection (due to handoff or physical interference with the signal) resulting in significant loss of data causing poor TCP and UDP throughput.

[1] e.g., a MH may roam frequently between rooms in an office building but remain for many hours within the building – each room is a cell and all cells in the building are controlled by one SH.

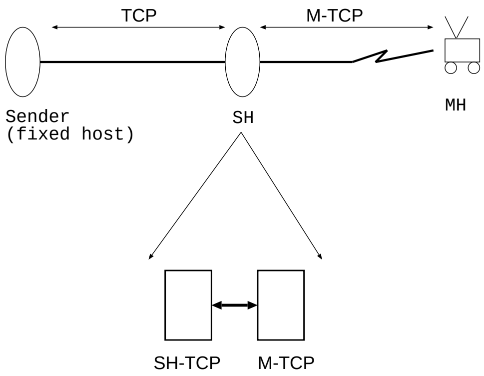

Figure 3: Setting up a TCP connection.

- Mobile devices are battery powered and hence power is a scarce resource. Protocols designed for use on mobile platforms must therefore be tailored to be power-efficient.

All of these factors point towards using transport protocols which are optimized specifically for the wireless link. It is not reasonable, however, to assume that the entire installed network base will (or should) upgrade to these mobile-friendly protocols as they may never have occasion to communicate with a mobile user. Hence, the obvious solution is to split transport connections in two. The existing protocols may continue to be used on the fixed network, and the protocols optimized for the wireless environment may be used in the mobile subnetworks. In our system, all connections set up by a MH, where the other endpoint is either another MH or is a fixed host, are split at the SH. Thus, the service-provider sets up a connection with the SH assuming the SH is the other end-point of the connection. The SH sets up another connection to the MH. This allows us to address the problems listed above as follows:

- A *bandwidth management module* at the SH (see [10]) assigns a fixed amount of bandwidth to each connection (this is recomputed periodically based on the changing needs of other MHs) and ensures that data is transmitted to all MHs at their assigned rate. This mechanism allows us to implement some limited form of QoS guarantees.
- In our architecture, the SH performs local error recovery to combat the efficiency problems resulting from loss over the wireless link.
- The SH tracks mobiles as they roam and uses this information to ensure that the number of duplicate packets transmitted to the mobile are kept small. Since every packet received by the MH has to be processed, consuming battery power, keeping the number of duplicates small reduces power consumption at the MH.

4 Design of M-TCP

Our goal in developing the M-TCP protocol was to provide a general solution to the problem of improving TCP's efficiency for mobile computing applications. Specifically, we wanted to design a protocol that had the following characteristics:

1. Improve TCP performance for mobile clients.
2. Maintain end-to-end TCP semantics.
3. Be able to deal with the problems caused by lengthy disconnections or by frequent disconnections.
4. Adapt to dynamically changing bandwidth over the already starved wireless link.
5. Ensure that handoffs (as a mobile roams) are efficient.

To this end we chose to use the *split connection* approach for implementing M-TCP because it fits in well with our general design philosophy articulated in the previous section and because it allows us to modify TCP on the mobile network to make it respond better to disconnections and low (or varying) wireless bandwidth.

Thus, every TCP connection is split in two at the SH. The TCP sender on the fixed network uses unmodified TCP to send data to the SH while the SH uses our version of TCP for delivering data to the MH. Figure 3 illustrates the manner in which a TCP connection gets split. The TCP client at the SH (called SH-TCP) receives segments transmitted by the sender and it passes these segments to the M-TCP client for delivery to the MH. ACKs received by M-TCP at the SH are forwarded to SH-TCP for delivery to the TCP sender. In the remainder of this section we discuss the behavior of SH-TCP and M-TCP in detail. Before doing so, however, we briefly discuss our assumptions regarding the mobile networking environment.

Wireless bandwidth will always be a precious resource and we therefore believe that it ought to be allocated to users based on their need. In our architecture, we use a bandwidth management module (see [10]), that resides at the SH, to perform this task. This module determines the amount of bandwidth to be allocated to each connection within each cell (unfortunately, a discussion of the implementation of this module is beyond the scope of this paper). Thus, when a mobile opens a TCP connection, it is allocated a fixed amount of bandwidth (this amount may change periodically depending on the needs of other mobiles and the current status of this mobile's connection). What implication does this have for the design of TCP? Clearly, since the bandwidth allocated to the mobile is fixed, there is little reason to implement TCP's sophisticated congestion control mechanisms between the SH and MH. A second assumption we make in our development is that the bit error rate visible at the transport layer will be small. This assumption is based on two observations: first, [14] notes that the BER over wireless channels is, in actuality, quite small (of the order of 10^{-5}) and, second, FEC and link-layer mechanisms can further reduce the observed bit error rate. Thus, we felt it necessary to design a TCP protocol that handles unremedied problems – disconnections and low-bandwidth – rather than a protocol that works well only in high BER environments.

4.1 The SH-TCP Client

When SH-TCP receives a segment from the TCP sender, it passes the segment on to the M-TCP client. However, unlike I-TCP and MTCP, it does not ACK this data until the MH does. SH-TCP is notified of MH ACKs by the M-TCP client running at the SH. It is easy to see that this behavior ensures that *end-to-end TCP semantics are maintained*. However, how do we ensure that the sender does not go into congestion control when ACKs do not arrive (because the MH was temporarily disconnected and did not receive the data, or could not send the ACKs)? Our approach is to **choke** the TCP *sender* when the MH is disconnected, and allow the sender to **transmit at full speed** when the MH reconnects by manipulating the TCP sender's window. Specifically:

- Let W denote the currently advertised receive window at SH-TCP. Say the window contains $w \leq W$ bytes. Assume that the MH has ACK'ed bytes up to $w' \leq w$. SH-TCP sends an ACK (or ACKs) for bytes up to $w' - 1$ in the normal way. As and when the MH ACKs more data, more ACKs are generated but one last byte is always left unacknowledged.

- Say the MH disconnects after having ACK'ed bytes up to w'. The M-TCP client assumes that the MH has been temporarily disconnected because it stops receiving ACKs for bytes transmitted after w'. M-TCP sends an indication of this fact to SH-TCP who then sends an ACK for the w'th byte to the sender. This ACK will also contain a TCP window size update that **sets the sender's window size to zero**. When the TCP sender receives the window update, it is forced into *persist* mode. While in this state, it will not suffer from retransmit timeouts and will not exponentially back off its retransmission timer, nor will it close its congestion window [2].

  Note that as long as the SH-TCP sends ACKs for the persist packets sent by the TCP source, the state of the sender does not change no matter how long the disconnection period lasts.

- If the MH has not disconnected but is in a cell with very little available bandwidth, SH-TCP still sends an ACK for byte w' with a window size set to 0. SH-TCP estimates the round trip time (RTT) to the TCP sender and estimates the RTO interval. It uses this information to preemptively shrink the sender's window before the sender goes into exponential backoff (to implement this scheme, we maintain a timer at the SH that is initialized to this estimated RTO value).

- When a MH regains its connection, it sends a greeting packet to the SH (see [16]). M-TCP is notified of this event and it passes on this information to SH-TCP which, in turn, sends an ACK to the sender and reopens its receive window (and hence the sender's transmit window). The window update allows the sender to leave

[2] RFC 1122 states that TCP clients go into persist mode when the window is shrunk to zero. However, in order to shrink the window to zero, the receiver must send a *new ACK*. This is why we hold on to the last byte at SH-TCP. To grow a window, however, we can retransmit an old ACK.

persist mode and begin sending data again. Since the sender never timed out, it never performed congestion control or slow-start. *Thus the sender can resume transmitting at full-speed!* The sender *will* begin sending from byte $w + 1$ here though, possibly duplicating previous sends, so we don't want to shrink the window more often than necessary.

A potential problem with this protocol is the following: say the sender only sends data to the MH occasionally. Specifically, if the sender only needed to send bytes 0 to 50 to the MH, according to our protocol, SH-TCP will ACK bytes up to 49 but will not ACK byte 50. This will cause the sender to timeout and retransmit this byte repeatedly. Eventually, after 12 retransmissions, the TCP sender will give up and quit! This is clearly an undesirable situation. We handle this problem by not allowing SH-TCP to shrink the sender's window if it believes that there will be no more new ACKs from the MH that will allow it to open up the sender's window back up again. Thus, when SH-TCP thinks that the sender will timeout (and if the MH is in a cell with plenty of available bandwidth), it simply ACKs the last byte. At this point, there will be no saved byte at SH-TCP to allow it to shrink the sender's window. Observe that this is not a problem because the sender didn't really have any new segments to send to the MH. As and when the sender does transmit new data to the MH, SH-TCP reverts to its previously described behavior.

4.2 Design of the M-TCP protocol between the SH and MH

Our goal in designing SH-TCP was to keep the TCP sender's congestion window open in the face of disconnection events at the MH. On the wireless side, on the other hand, our goal is to be able to recover quickly from losses due to disconnections, and to eliminate serial timeouts.

In designing M-TCP, we use a scheme similar to SH-TCP except that we have more design freedom since we can modify the protocol at both ends of M-TCP. M-TCP at the mobile receiver is very similar to standard TCP except that it responds to notifications of wireless link connectivity. When M-TCP at the MH is notified (by the communications hardware) that the connection to its MSS has been lost, it freezes all M-TCP timers. This essentially ensures that disconnections do not cause the MH's M-TCP to invoke congestion control. Observe that neither acknowledgements nor data are lost during a disconnection. When the connection is regained, M-TCP at the MH sends a specially marked ACK to M-TCP at the SH which contains the sequence number of the highest byte received thus far. It also unfreezes M-TCP timers to allow normal operation to resume.

At the SH we need to ensure that M-TCP monitors link connectivity and takes appropriate action either when a disconnection occurs or when a reconnection occurs. How does M-TCP determine that the MH has been disconnected? In our design M-TCP monitors the flow of ACKs from the MH in response to segments transmitted on the downlink. If it does not receive an ACK for some time, it automatically assumes that the MH has been disconnected. It then informs SH-TCP of this fact and SH-TCP reacts by putting the TCP sender in persist mode. M-TCP at the SH knows when the MH reconnects because M-TCP at the MH retransmits the last ACK, and marks it as a reconnection ACK, on being reconnected. M-TCP responds by informing SH-TCP who reopens the sender's transmit window. This behavior is based on our assumption that the bandwidth management module at the SH assigns bandwidth to each connection and regulates its usage so there is no need to invoke congestion control at the SH when ACKs are not received from the MH. M-TCP at the SH works as follows:

- When a retransmit timeout occurs, rather than retransmit the segment and shrink the congestion window we force M-TCP into persist mode. We assume the absence of an ACK means that the mobile receiver has temporarily disconnected, hence retransmissions are futile until the MH notifies the SH that it has reconnected.

  At first glance our implementation may appear to have a problem when operating in high BER environments. This is because a significant proportion of packet loss in such environments will be due to the high BER and not due to disconnections. We respond to this observation by making two points:

  - We believe that in most mobile environments link layer solutions will ensure that the bit error seen at the transport layer is small.
  - Even if the mobile environment does have high BER, the M-TCP sender will come out of persist mode quickly and resume regular data transmissions. This is because, when in persist mode, M-TCP will send persist packets to the MH who will be forced to respond when it receives these packets. Currently, the persist packets are generated at increasing intervals starting at 5 seconds and doubling until the interval becomes 60 seconds. We could change the interval to be equal to the RTT between the SH and the MH to ensure earlier recovery (we have not implemented this simply because of our assumption of low BER).

- When the SH receives a specially marked reconnection ACK, it moves back out of persist mode and retransmits all packets greater than the ACK'ed byte as these must have been lost during the disconnection. If the SH misinterpreted a lengthy delay as a disconnection, the MH will eventually ACK the transmitted packets, which will also move the SH M-TCP out of persist but will not cause all packets to be retransmitted.

Since M-TCP at the SH begins sending again immediately each time it is notified that the MH has reconnected, it recovers immediately from disconnections. Since it does not rely on the retransmit timer except to eliminate duplicate sends, and never backs off the timer in any case, serial timeouts are eliminated. Finally, since M-TCP relies on the SH's regulation of bandwidth and not the congestion window, it always sends at the maximum rate possible.

4.2.1 Enhancements to M-TCP: Compressed M-TCP

Another problematic aspect of the wireless link is the low (and possibly varying) bandwidth available to the mobile. We have added an option to allow data compression in order to make more efficient use of the limited wireless bandwidth. If the option is enabled, the data portion of each downlink M-TCP packet is compressed at the SH before transmission and decompressed at the mobile receiver.

4.3 Benefits of our Approach

What are some benefits of our approach?

- Clearly, end-to-end TCP semantics are maintained.
- The efficiency of the TCP connection is not degraded due to disconnections since the sender is prevented from going into exponential backoff and slow-start (this is even true for arbitrarily long disconnections – e.g., Metricom's Ricochet network where disconnections are as long as 1 minute).
- The sender's timeout estimate is based on the round trip time to the MH (including effects of disconnections). This means that the timeout intervals at the sender will be more 'realistic' causing fewer unnecessary retransmissions over the wireline networks. This also ensures that the sender's transmission rate adapts to the available wireless bandwidth.
- If the MH moves from the domain of one SH into the domain of another SH, the old SH does not need to forward the socket buffers (as in I-TCP). When the MH temporarily disconnects while moving between the two SHs, the old SH's SH-TCP shrinks the TCP sender's window to zero. When the MH connects to the new SH, SH-TCP at the new SH grows the sender's window to its full size which causes the sender to automatically retransmit all outstanding segments to the *new SH.* Thus, only a small amount of state is transferred from the old SH to the new SH (2 tcpcbs and inpcbs and 1 mtb) – no more than 500 bytes in all. It is noteworthy that, in contrast to our protocol, transfer of large socket buffers in I-TCP [3] may take as long as 1.5 seconds (see [4]).
- Serial timeouts and the accompanying periods of inactivity are avoided by disabling exponential backoff in M-TCP.

Observe that M-TCP is more efficient than TCP because the sender is spoofed into transmitting data to the SH at the rate at which data can be delivered to the MH. In addition, the problems associated with using TCP in a mobile environment are avoided.

5 Implementation of M-TCP

In this section, we examine implementation issues and the protocol in more detail. Again, consider the TCP to SH-TCP portion of the connection first. Figure 4 shows the state diagram of SH-TCP's window shrink control functions. WS is the window state, and the wintimer is the SH's timer controlling the sending of window shrink updates to the TCP sender. In most cases, SH-TCP resides in the NOSHRINK state, and an M-TCP ack for byte x generates a TCP ack for byte $x-1$. Normally, when the timer expires, SH-TCP sends a window shrink update to shrink the TCP sender's window to 0 forcing it into persist state. However, SH-TCP **does not** shrink the window if:

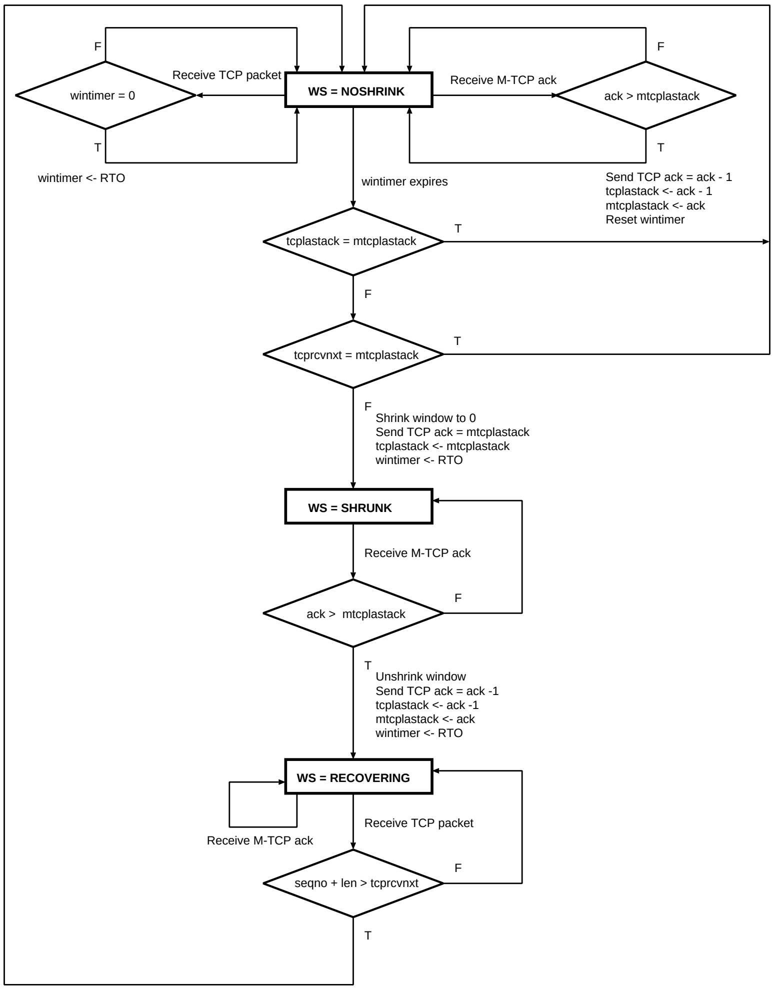

Figure 4: SH-TCP window shrinking causality.

- It has already acknowledged all the data to the TCP sender and thus has no unacknowledged bytes to use to shrink the sender's window. Note that window window shrinks are ignored unless accompanied by a **new** acknowledgement.
- Or if the saved byte will end up acknowledging all the data SH-TCP has received. In this case, SH-TCP just acknowledges the last byte without shrinking the window. This ACK will reset (or clear) the REXMT timer at the TCP sender. This scheme ensures that SH-TCP acknowledges everything in a timely manner in the event that the TCP sender has no further data to send (but does not end the connection). If it so happens that there is outstanding data, but SH-TCP has not received it, the acknowledgement at least keeps the TCP sender from invoking congestion control for another RTO.

In all other cases, expiration of the wintimer moves SH-TCP to the SHRUNK state. While in this state, SH-TCP acknowledges persist packets from the TCP sender but keeps the window size set to zero. When a new M-TCP acknowledgement is received, SH-TCP reopens the send window and moves into RECOVERING state. This state is necessary because many existing TCP implementations reset snd_max (the highest byte sent) to snd_una (the highest byte acknowledged) when they exit persist state. SH-TCP needs to ensure that it does not acknowledge a byte that is greater than the TCP sender's view of snd_max. Since SH-TCP very well may have received bytes higher than snd_una, it needs to keep track of the sequence numbers of incoming packets and only move back to the NOSHRINK state when its own view of snd_max is consistent with the TCP sender's. This state is unnecessary but benign for TCP implementations which do not reset snd_max.

5.1 M-TCP State Diagram

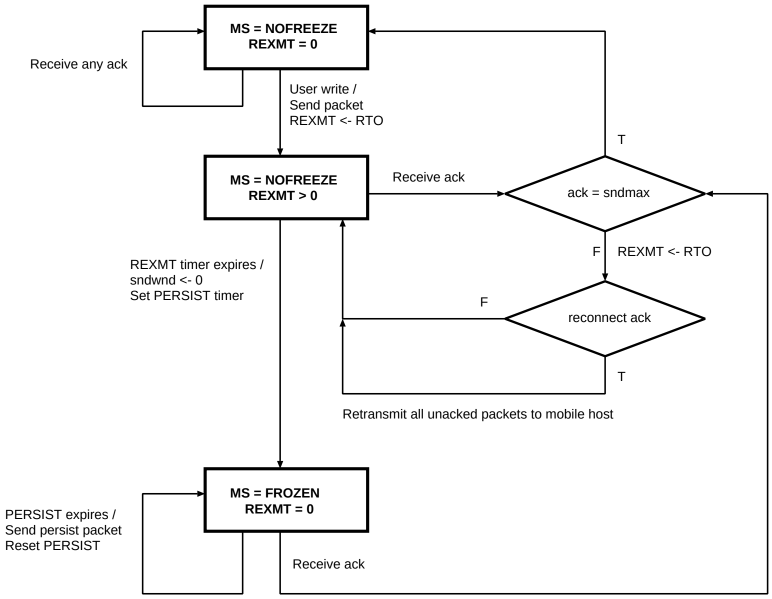

Figure 5: M-TCP timer freezing causality at the SH.

Now we consider the behavior of the M-TCP protocol (at the SH) on the wireless portion of the connection. Figure 5 shows a state diagram illustrating M-TCP's freezing functions (where freezing is equivalent to moving into persist state voluntarily). In most cases, M-TCP's state (MS) is NOFREEZE, either with a zero or non-zero REXMT timer depending on whether there is outstanding data to be acknowledged by the mobile receiver or not. When the REXMT timer goes off, rather than invoking congestion control, M-TCP voluntarily moves into persist state (MS = FROZEN). It then waits for an acknowledgement from the mobile receiver. Since, M-TCP at the SH only enters FROZEN state when there is unacknowledged data, the mobile will eventually send an acknowledgement to unfreeze it. There are two cases we need to consider:

- The MH really did temporarily disconnect, in which case it will generate a specially marked acknowledgement on reconnection. This moves M-TCP at the SH back into NOFREEZE state and causes it to retransmit all unacknowledged bytes.

  We use an unused combination of TCP flags in a standard TCP acknowledgement packet to allow the MH to signal its reconnection to the SH.

- The MH did not disconnect, in which case it will acknowledge the data the SH sent it. Again, this moves the SH back into NOFREEZE state but does not cause it to resend packets thus keeping the number of duplicates small.

Note that in either of these cases, the acknowledgement may be lost. Again, two methods for recovery are available. If the MH moves through another cycle of disconnection and reconnection, it will generate another specially marked acknowledgement. Otherwise, the MH stays connected in the same cell, allowing it to respond when M-TCP at the SH sends a persist packet. Thus, M-TCP at the SH recovers whether the MH moves or not.

5.2 Implementation Notes

M-TCP/SH-TCP Interaction

M-TCP and SH-TCP communicate through a shared data structure, the mtb (M-TCP transition block), which tracks the last acknowledged byte in each direction, the state of the connection on the fixed and wireless segments, the window shrink and freeze state, and other information pertinent to the connection. When passing application data from one protocol stack to the other, an attempt was made to make this process as efficient as possible. No extra copies were introduced at this stage. Mbufs are moved directly from the incoming socket buffer of one protocol to the outgoing socket buffer of the analogous protocol. Note that TCP's (and hence M-TCP's) reliable delivery feature requires that a copy be done when actually *sending* a packet. Hence, in comparison to a router which would just pass the packet through, SH-TCP/M-TCP does one additional copy.

Handoff

Transfer of state is actually handled by the handoff module that is incorporated within out transport layer design (see [10]). When handoff begins, M-TCP is notified and the send window is shrunk on the TCP side. Next, some of the state regarding the connection is serialized at the old SH and passed to the new SH. At the new SH, the connection state is unserialized, and the TCP send window is re-opened. Note that the contents of the socket buffers (which can be quite large) are *not* serialized or transferred – they are discarded. The new SH reopens the send window at the TCP sender after handoff is completed. Thus, the TCP sender retransmits a full window worth of packets to the new SH. This approach makes for very fast handoffs at the expense of additional retransmissions on the fixed network. Serialization and unserialization require several functions at the SH including:

- Freezing of connection state.
- Removal of inpcbs and tcpcbs without affecting connection state.
- Spontaneous generation of inpcbs and tcpcbs in connected states.
- The ability to bind inpcbs to non-local addresses.
- Unfreezing of connection state.

5.3 Design Decisions

5.3.1 Connection Setup

When initially setting up the split connection, an order for completion must be determined. For a connection originating on the fixed network (the order for a connection originating on the mobile subnetwork is symmetric) there are two alternatives:

- Complete the TCP connection between the fixed host and the SH. Then initiate and complete the M-TCP connection between the SH and the MH.
- Complete the connections on both halves in lock-step. In other words, when the SYN packet is received from the fixed host, send a SYN packet to the MH. When the SYN/ACK is received from the MH, send a SYN/ACK to the fixed host. Continue until both segments of the connection are completed.

We chose the former option because it allowed a cleaner separation of the two protocols. If a mobile receiver moves during the connection setup phase, handoff of state to a new SH is simplified if only one segment of the connection is incomplete.

Another decision we made was to make the SH's presence transparent to the TCP sender in the fixed network from the addressing point of view. On connection setup, the SH creates sockets whose local addresses are bound to the TCP sender's address and to the MH's address. In this way, the TCP sender always sends to the MH's address, and the MH always sends to the TCP sender's address.

5.3.2 Wintimer Settings

The wintimer is used to determine when to send a window shrink update to the TCP sender. As such, it should expire at a time giving the SH the opportunity to generate a packet, and for the packet to propagate to the TCP sender before the sender times out and invokes congestion control. Assume that the TCP sender REXMT timer is set to $\mathrm{F}_{TCP}\mathrm{RTO}$, which is based on the sender's estimate of the round-trip propagation delay (RTT) between itself and the MH. Since the SH is situated between the TCP sender and the MH, it has RTT and hence RTO estimates for both segments of the connection ($\mathrm{S}_{TCP}\mathrm{RTO}$ and $\mathrm{S}_{M-TCP}\mathrm{RTO}$) and can thus estimate the TCP sender's RTO. We assume that $\mathrm{F}_{TCP}\mathrm{RTO} \approx \mathrm{S}_{TCP}\mathrm{RTO} + \mathrm{S}_{M-TCP}\mathrm{RTO}$.

How can this information be used to set the wintimer? Let us say that the sender generated a new segment at time 0. Its retransmit timer is set to go off at time $\mathrm{F}_{TCP}\mathrm{RTO}$. We can assume that the packet takes $\mathrm{F}_{TCP}\mathrm{RTO}/2$

time to propagate to the SH. The SH forwards the packet to the MH and hopefully receives an ACK. The total time elapsed (counting from time 0) is thus $\mathrm{F}_{TCP}\mathrm{RTO}/2 + \mathrm{S}_{M-TCP}\mathrm{RTO}$. The ACK is forwarded on to the sender and arrives at time $\mathrm{F}_{TCP}\mathrm{RTO}$ preempting a retransmit event. If, however, the SH did not receive an ACK within time $\mathrm{S}_{M-TCP}\mathrm{RTO}$, it needs to send a window shrink message to the sender so that it arrives before the retransmit timer goes off. Therefore we need to set wintimer to $\mathrm{S}_{M-TCP}\mathrm{RTO}$ whenever a new segment arrives at the SH.

5.3.3 Selective Acknowledgements (SACK)

We have chosen not to implement SACK at this time. Although SACK has been shown to be very effective against high BER, its effect on disconnections is likely to be marginal. Note that since the mobile host acknowledges the highest byte received in the end-of-disconnection ack the only scenario where selective acks could be useful is as follows:

1) SH begins sending packets.

2) MH goes into fade and comes back out before SH transmits 1 window worth of packets.

3) MH receives some packets at the end of the window.

4) MH can use SACK so that SH need not retransmit packets at end of window.

Since fades are relatively long, this situation is unlikely to happen very frequently. Hence SACK is of limited benefit when dealing with disconnections.

5.3.4 Presence of Slow-Start Mechanism in M-TCP

We have chosen not to shrink the congestion window (on M-TCP at the SH) on timeout, since these timeouts are caused by disconnections and not congestion. As a result, slow-start behavior is limited to the beginning of a connection. It may also be beneficial to remove congestion control, and hence slow-start, at this point. Since the SH controls bandwidth allocation for each MH, the ramping-up of throughput to test bandwidth availability is unnecessary. For large data transfers, the presence or absence of congestion control will have little effect on delay, but small data transfers may benefit from the complete removal of congestion control.

6 Performance Measurement

We implemented M-TCP in the NetBSD kernel and in this section we discuss the performance of our implementation. Our goal in running the various experiments was to examine M-TCP's performance and, in addition, to determine if the gateway approach we have adopted could be a potential bottleneck in the deployment of large-scale mobile networks. Specific questions we looked at were:

1. How well does M-TCP perform in environments where the mobile is subjected to frequent disconnection events? How does it perform when the wireless bandwidth available is low? Performance was measured as the time taken to transfer large files.

2. How do losses over the wired Internet affect M-TCP's performance? If the loss over the Internet is high we expect disconnection events and high BER over the wireless link to have a much smaller impact on end-to-end TCP performance.

3. What is the performance gain if the SH compresses data? Note that if wireless bandwidth is at a premium, it may be a good idea to employ compression. However, the processing cost at the SH will increase substantially.

4. How many duplicates does the MH receive? This measures the extra amount of processing that the MH will have to perform. Since processing consumes battery power, we would like to keep the number of duplicates received to a minimum.

5. What is the processing overhead at the SH? This overhead will include the extra checksum computation by SH-TCP and M-TCP and extra data copies (if any). Based on the processing overhead, we can determine if the gateway approach is scalable to large mobile networks.

In order to evaluate the performance of our protocol, we used the experimental testbed shown in Figure 6. As illustrated, the SH and MSS code run on a single Pentium PC. The MH code runs at another Pentium PC that is connected on an isolated ethernet to the SH. We employed two different senders for the experiment. The close sender was located 5 hops away (connected via ethernets) and the distant sender was located 15 hops away. In order to emulate[3] a low-bandwidth downlink (32kbps) we modified the IP code at the SH as follows. All calls to ip_output() are intercepted and then, based on the link speed and packet size (including TCP and IP headers), a timer is set to go off each time a packet can be sent on the wireless link. At each timeout one packet is removed from a link queue and ip_output() is called normally. We emulated a low-bandwidth uplink from the MH to the SH by modifying the MH's IP code in a similar manner. In order to emulate disconnections, we modified the MH's IP code as follows. A MH may be viewed as either being connected or disconnected. We thus divide time into alternating periods of connection and disconnection. Packets transmitted to the MH during a disconnection period are discarded by IP at the MH. In addition, IP informs M-TCP at the MH of the disconnection event.

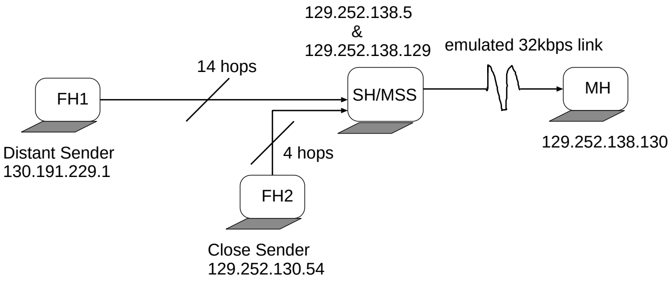

Figure 6: Experimental set up.

For our experiments we used a 32 Kbps downlink and a 8 Kbps uplink. In order to model disconnection events, for most of our experiments, we assume a cell latency mean of 5 seconds[4]. Disconnection length means varying from 0.5 to 4.5 seconds giving disconnections that range from from 10% to almost 50% of the cell latency[5]. If we transfer a 500KByte file over the 32kbps link, it takes 62.5 seconds. With a 5 second mean cell latency, the mobile will encounter approximately 12 disconnection events. Note that [3, 12] also use cell latencies and disconnection periods in the same range. For different experiments, the cell latency and disconnection lengths were chosen either from a standard normal distribution or from an exponential distribution.

In order to get a more complete picture of M-TCP's performance, we ran additional experiments using much longer cell latencies and disconnection intervals, typically of the order of minutes, to model the environment of large cells. However, while these results also show that M-TCP performs far better than TCP, we find them to be of limited interest because, if a mobile has a very long cell latency, it is likely that it will receive all of the data before encountering a disconnection event (except in the case of long data transfers). Furthermore, we expect cells to be relatively small in the future (to enable higher bandwidth wireless data access) implying much smaller cell latencies and disconnection intervals (as we discussed in the previous paragraph).

6.1 M-TCP Performance in the Wireless Environment

We tested M-TCP performance against TCP when the emulated wireless link described above was used. The time to transfer a file from a fixed host TCP sender to a mobile receiver was measured. In Figure 7 we plot file transfer time for 500K and 1MB files (respectively) against disconnection length where latency and disconnection length were normally distributed random variables and the TCP sender was 5 hops away. Here we can see that M-TCP performs much better than TCP across the range of disconnection lengths and perhaps, more importantly, is much more stable than TCP. As we can see by the 95% confidence intervals (each test was repeated 10 times), TCP transfer times varied quite widely and in fact some transfers failed to complete (the results from these tests were discarded), whereas M-TCP times were very consistent. Note also that M-TCP performance is close to optimal, where we calculate optimal times based on the assumption that the wireless link is the bottleneck, i.e., how long

[3]Emulating the wireless link in this manner gives us precise control over the available wireless bandwidth, disconnection events, etc.

[4]If a cell has a diameter of 100 meters (approximately one city block), then a car travelling at 60Kmph will have a cell latency of 6 seconds not including handoff delays, fades, etc.

[5]When a mobile moves into a new cell, it has to complete a handoff before the MSS can begin sending data. It is also possible that the new mobile may not receive any bandwidth in a crowded cell at all. Thus, it is reasonable to have disconnect period lengths of the stated range.

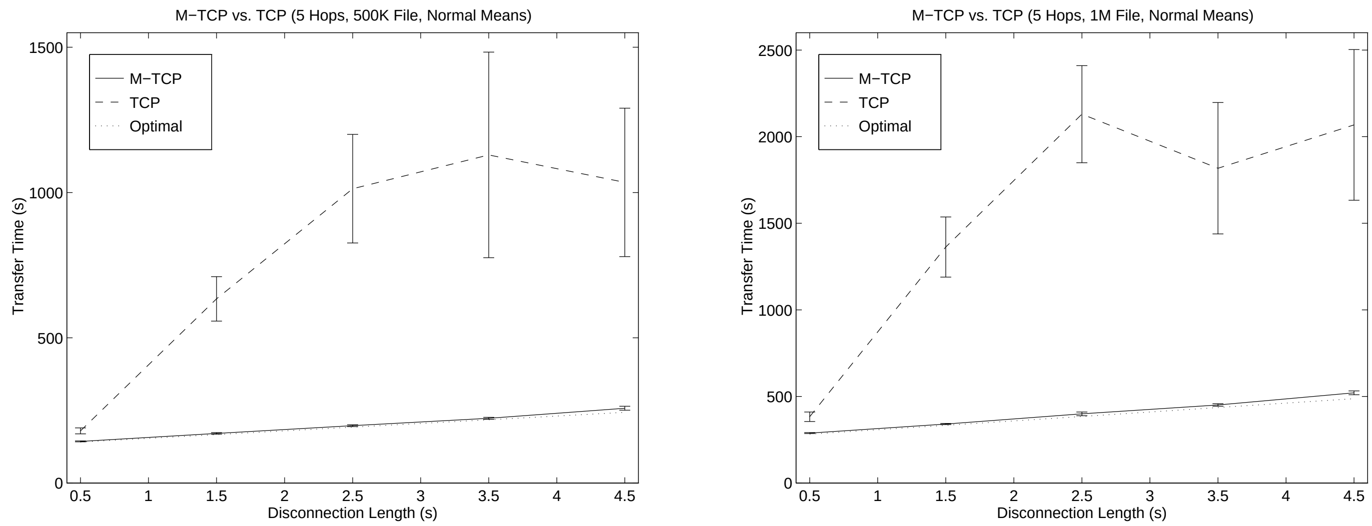

Figure 7: M-TCP vs TCP performance.

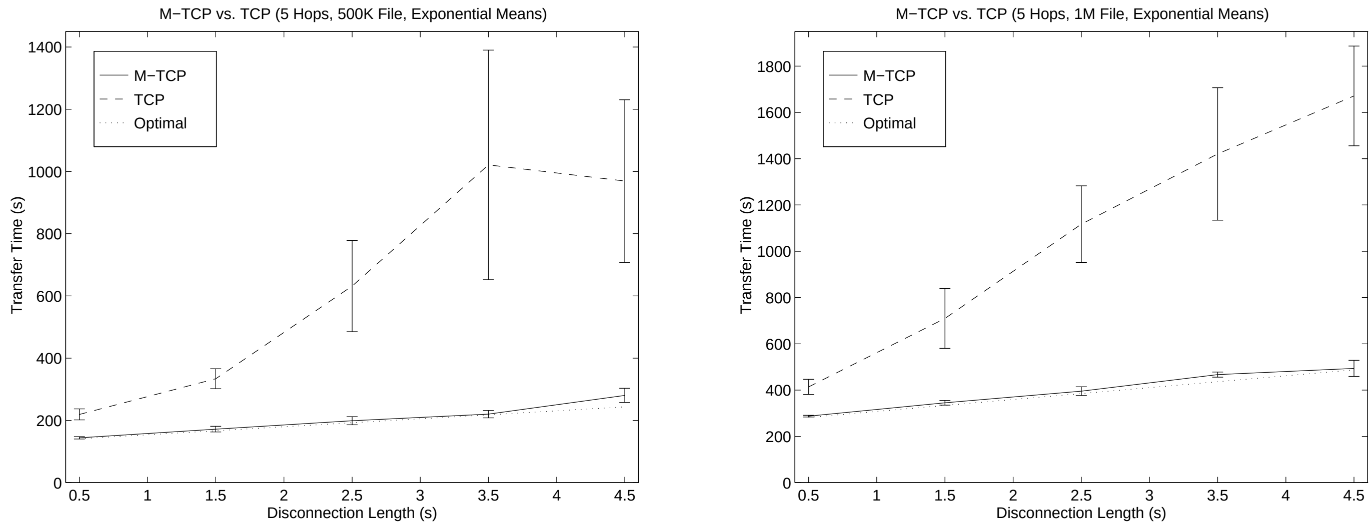

Figure 8: M-TCP vs TCP performance.

does it take to transfer a file over a 32 kbps link under disconnections of the given distribution. We obtain similar results for the exponentially distributed means, see Figure 8, although there was slightly more variation in M-TCP's performance.

In Figure 9, we plot M-TCP's transfer time against the optimal transfer time for 3 file sizes (100K, 500K and 1M) over 5 hops. The figure on the left shows the results of tests performed when the cell latency and disconnection length means were normally distributed, and the figure on the right shows results for exponential means. For both distributions, all three file sizes, and all disconnection lengths, M-TCP's performance is very close to optimal. It is interesting to observe that as the file size increases, the difference between M-TCP's transfer time and the optimal transfer time increases. This increase is due to the fact that more packets are transferred and each packet encounters additional processing delay at the SH. As we will discuss in section 6.4, processing delay is cumulative over several packets.

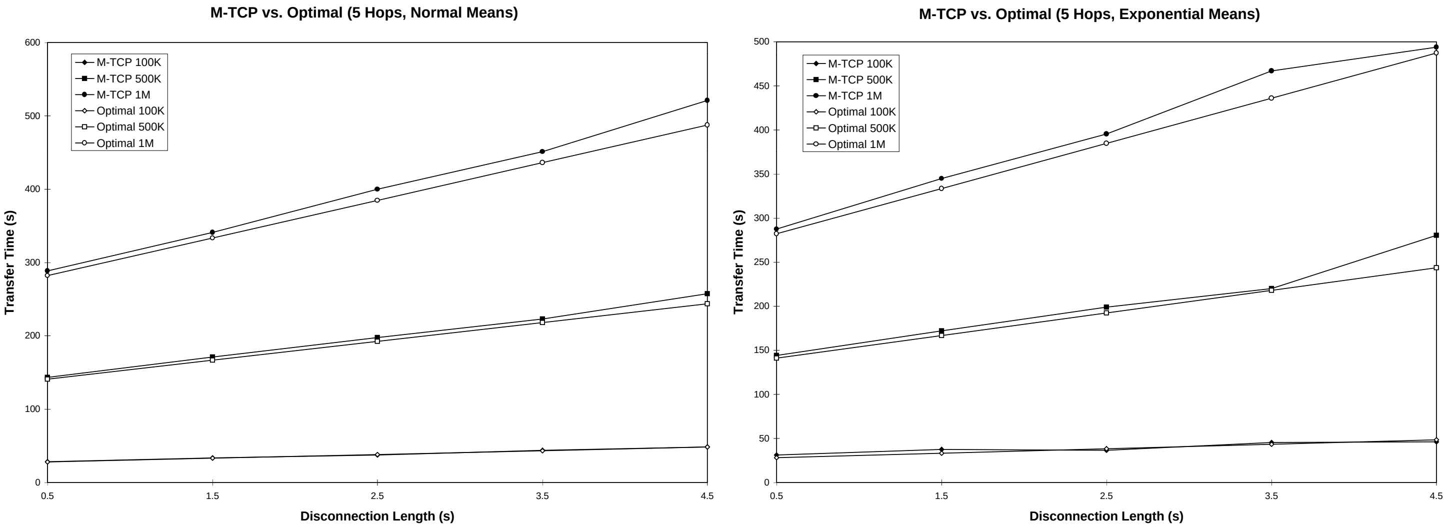

Figure 9: M-TCP vs. Optimal performance.

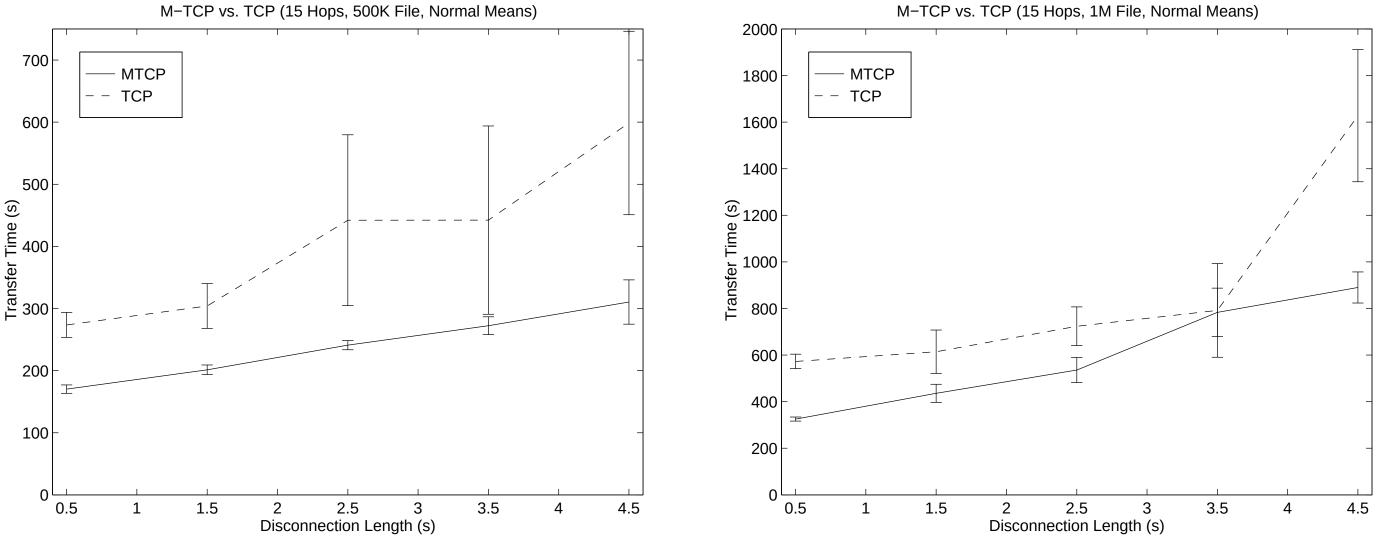

Figure 10: M-TCP vs TCP performance.

When we moved the TCP sender 15 hops away from the mobile receiver, a great deal more variation was observed in the performance of both, TCP and M-TCP. Figures 10 and 11 plot the transfer time against disconnection lengths for normal and exponential distributions respectively. In each figure, the graph on the left plots the results for a 500KB file transfer and the graph on the right plots the results for a 1MB file transfer. The variation appears to be caused by losses on the fixed network, which we had originally assumed to be fairly negligible. Upon further measurement, however, we found losses on the Internet to be substantial, even during times of light traffic. We found loss rates of 5.1% when sending independent bursts of 6 packets and loss rates of 23.8% when sending bursts of 32 packets (these results were obtained at night and the packets were sent by the FH1 in Figure 6). Loss rates during the daytime were much worse.

Losses in the fixed network require a fast retransmit or retransmit timeout at the TCP sender for recovery, both of which incur delays and which cause (at least partial) congestion control to be invoked at the sender. Due to this effect and the apparent random nature of these losses, M-TCP's relative performance gain is reduced and its variability increased. In most cases, however, M-TCP still performs much better than TCP and has much lower variability. Finally, note that for a lossy fixed network, we do not believe that the optimal transfer time can be modelled based only on the assumption that the wireless link is the bottleneck. Therefore, we have not plotted the optimal transfer time in the these graphs.

Finally, we conducted experiments to study the effect of very long cell latencies and disconnection periods on M-TCP's performance. We expected a smaller performance difference between TCP and M-TCP for these cases because, even though the TCP sender (in the case of unmodified end-to-end TCP) will shrink its congestion window to 1 due to repeated timeouts during the long disconnection period, it has plenty of time to grow the window back up to its maximum size after reconnection. Figure 12 plots the file transfer times for a 12MB file when the mean cell latency is 5 minutes and the mean disconnection length is 1 minute. As we can see M-TCP outperforms TCP for both, the close sender and the far sender cases. However the performance difference is less pronounced for the far sender case due to the higher error rate over the Internet.

6.2 Power used on the Mobile Host: number of duplicates received by the MH

Since mobile devices will be battery operated, power consumption at the mobile receiver is an important measure of protocol performance. We therefore determined the number of duplicates received per packet at the MH. Since each reception consumes power, duplicates should be kept to a minimum. Figure 13 shows the number of receives per byte for normal and exponential (respectively) distributions of cell latency and disconnection length means. While M-TCP generally sends fewer duplicates than TCP, the difference in the ratios (e.g., 1.02 vs. 1.06) is fairly negligible.

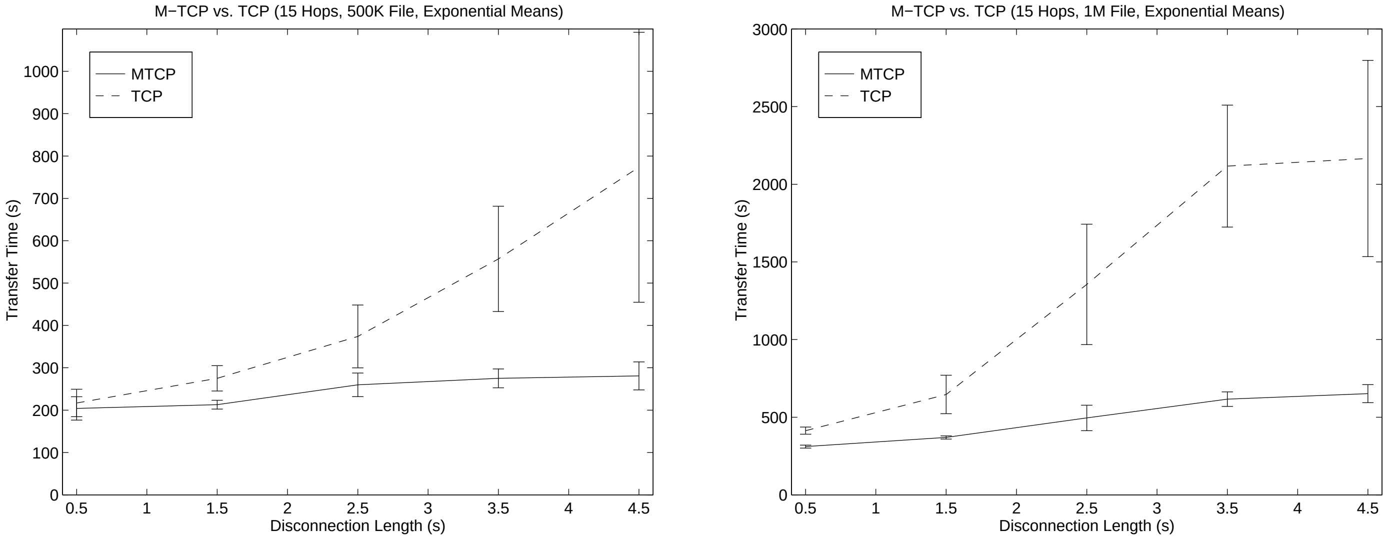

Figure 11: M-TCP vs TCP performance.

6.3 Dealing with a Slow Wireless Link : Compressed M-TCP

Another problematic aspect of the wireless link is its slow speed. This means that large file transfers take a long time, a problem which is only exacerbated by the effects of frequent disconnections. One possible scheme for reducing bandwidth usage on the wireless link is to implement M-TCP header compression, much like the TCP header compression used on SLIP links. Unfortunately, however, there may be problems with this scheme in the face of high BER and lost packets since SLIP header compression relies on incremental differences in consecutive packets. In addition, the packet size is not decreased by that much using header compression (40 bytes may be transformed to as few a 3 bytes) – 2% to 7% depending on packet size.

We believe that much higher gains can be had by compressing the data portion of the packet. We therefore have added data compression to M-TCP (as an option) to make more efficient use of the limited wireless bandwidth. The data portion of each downlink M-TCP packet is compressed before sending and decompressed at the mobile receiver. It is important to note, though, that processing costs at the SH may be prohibitive. Since the SH may maintain connections for many MHs, and compression is a cpu-intensive operation, it may not be possible to use compression on all open connections. We suggest that compression only be used on connections where the wireless bandwidth is very limited, and hence where the transfer time savings are especially large.

We implemented compression by using existing gzip code. Essentially, we pushed gzip down into the kernel, stripping off the user interface sections and reducing the encryption/decryption methods to one. We also threw out most of the gzip header (which is prepended to each packet) to save bandwidth on the wireless link.

Figure 14 shows processing times at the SH and the MH for compression and decompression (respectively) of packets ranging in size from 100 bytes to 1400 bytes. Gzip provides a compression level option which we varied from 1 to 9 (1 being the least amount of compression, 9 being the most). In fact, the compression ratio only changed by a few percent between the two extremes, probably due to the small size of the packets. The processing times do differ by a non-negligible amount, up to $500\mu s$ in some cases. Based on the tradeoff between processing time and compression ratio obtained, we decided to use a compression level of 3.

In Figure 15 we show file transfer times of a 1M file versus disconnection lengths for a TCP sender which is 5 hops away, for cell latencies and disconnection length means which are normally and exponentially distributed respectively. Original packet sizes were 1460 bytes. Compressed M-TCP outperformed M-TCP by a wide margin for both text and binary files. We can see that the binary files took longer to transfer than the text files because they did not compress as well, but both compressed types transferred much faster than uncompressed M-TCP. Figure 15 shows the results for a TCP sender which is 15 hops away. Compressed M-TCP still outperforms uncompressed M-TCP, but by a smaller margin. This is due to the fact that the packet size used in non-local connections is only 512 bytes, giving less opportunity for compression.

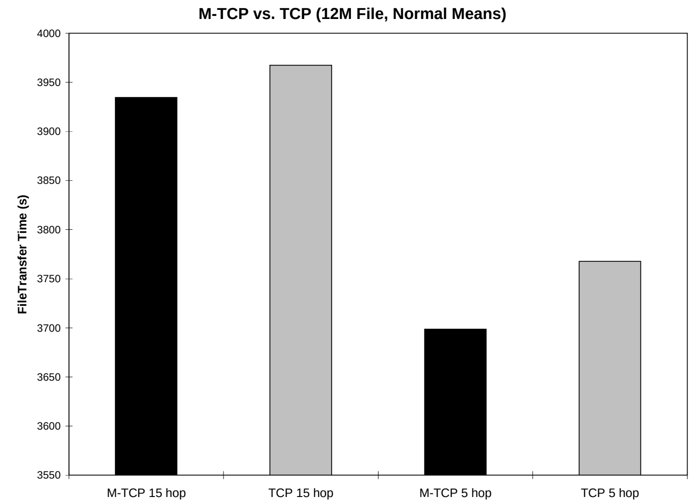

Figure 12: Relative performance of M-TCP and TCP for long cell latencies.

6.4 Justification of the Split-Connection Approach

Several criticisms, including extra processing overhead and potential delays have been aimed at approaches that use split-connections for implementing mobile transport protocols. We performed three different sets of experiments to refute these criticisms:

1. The first set of experiments measures the processing cost at the gateway nodes as a way to determine the feasibility of using the split-connection approach for deploying large-scale mobile networks.

2. M-TCP takes a small performance hit for small send buffer sizes. This is because there is a per-packet processing delay at the SH. With larger send buffer sizes, this cost is amortized over the window size. The second set of experiments illustrates this cost and justifies the 16K send buffer size we used in the previous section.

3. Finally, a macroscopic way of determining the total cost of using M-TCP is to run experiments *without wireless links and without disconnections*. If M-TCP fares better than TCP then, clearly, the delay incurred at the SH (by M-TCP) can be ignored.

A common argument against the split-connection approach is the extra processing (extra checksum computation as well as the extra data copies) required for each packet as it passes up to the transport layer and back down at the gateway, rather than just being routed through the network layer. In our implementation we eliminate any extra data copies when moving between socket buffers by passing pointers to the data. We added instrumentation to the kernel at the entry and exit points of IP and measured packet processing times for standard TCP packets, which were routed straight through, and for M-TCP packets, which passed through the transport layer. The processing times at the SH for packets of varying sizes are shown in the left-hand graph in Figure 17.

The TCP results include just IP processing and are steady around 31μs for all packet sizes. A larger packet size is necessary to see the effects of the IP checksum on processing time. M-TCP processing time grows with packet size from 91 to 112μs as expected. Subtracting the 31μs of IP processing (which is not entirely valid, since it includes ip_forward processing which does not apply to M-TCP packets), gives 60-81μs of transport-layer processing per packet. At 31μs per packet (1400 bytes each), a router could process 361 Mbps of IP forwarding traffic. At 112μs per packet, a gateway (Pentium PC) could potentially process 100 Mbps of MTCP data traffic (we are, of course, ignoring data transfer speeds within the PC itself).

We tested the file transfer time (for a 1MB file) of TCP and M-TCP using different send buffer sizes at the TCP source. These tests were conducted from the source 5 hops away, contain no disconnection events and run at full

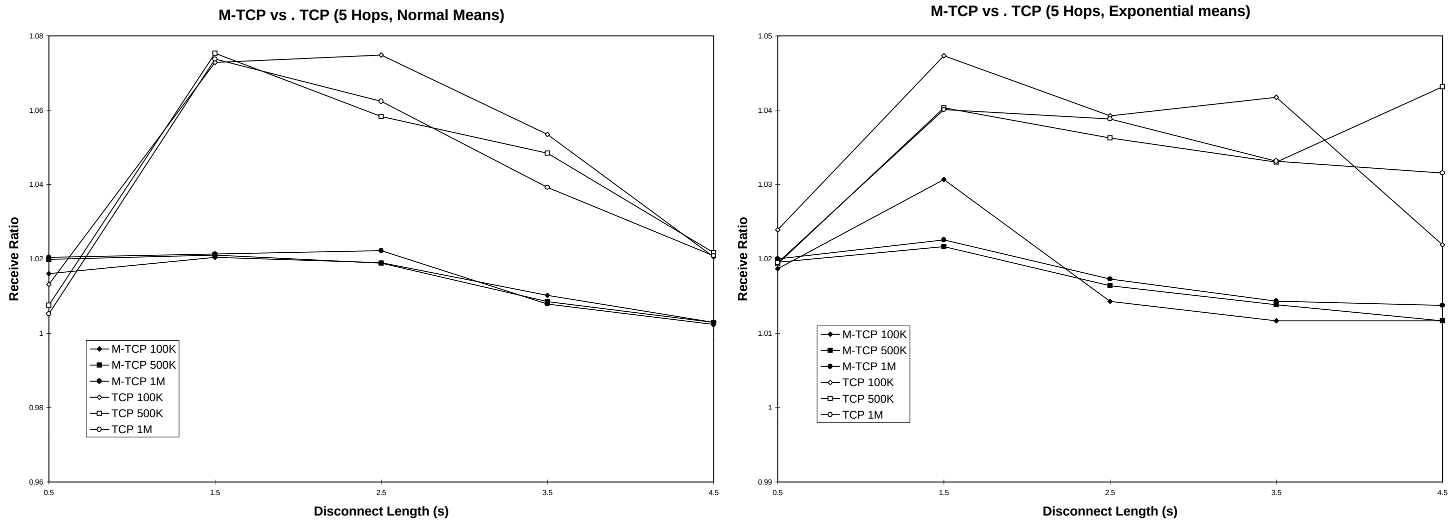

Figure 13: M-TCP and TCP receive rate performance.

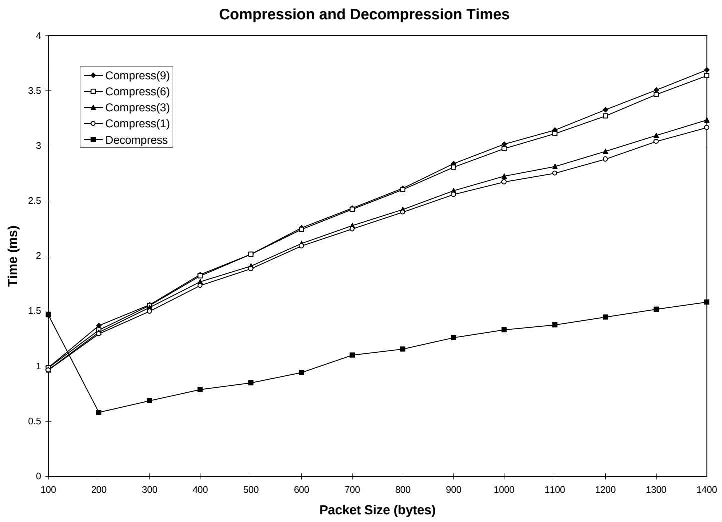

Figure 14: Compression and decompression times in M-TCP.

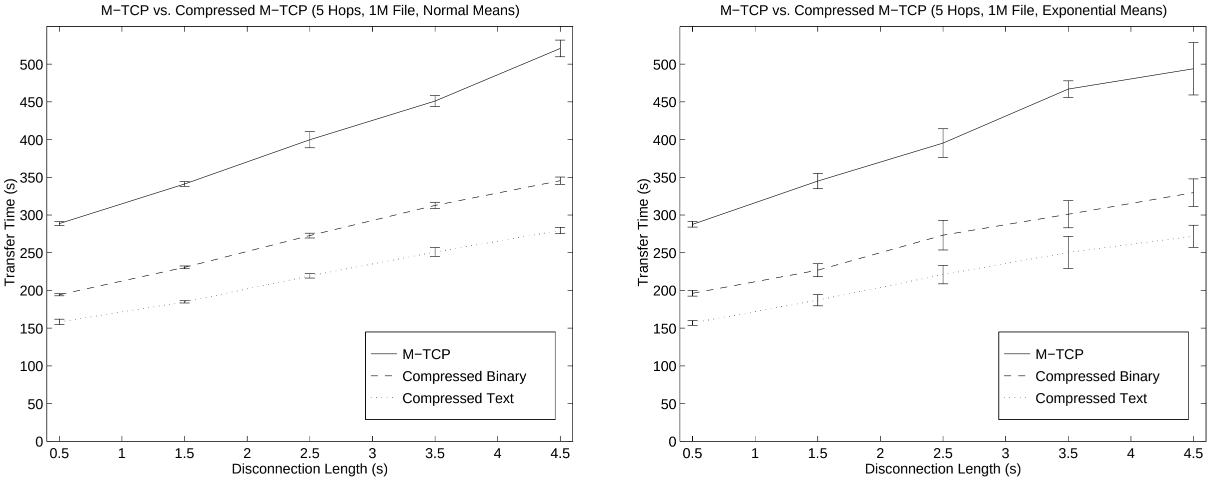

Figure 15: Compressed M-TCP performance.

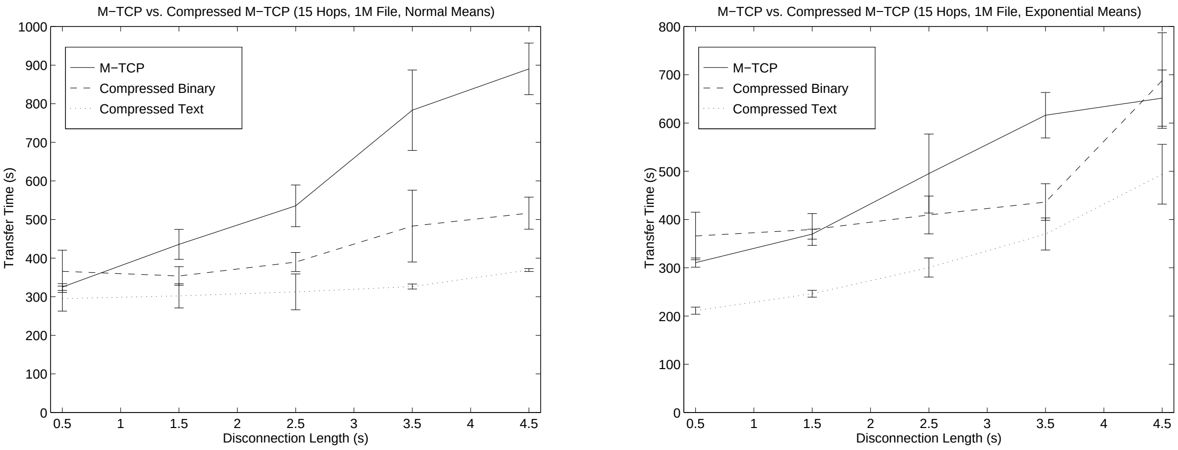

Figure 16: Compressed M-TCP performance.

speed (i.e., we do not perform any link emulation to bring down the speed to 32kbps). The TCP sender send buffer size is varied from 1/2K to 16K and 1M of data is sent for each trial with 50 repetitions per test. In Figure 17 we see that M-TCP takes a much larger penalty hit for small window sizes than TCP. This is because we do extra processing for each window, and the effect is thus cumulative – more windows implies more total delay. Using a larger window size reduces this cumulative effect to a negligible level.

Finally, we measured the delay experienced in transferring files of varying sizes using TCP and M-TCP (again without using the slow link emulation and with no disconnections). The TCP sender used a send buffer of size 16K and was located either 5 hops or 15 hops from the mobile receiver. The results are shown in Figure 18 (the 95% confidence intervals are also shown). In the 5-hop tests, M-TCP tends to be slightly slower than TCP, due to extra processing. In the 15-hop tests, note that M-TCP is statistically better for all 3 file sizes. This is due to the fact that the round-trip times are shorter on the two portions of the connection than on the entire connection, allowing both the TCP sender and M-TCP at the SH to respond more quickly to bandwidth variation and loss.

Based on these results we can conclude that the gateway approach (where we split the TCP connection) is scalable to large mobile networks and does not suffer any noticeable performance hits.

6.5 Performance Summary

Our experimental results clearly show that, in addition to maintaining end-to-end TCP semantics, M-TCP performs very well in environments where the mobile is subjected to frequent disconnections (note that M-TCP will perform

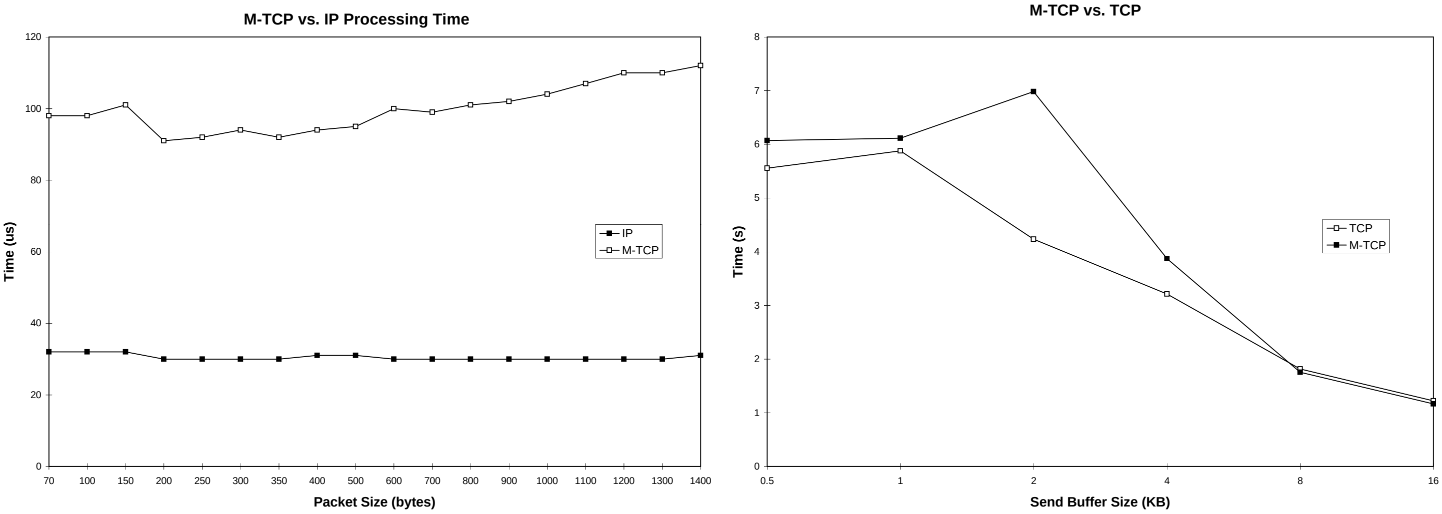

Figure 17: Processing time at the gateway and effect of send buffer sizes.

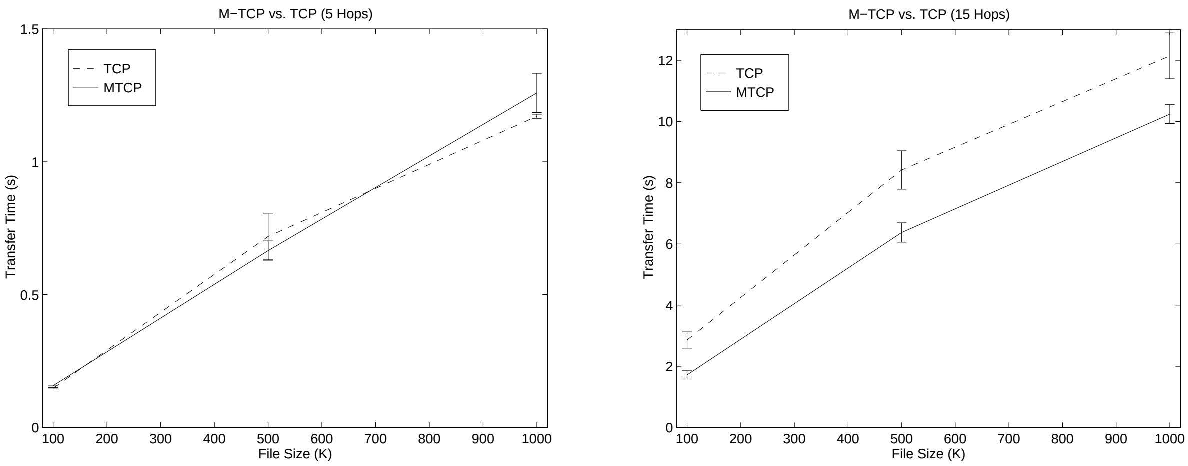

Figure 18: M-TCP vs. TCP performance without slow link emulation or disconnections.

well even in the presence of error-prone wireless channels for the same reason that I-TCP does – local retransmissions of lost segments). Specifically,

1. In the presence of disconnections, M-TCP's file transfer time is less than a third of TCP's file transfer time over 5 hops. Over 15 hops, M-TCP's performance is still much better than TCP's, however, due to a high loss rate over the Internet, the performance gap is smaller.

2. Data compression reduces M-TCP's file transfer time even more. However, the processing overhead at the SH (about 3 msec for a 1400 byte packet) is significant. Thus, compression can be used profitably only in environments where the mobile is in a severely bandwidth-starved cell.

3. The processing time at the gateway nodes is small and does not appear to be a performance bottleneck.

7 Conclusions

In this paper we develop a TCP protocol for use in mobile networks. Our design is based on the observation that, in mobile networks, users will be plagued by frequent disconnection events (caused either by signal fades, lack of bandwidth or by handoff) that must be explicitly handled by the protocol. We also examine the use of data compression for transmission over very low bandwidth links and show that there are significant performance gains to be had. Finally, we answer a frequently leveled criticism of the split connection approach – *viz.,* the gateway will be unable to handle many connections because of the additional processing it needs to do. We show that, in fact, a 100 MHz Pentium PC can adequately handle numerous simultaneous M-TCP connections with little performance degradation.

References

[1] Personal communication with Mike Cunningham. See also `www.metricom.com`.

[2] E. Ayanoglu, S. Paul, T.F. LaPorta, K.K. Sabnani and R.D. Gitlin, "AIRMAIL: A link-layer protocol for wireless networks", *J. Wireless Networks*, Vol. 1, (1995), pp. 47-60.

[3] A. Bakre and B. R. Badrinath,"I-TCP: Indirect TCP for Mobile Hosts", *Proceedings of the 15th International Conference on Distributed Computing Systems, Vancouver, Canada,* June 1995, pp. 136–143.

[4] A. Bakre and B. R. Badrinath,"Handoff and Systems Support for Indirect TCP/IP", *2nd Usenix Symposium on Mobile and Location-Independent Computing,* April 1995.

[5] A. Bakre and B. R. Badrinath,"Indirect Transport Layer Protocols for Mobile Wireless Environment", in Mobile Computing, eds. T. Imielinski and H.F. Korth, Kluwer Academic Publishers, (1996), pp. 229-252.

[6] H. Balakrishnan, S. Seshan, and Randy Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks", *Wireless Networks*, Vol. 1, No. 4, December (1995).

[7] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and Randy Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", *ACM SIGCOMM'96*, pp. 256-269, Palo Alto, CA, August (1996).

[8] B.R. Badrinath and T. Imielinski,"Location Management for Networks with Mobile Users", in Mobile Computing, eds. T. Imielinski and H.F. Korth, Kluwer Academic Publishers, (1996), pp. 129-152.

[9] P. Bhagwat, S. Tripathi and C. Perkins,"Network Layer Mobility: an Architecture and Survey", *Technical Report*, CS-TR-3570, University of Maryland (September 13, 1995).

[10] K. Brown and S. Singh, "A Network Architecture for Mobile Computing", *Proc. IEEE INFOCOMM'96*, S.F. CA, March 1996, pp. 1388-1396.

[11] K. Brown and S. Singh, "M-UDP: UDP for Mobile Networks", *ACM Computer Communication Review*, Vol. 26(5), Pct. 1996, pp. 60-78.

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

[13] S. Cheshire and M. Baker,"Internet Mobility 4×4", *Proceedings ACM SIGCOMM'96*, (Oct. 1996), pp. 318-329.

[14] D. Eckhardt and P. Steenkiste,"Measurement and Analysis of the Error Characteristics of an In-Building Wireless Network", *Proceedings ACM SIGCOMM'96*, (Oct. 1996), pp. 243-254.

[15] R. Ghai and S. Singh, "An Architecture and Communication Protocol for Picocellular Networks", *IEEE Personal Communications Magazine*, Third Quarter 1994, pp. 36-46.

[16] R. Gopalakrishnan, K. Brown and S. Singh, "Transport State Handoff in Mobile Networks", Manuscript.

[17] S. Nanda, R.P. Ejzak and B.T. Doshi,"A retransmission scheme for circuit-mode data on wireless links", *IEEE JSAC*, 1994.

[18] K. Seal and S. Singh,"Loss Profiles: A quality of service measure in mobile computing", *J. Wireless Networks*, Vol. 2 (1996), pp. 45-61.

[19] S. Singh,"Quality of Service Guarantees in Mobile Computing", *Journal Computer Communications*, Vol 19(4), pp. 359-371 (April 1996).

[20] F. Teraoka and M. Tokoro, "Host Migration Transparency in IP Networks: The VIP Approach", *SIGCOMM CCR*, Vol. 23, No. 1, Jan 1993, pp. 45-65.

[21] R. Yavatkar and N. Bhagawat,"Improving End-to-End Performance of TCP over Mobile Internetworks", *IEEE 1994 Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA*, (1994).