Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments

Tom Goff
James Moronski
D. S. Phatak
Electrical Engineering Department
State University of New York, Binghamton, NY 13902-6000
phatak@ee.binghamton.edu

Vipul Gupta
Sun Microsystems Inc.
Mountain View, CA 94043-1100
vipul.gupta@sun.com

(*To be presented at INFOCOM'2000*)

Abstract— **Optimizing TCP (Transport Layer) for mobility has been researched extensively. We present a brief summary of existing results which indicates that most schemes require intermediaries (such as base stations) to monitor the TCP traffic and actively participate in flow control in order to enhance performance. Although these methods simulate end-to-end semantics, they do not comprise true end-to-end signaling. As a result, these techniques are not applicable when the IP payload is encrypted. For instance IPSEC, which is expected to be standard under IPv6, encrypts the entire IP payload making it impossible for intermediaries to monitor TCP traffic unless those entities are part of the security association. In addition, these schemes require changes (in the TCP/IP code) at intermediate nodes making it difficult for the mobile clients to inter-operate with the existing infrastructure.**

In this paper we explore the "Freeze-TCP" mechanism which is a true end-to-end scheme and does not require the involvement of any intermediaries (such as base stations) for flow control. Furthermore, this scheme does not require any changes on the "sender side" or intermediate routers; changes in TCP code are restricted to the mobile client side, making it possible to fully inter-operate with the existing infrastructure. We then outline a method which integrates the best attributes of Freeze-TCP and some existing solutions. Performance results highlight the importance of pro-active action/signaling by the mobile-host. The data indicate that in most cases, simply reacting to disconnections tends to yield lower performance than pro-active mechanisms such as Freeze-TCP.

Keywords— **TCP, Mobile-IP, Wireless networks, Protocol design, implementation, analysis and performance.**

I. Introduction

With explosive growths in wireless services and their subscribers, as well as portable *and affordable* computing devices; it is natural that supporting user mobility in the Internet is a hot and exciting issue that has attracted extensive efforts (for example [1]). As a result of these intense efforts, since its beginnings in early 90s, Mobile-IP has rapidly matured to a stage where it is being proposed as a standard by the IETF [2]. Now that the basic mobile IP protocol is more or less standardized, researchers are beginning to focus on performance enhancing mechanisms at all layers of the networking stack in order to deliver high performance at the end-user level.

TCP is a vital component of the Transport layer of the Internet protocol suite. It is intended to provide connection oriented reliable service over an underlying unreliable network. It is therefore not surprising that TCP has received a lot of attention and fairly large number of researchers have tried to optimize and improve TCP for different environments characterized by heterogeneous subnetworks with widely different bandwidths and latencies (for instance TCP over wireless links, satellite links, slow serial links, etc.).

In the following, we first outline the problems with TCP in mobile environments. Next, we summarize the proposed solutions, indicating their strengths and weaknesses, and the current status of TCP enhancements/modifications being considered for adoption in future versions of TCP by an IETF working group overseeing this area. We then explore the "Freeze-TCP" mechanism to enhance TCP for mobile environments and identify its advantages and drawbacks.

II. TCP's Window Management and Problems in Mobile Environments

TCP uses a sliding window mechanism to accomplish reliable, in-order delivery and flow/congestion control. Figure 1 shows this graphically, with the window sliding towards the right. The window size (W) is determined as the minimum of receiver's advertised buffer space, and the perceived network congestion. The sender allows up to W outstanding or unac-

This work was supported in part by NSF grants CDA-800828 and CDA-9617355

knowledged packets at a time. This results in a “usable window” size equal to W minus the number of outstanding packets.

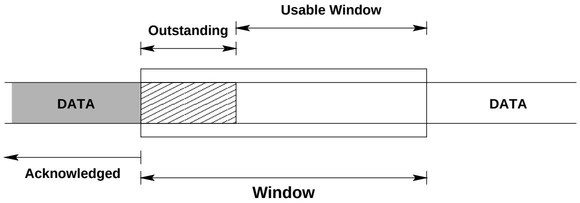

Fig. 1. TCP Window Management

Under normal conditions, the right edge of the window stays fixed (when the packets in the current window remain unacknowledged), or advances to the right along with the left edge of the window, as packets are acknowledged. If the consuming process at the receiver end is slower than the sender, the receiver’s buffers will begin to fill causing it to advertises progressively smaller and smaller window sizes. Eventually the receiver may run out of buffer space in which case it advertises a window size of zero.

Upon seeing an advertised window size of zero, the sender should freeze all re-transmit timers and enter a persist mode. This involves sending probes (called the Zero Window Probes or ZWPs) until the receiver’s window opens up. In a strict sense, each ZWP should contain exactly one byte of data [12] but many TCP implementations including those in Linux and FreeBSD do not include any data in their ZWPs. The interval between successive probes grows exponentially (exponential back-off) until it reaches 1 minute, where it remains constant. Because these probes are not delivered reliably, the sender *does not* drop its congestion window if a Zero Window Probe itself gets lost. Eventually the receiver responds to a ZWP with a non-zero window size, and the sender will continue transmission using a window size consistent with the advertised value.

An exception to this normal window management operation can occur if the receiver “shrinks” its advertised window, that is moves the right edge towards the left. This can suddenly create a negative usable window size which might confuse the sender. While this behavior is discouraged, the sender must recover if it occurs. As stated in [3] and [4], the sender is allowed to retransmit any outstanding packets (up to W), but should not send new data. Also, any lost packets from within the old window (and now to the right of the new window because the right edge moved leftward) should not cause the congestion window to drop. This means that if the receiver shrinks its window to zero, all outstanding packets can be lost without affecting the sender’s congestion window and the sender should enter the persist mode described above.

A. Problems with TCP in mobile environments

TCP was conceived for wired, fixed topologies which are fairly reliable. Hence it operates on the assumption that any losses are due to congestion, which is reasonable for a reliable infrastructure. In mobile environments, however, losses are more often caused by

(i) The inherently higher bit error rates of the wireless links, and

(ii) Temporary disconnections (due to signal fading or other link errors; or because a mobile node moves, etc).

To better illustrate the second item above, it should be noted that mobility is distinct from wireless connectivity. For instance, a user working in the office on a notebook wants to move (with the notebook) to a laboratory or a meeting room at the other end of a building or in the next building, where the IP addresses can be on different subnets; possibly across one or more firewalls. FTP, Telnet sessions and other connections can certainly remain alive for a few minutes it might take to go from one end of a building to another. The idea behind mobility is that such open connections should be retrieved seamlessly despite the move and a change of the underlying IP address.

Even if a single packet is dropped for any reason, the current standard implementation of TCP assumes that the loss was due to congestion and throttles the transmission by bringing the congestion window down to the minimum size. This, coupled with the TCP’s slow-start mechanism means that the sender unnecessarily holds back, slowly growing the transmission rate, even though the receiver often recovers quickly form the temporary, short disconnection. This is illustrated in Figure 2 where it is seen that the network capacity can remain unutilized for a while even after a reconnection.

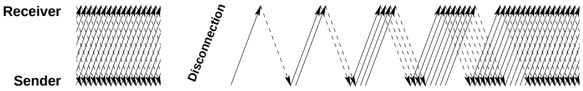

Fig. 2. TCP Slow-Start

III. Existing Solutions

Several approaches have been proposed to overcome these shortcomings of standard TCP. The Berkeley Snoop module [5], [6] resides on an intermediate host (preferably the base station), near the mobile user. It caches packets from the sender and inspects their TCP headers. Using the snooped information, if the module determines that a packet has been lost, it retransmits a buffered copy to the mobile node (which is intended to be a local retransmission over one or a few links). It maintains it’s own timers for retransmission of buffered packets, implements selective retransmissions, etc.

Indirect TCP (I-TCP) [7] proposes to split the connection between a fixed sender host (FS) and mobile host (MH) at a mobility support station (which should ideally be the base station, BS). The data sent to MH is received, buffered and ACKed by BS. It is then the responsibility of BS to deliver the data to MH. On the link between BS and MH, it is not necessary to use TCP. One can use any other protocol optimized for wireless links. MTCP proposed in [8] is similar to I-TCP and also splits a TCP connection into two: one from MH to BS and the other from BS to FH. The MH to BS connection passes through a session layer protocol which can employ a selective repeat protocol (SRP) over the wireless link.

In [9], a method is proposed to alleviate the performance degradation as a result of disconnections due to handoffs. If packets are lost during handoff, the standard TCP at the sender end drops its congestion window and starts a timeout. If this timeout period is longer than the handoff disconnection, the mobile client does not receive any data until the timeout period is over. To reduce this waiting period, [9] makes the mobile host re-transmit 3 copies of the ACK for the last data segment it received prior to the disconnection, immediately after completing the handoff. This causes the sender to immediately re-transmit one segment, which eliminates the waiting period.

In [10] it is proposed to delay the duplicate ACKs for a missing packet (which could trigger a fast retransmission from the sender) in order to allow any special local retransmissions on the wireless links to work, before forcing the sender to fast-retransmit the missing packet(s). In [11] an explicit bad-state notification (EBSN) scheme is presented, wherein, for each failed attempt to send a packet to a MH, the base station (BS) sends an explicit bad-state notification to the sender. Upon the receipt of each EBSN, the sender resets retransmit timer(s) to original value(s). The idea is that these explicit notifications prevent the sender from dropping congestion window (only when the TCP code on the sender side is modified accordingly).

It is possible to exploit TCP's response to a receiver shrinking its window to zero in order to enhance performance in presence of frequent disconnections. The main advantage is that when the sender enters persist mode, it freezes all packet retransmit timers and does not drop the congestion window so that the idle time during the slow-start phase can be avoided. M-TCP proposed in [13] uses this idea. It also splits up the connection between a sender (FH) and mobile receiver (MH) in two parts: one between FH and BS (base station/mobility support station) and one between BS and MH, which uses a customized wireless protocol.

Whenever the base station (BS) detects a disconnection or packet loss, it sends back an ACK to the sender (FH) with a zero window size to force the sender into persist mode, and not drop it's congestion window. To maintain end-to-end semantics, the BS relays ACKs back to the sender only when the receiver (MH) has ACKed data. This can lead to problems: for instance, assume that the sender has transmitted one widow full of packets and is waiting for ACKs. Suppose the receiver receives them all and ACKs the last transmission (TCP ACKs are cumulative) and then immediately gets disconnected. If the BS relays back the ACK to sender, it will keep transmitting eventually leading to packet loss and congestion window throttling. One could send a duplicate ACK for the last segment, advertising a window size of zero, but such duplicate ACKs may be ignored by the sender. Hence, the M-TCP scheme proposes that the base station hold back the ACK to the last byte. For instance, if the MH has ACKed bytes up to and including sequence number X, the BS ACKs bytes only up to $(X-1)$ to the sender. It holds back the ACK for the last byte (X) so that if a disconnection is detected, that ACK can be relayed back to the sender, with a zero window size. Mechanisms to "release" the last-byte-ACK (and the motivations behind them), as well as other details can be found in [13].

A. Strengths and Drawbacks of Existing Solutions

Next we consider major factors (not necessarily in the order of importance) that should be considered in assessing any TCP enhancement scheme.

(1) One of the main considerations is inter-operation with the existing infrastructure. To realize this goal, ideally, there should not be any change required at intermediate routers or the sender because these are likely to belong to other organizations, making them unavailable for modifications. All approaches that split the connection into two parts (this includes all the schemes mentioned in the subsection above, except [10] and [11]) require substantial modification and processing at an intermediate node (BS). Some schemes, such as EBSN [10], also require modifications at the sender side. This makes it difficult for these schemes to inter-operate with the existing infrastructure.

(2) The second important issue is encrypted traffic. As network security is taken more and more seriously, encryption is likely to be adopted very widely. For instance, IPSEC is becoming an integral part of IPv6, the next generation IP protocol. In such cases the whole IP payload is encrypted, so that the intermediate nodes (be it the base station or another router) may not even know that the traffic being carried in the payload is TCP. Any approach (such as SNOOP, I-TCP, MTCP, M-TCP ...) which depends on the base station doing a lot of mediation will fail when the traffic is encrypted.

(3) Even more serious, sometimes data and ACKs can take different paths (for instance, in satellite networks). Schemes based on "intermediary" involvement will have serious problems such a case.

(4) Yet another consideration is maintaining true end-to-end semantics. I-TCP and MTCP do not maintain true end-to-end semantics. M-TCP in [13] does maintain end-to-end semantics, but requires a substantial base-station involvement nonetheless. Thus there is a need for true end-to-end signaling without involving any intermediary.

(5) Even if one assumes that issues (1)–(4) above are not relevant, and that an intermediary (such as a base station) can be brought in for performance enhancements; there is still a need to consider whether the intermediary will become the bottleneck. It is clear that the base stations (BS) in SNOOP, I-TCP, MTCP, M-TCP will all have to buffer at least some amount of data (to perform local retransmission, etc.) and do some extra processing for each connection going through them. If hundreds or thousands of nodes are mobile in the domain of a base station, it could get overwhelmed with the processing of traffic associated with each connection. When a mobile node moves from the domain of one BS to another, the entire "state" of the connection (including any data that was buffered for retransmissions) needs to be handed over to the new base station. This can cause significant amount of overhead and might lead to the loss of some packets and the sender dropping congestion window, which would defeat the original purpose behind the whole endeavor.

On the positive side, if the above issues can be ignored, then most of the proposed solutions (especially M-TCP) do yield performance improvements (although holding back a byte in the

	SNOOP [5], [6]	ITCP [7] & MTCP [8]	M–TCP [13]	Delayed [10] Dupacks	EBSN [11]	Our Freeze-TCP
Requires intermediate node TCP mods?	yes	yes	yes	no	yes	no
Handle encrypted traffic?	no	no	no	yes	no	yes
End-to-end TCP semantics	yes	no	no	yes	no	yes
Handle long disconnections	no	may run out of buffers	yes	no	yes	yes
Frequent disconnections	no	handoff costly	handoff may be costly	no	yes	yes
Handle high BER	yes	yes	yes	no	no	no

TABLE I

Characteristics of various mobile TCP solutions (BER refers to bit-error-rate).

M-TCP scheme might force re-packetization at the sender end, thereby degrading the performance). In [13] it was observed that SNOOP, I-TCP, MTCP handle bit-errors well but do not effectively deal with frequent disconnections of sizable duration or frequent handoffs. The delayed duplicate ACKs scheme [10] was found to improve performance in presence of occasional transmission losses, but it can degrade performance in case of actual congestion losses [11]. Likewise, the explicit bad-state notification (EBSN) scheme works well if the “bad state” lasts for significant duration or when large error bursts occur. However, it may not be as effective as the SNOOP method for random occasional errors [11]. We have summarized the characteristics of some of the proposed solutions in Table I.

A very good summary of current state-of-the-art approaches to optimizing the transport layer for mobile environments can be found in the Internet Draft [14]. In that draft only SNOOP plus SACK is being recommended for adoption, after issues related to IP encrypted payloads (such as those in IPSEC) have been resolved. Some recommendations from the draft to resolve these issues are:

(i) Make the SNOOPing base station a party to the security association between the client and the server, or

(ii) Terminate the IPSEC tunneling mode at the SNOOPing base station.

The draft also recommends adopting delayed dupacks when that technique eventually stabilizes through further research and experimentation. Likewise, the draft recommends only those schemes that require changes at base stations and mobile ends be further researched.

IV. Our Freeze-TCP approach

The main idea behind Freeze-TCP is to move the onus of signaling an impending disconnection to the client. A mobile node can certainly monitor signal strengths in the wireless antennas and detect an impending handoff; and in certain cases, might even be able to predict a temporary disconnection (if the signal strength is fading, for instance). In such a case, it can advertise a zero window size, to force the sender into the ZWP mode and prevent it from dropping its congestion window. As mentioned earlier, even if one of the zero window probes is lost, the sender does not drop the congestion window [12]. To implement this scheme, only the client’s TCP code needs to change and there is no need for an intermediary (no code changes are required at the base station or the sender).

If the receiver can sense an impending disconnection, it should try to send out a few (at least one) acknowledgements, wherein it’s window size is advertised as zero (let an ACK with a zero receiver window size be abbreviated “ZWA”, i.e., Zero Window Advertisement). The question is: how much in advance of the disconnection should the receiver start advertising a window size of zero? This period is in a sense the “warning period” prior to disconnection. Ideally, the warning period should be long enough to ensure that exactly one ZWA gets across to the sender. If the warning period is any longer, the sender will be forced into Zero Window Probe mode prematurely, thereby leading to idle time prior to the disconnection. If the warning period is too small, there might not be enough time for the receiver to send out a ZWA which will cause the sender’s congestion window to drop due to packets lost during the disconnection (which, in turn leads to some idle-time/underutilization after the reconnection).

Given this, a reasonable warning period is the round-trip-time (RTT). During periods of continuous data transfer, this allows the sender to transmit a packet and then receive its acknowledgment. Experimental data corroborates this: warning periods longer or shorter than RTT led to worse average performance in most cases we tested. Note that Freeze-TCP is only useful if a disconnection occurs while data is being transfered (as opposed to when the receiver is idle for some time and then gets disconnected), which is the most interesting case anyway.

Since the ZWPs are exponentially backed off, there is the possibility of substantial idle time after a reconnection. This could happen, for instance, if the disconnection period was long and the the reconnection happened immediately after losing a ZWP from the sender. In that case, the sender will go into a long back-off before sending the next probe. Meantime the receiver has already reconnected, but the connection remains idle until the sender transmits its next probe. To avoid this idle time, we also implement the scheme suggested in [9]. As soon as a connection is re-established, the receiver sends 3 copies of the ACK for the last data segment it received prior to the disconnection. This scheme is henceforth abbreviated as “TR-ACKs” (Triplicate Reconnection ACKs). Note that even in standard TCP, packet retransmissions are exponentially backed off. Therefore the post reconnection idle time can occur there as well. For a fair comparison, the Standard TCP on the receiver side was also modified to optionally send TR-ACKs. This way, the effect of only the Freeze-TCP mechanism (i.e., forcing the sender into ZWP mode *prior* to a disconnection) can be isolated.

Unlike M-TCP, there is no advantage to holding back the ACK to the last byte. For M-TCP it was useful because even when the mobile client was disconnected, the base station could still signal the sender on behalf of the client. In the case of Freeze-TCP, since changes are restricted to the client end, holding back the ACK for the last byte does not help. Note that Freeze-TCP will avoid any re-packetization penalty at the sender end (which M-TCP might incur because it holds back the ACK to the last byte).

Figures 3 and 4 help estimate the performance gain possible due to the Freeze-TCP technique.

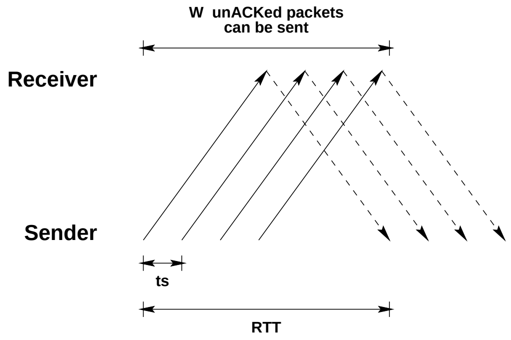

Fig. 3. Relation between t_s, RTT and W

In Figure 3, t_s is the time required to “put or write the packet on the wire”, RTT is the total round trip delay including the t_s delays at sending, receiving as well as any intermediate nodes; and W is the sender’s window. From the figure, it is seen that if any idle periods are to be avoided:

$$W \cdot t_s \geq \text{RTT} \quad \text{or} \quad W \geq \frac{\text{RTT}}{t_s} \tag{1}$$

Since $t_s \approx \frac{\text{packet-size}}{\text{bandwidth}}$ (ignoring processing/queuing delays internal to the host, collisions in case of shared medium, etc.) it is seen that the [delay × bandwidth] product is important in determining how big the congestion window W needs to be if underutilization of network capacity is to be avoided.

$$\text{Assuming} \quad \frac{RTT}{t_s} \gg 1; \quad W \gg 1 \text{ is required for} \tag{2}$$

full network capacity utilization. Figure 4 pictorially illustrates the increased throughput under this condition, when Freeze-TCP prevents sender side window, W, from dropping and regrowing (due to packet losses).

From the figure it can be seen that the (approximate) number of *extra* packets transferred by the Freeze-TCP scheme is given by

$$\text{Extra Segments} = \frac{W^2}{8} + W \lg W - \frac{5W}{4} + 1 \tag{3}$$

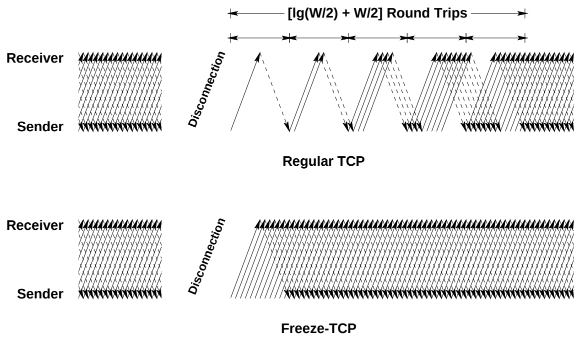

Fig. 4. Illustration of increased throughput due to Freeze-TCP

In addition to (2), the above expression (3) also assumes that upon a disconnection (and the loss of packets), regular TCP drops the congestion window all the way down to 1, and first grows it by a factor of 2 each time an ACK is received, until it reaches $W/2$. From there on, it is incremented by 1 each time an ACK is received until it reaches the same size W prior to disconnection. This congestion window growth mechanism is dubbed “slow-start congestion avoidance” and is adopted in 4.3 BSD Reno release and onwards [12].

It should be noted that (3) is an approximate expression, ignoring collisions, and other factors that might affect the traffic.

A. Experimental Setup

We carried out experiments by modifying the Linux 2.1.101 TCP source code. The receiver side Freeze-TCP mechanism was implemented on a PC designated to emulate the mobile host. All performance measurements used client-server programs specifically written to emulate frequent disconnections during data transfer and measure the resulting data transfer rates and delays.

The server is on the “sender side” without any changes to its underlying TCP code. It runs as a normal user process without any special privileges. The server could be operated in two modes:

(i) Continually send data until the client disconnects, or

(ii) Only send a stream of specified length to the client

In either mode the segment size can be specified by the client. In addition, Each segment has its own serial number included as data, along with a check-sum. This way, the client can easily monitor corrupt or missing packets.

The client runs on the “receiver side” which emulates a mobile node implementing the Freeze-TCP scheme. The client maintains a time ordered list of events to be executed in course of the experiment; such as FreezeOn (start advertising a zero window size), InterfaceOff (simulate a disconnection), FreezeOff, InterfaceOn, Count (which prints the amount of data received since last Count event as well as cumulative time and data bytes since the start of the experiment); etc.

A disconnection is simulated by deliberately corrupting the IP checksum (in the IP header) if a kernel variable is set, in reality this could be easily linked to signal strengths measured by the antenna(s). This way, incoming packets never reach the mobile node's TCP layer; and any TCP packets that are sent out by the mobile node are also dropped by peer IP layers; very closely simulating a disconnection (from the TCP layer's perspective). The InterfaceOff and On events simply set and unset this flag. This mechanism is general and independent of the specific connection type (Ethernet, SLIP or whatever) as well as the network interface card.

Between FreezeOn and InterfaceOff events (i.e., in the warning interval prior to disconnection) data segments from the sender are accepted as usual, but the receiver "window size" is advertised as zero in all acknowledgements sent out during this period. This accommodates some of the overflow that happens even after the sender has seen a ZWA. Such overflow was found to occur in all TCP implementations except in FreeBSD and is caused by

(i) Packets already in flight which the receiver gets *after* transmitting the ZWA, and

(ii) The sender getting confused (in most implementations) because of the sudden window shrinkage which in turn causes it's usable window to become negative. In this situation, as per RFC's [3] and [4], the sender is allowed to empty out the packets in the previous non-negative sized window.

The client can be operated in two modes:

(i) Send a triplicate ACK (TR-ACK) after a reconnection, i.e., immediately after every InterfaceOn event, and

(ii) "No TR-ACK" or standard mode, where the receiver does *not* send extra ACKs after a reconnection, it just waits for the next segment from the sender.

The functions required to implement the FreezeOn/Off, InterfaceOn/Off and TR-ACK events were added to the TCP code in the kernel and made available to the client program as system calls. At the start, the client program reads all the events along with their execution times as well as the desired operational modes (TR-ACK vs. No TR-ACK, fixed data length or fixed time duration, etc.) from an input file. This gives a lot of flexibility to explore different warning periods, operational modes, etc., by simply changing the input file. Note that the warning periods (equal to RTT) were evaluated for each connection and operational mode before running the client program.

B. Experimental Data

As mentioned above, the performance of the Freeze-TCP scheme depends on the RTT/t_s ratio which is proportional to [delay × bandwidth]. The higher this ratio, the more effective this mechanism can be. To illustrate these trends we consider the following environments which span a large range and variety of bandwidths: 10 Mbps Ethernet, 100 Mbps Ethernet, and 38.4 Kbps PPP. To vary the round-trip delay, the 10 Mbps Ethernet and 38.4 Kbps PPP experiments were run against local (same room) as well as distant (across the county) servers. The RTT/t_s ratios for each case are listed below.

Local Media	Server	RTT/t_s
10 Mbps Ethernet	local	2
10 Mbps Ethernet	distant	63
100 Mbps Ethernet	local	4
38.4 Kbps PPP	local	0.25
38.4 Kbps PPP	distant	0.3

The 10 Mbps Ethernet experiments were performed to emulate a mobile host in a wireless Ethernet cell, where, under light load conditions, the bandwidth available can be comparable to wired Ethernet. In addition, 10 Mbps Ethernet is the most commonly used local network technology. Since a local network segment is highly "controllable", it possible to guarantee any desirable set of traffic patterns. Hence, experiments on the local networks serve to quickly bring out the intrinsic trends without interference from true congestion and other unpredictable events found in the Internet at large. Moreover, it is likely that mobile nodes do most of their data communication with a server that is likely to be the base station and servers attached to it (most likely supplied by the ISP). Hence it is useful to look at the performance across a small number of 10 Mbps Ethernet hops, which is the first set of experiments conducted.

In this set we performed experiments with several servers and present the results for a local host running Solaris 2.6 and a distant host running AIX 4.2. Each experiment was repeated 10 times with base and Freeze-TCP cases running alternately, so that for every bases case run, the corresponding Freeze-TCP case happened under almost identical conditions (time of day, network traffic, etc.). The results are shown in Tables II and III.

Table II shows that the Freeze-TCP scheme enhances performance substantially. Besides showing the overall gain [i.e., the performance of Freeze-TCP along with TR-ACKs as compared with the plain/standard TCP implementations found on all Unix flavors today], we have also evaluated the performance enhancements (± percentage improvements) for the TR-ACK and No TR-ACK case separately. In other words, standard TCP with TR-ACK is compared with Freeze-TCP with TR-ACK (in column 3), and Standard TCP without TR-ACK (which is what is found in all current implementations) is compared with Freeze-TCP without the TR-ACK (in column 5). This helps isolate the performance enhancements due to the Freeze-TCP mechanism (i.e., putting the sender into the Zero Window Probe mode prior to a disconnection) from that due to the TR-ACKs. The data demonstrates that the Freeze-TCP mechanism by itself can yield a sizable performance enhancement. In addition, it can be used along with other techniques (such as TR-ACKs), to further enhance the gain.

Table III shows the results obtained in experiments with a very remote host: 17 hops, and geographically across the country. Once again, it is seen that Freeze-TCP leads to better performance in almost all cases. Here, there is more variability in the data which is attributable to the unpredictable traffic conditions in the Internet at large. However, the main trends are same as those seen in Table II. Notice that in most cases using Freeze-TCP leads to substantially higher performance. Here too, the data from the No TR-ACKs cases showed that Freeze-TCP with TR-ACKs leads to a substantial improvement over standard TCP (which does not have TR-ACKs), as in Table II. However, that

Disconnect time	Transfer time (Sec), averaged over 10 runs (100 disconnection events)				Overall Gain (Freeze-TCP + TR-Acks over Normal TCP)
	Triplicate Reconnection (TR)-Acks		No TR-Acks		
	Standard TCP	Freeze-TCP	Standard TCP	Freeze-TCP	
2.6 ms	18.7	13.0 (+ 30.4%)	18.6	17.1 (+ 8.1%)	[+ 30.1%]
30 ms	17.9	13.2 (+ 26.2%)	17.8	16.9 (+ 5.4%)	[+ 25.8%]
0.1 s	18.4	13.8 (+ 25.2%)	18.7	17.5 (+ 6.3%)	[+ 26.2%]
0.5 s	19.4	17.3 (+ 10.9%)	21.2	21.8 (– 3.2%)	[+ 18.4%]
1 s	25.7	22.4 (+ 12.7%)	28.2	28.6 (– 1.5%)	[+ 20.6%]
2 s	40.0	32.5 (+ 16.6%)	66.1	66.6 (– 0.8%)	[+ 50.8%]
5 s	71.0	63.6 (+ 10.4%)	116.3	95.0 (+ 18.4%)	[+ 45.3%]
10 s	143.8	116.6 (+ 18.9%)	190.6	184.3 (+ 3.3%)	[+ 38.8%]

TABLE II

Local host (3 hops). Number of disconnections per run = 10. Interval between events = 1 s.
Data stream size = 10 MBytes. Effective throughput of 10 Mbps Ethernet ≈ 1 MBytes/sec.
Warning Period = 2.6 ms ≈ RTT for 1000 byte packets.

Disconnect time	Disconnections per run/event	Data Stream Length (Bytes) repetition interval	Transfer time (Sec), averaged over 10 runs	
			Triplicate Reconnection (TR)-Acks	
			Standard TCP	Freeze-TCP
2.6 ms	8	10 M	103.4	88.9 (+ 14.1%)
30 ms	8	10 M	114.2	90.8 (+ 20.6%)
0.1 s	8	10 M	109.3	90.1 (+ 17.6%)
0.5 s	8	10 M	109.9	92.2 (+ 16.1%)
1 s	8	10 M	113.0	96.6 (+ 14.5%)
2 s	8	10 M	116.1	106.8 (+ 8.10%)
5 s	8	10 M	140.4	134.8 (+ 3.99%)

TABLE III

Remote host (17 hops, across the country). Effective throughput of 10 Mbps Ethernet ≈ 10 KBytes/sec.
Warning Period = 85 ms ≈ RTT for 1000 byte packets.

Disconnect time	Number of Disconnections Per run	Interval	Data Stream Length (Bytes)	Transfer time (Sec), averaged over 10 runs Triplicate Reconnection (TR)-Acks	
				Standard TCP	Freeze-TCP
0.4 ms	11	75 ms	15 M	2.35	1.45 (+ 38.4%)
1 ms	8	75 ms	15 M	23.5	14.5 (+ 38.1%)
10 ms	10	50 ms	10 M	1.98	1.13 (+ 43.3%)
0.1 s	8	160 ms	15 M	3.40	2.30 (+ 32.4%)
0.4 s	10	160 ms	20 M	8.81	6.12 (+ 30.6%)
1 s	8	160 ms	20 M	13.1	9.96 (+ 23.9%)
2 s	8	160 ms	20 M	26.3	17.9 (+ 31.8%)

TABLE IV

Local host (2 hops). Effective throughput of 100 Mbps Ethernet ≈ 10 MBytes/sec.
Warning Period = 0.3 ms ≈ RTT for 1000 byte packets.

data does not lead to new insights and hence, it has been excluded from the table for the sake of clarity and brevity. In fact the remaining data (which is presented in the table) is the most pessimistic because it compares Freeze-TCP using TR-ACKs with standard TCP using TR-ACKs, thereby isolating the enhancements due to the Freeze-TCP mechanism alone.

The experiments with remote hosts clearly demonstrate the inter-operability of Freeze-TCP with the existing infrastructure: the server can be anywhere; all the changes to TCP code are confined to the client which was under our control (none of the remote servers were under our control, we only had access to normal user accounts there, to run the TCP server application as an ordinary user process).

As both tables show, in the cases where Freeze-TCP does not give better results, the loss is very small: (no more than 3.2%). This is the main advantage of Freeze-TCP: most of the time it leads to better performance and the enhancement can be substantial, while the few times it leads to worse results, the loss is marginal.

To illustrate how the results vary with bandwidth, the next set of experiments were done on a local 100 Mbps Ethernet segment. Because of the bandwidth (10 times that of normal Ethernet), the interval between successive events was lowered by a factor of about 10 (otherwise the whole data stream would quickly get transferred in one of the continuous periods between disconnections, prematurely ending the experiments). As expected, Table IV shows that the benefits of using Freeze-TCP are significant in high bandwidth environments. Also, the 100 Mbps TR-ACKs results illustrate better improvements on average than the 10 Mbps TR-ACKs data. The substantial performance gains are facilitated by the high bandwidth, which leads to a higher [delay × bandwidth] product (and higher RTT/t_s ratio). Since the [delay × bandwidth] product required to "fill the pipe" is greater than that of 10 Mbps Ethernet, higher window sizes are required to saturate the network. Consequently, the gain that can be realized when such a large congestion window is prevented from dropping is also higher. This is consistent with equation (3) which shows that the gain grows (essentially) quadratically with the window size W.

The final set of experiments emulate a mobile client connecting via a wireless modem; resulting in data rates that are much lower than wireless Ethernet LANs. To simulate such low link speeds, we connected the receiver (emulating the mobile client) to a router with a serial link using the PPP protocol. The serial link makes it possible to operate at any desired speed. The results are shown in Tables V and VI.

As the tables show, the current implementation of Freeze-TCP can lead to performance enhancement in several cases. It rarely performs significantly worse than normal TCP, although it does appear to yield almost similar performance in many cases. This happens due to the following reasons:

(1) The RTT/t_s ratio is low to begin with (an order of magnitude lower than 10 Mbps Ethernet as seen in the list at the beginning of this section). In other words, the network can be saturated fairly quickly, with small sized windows. Consequently, the gain that can accrue from preventing the congestion window from falling is not as high as Ethernet environments where the ratio is much higher.

(2) Due to the large bandwidth mismatch between Ethernet and PPP, the intermediate router at the other end of the PPP link drops packets as its buffers become full. This was observed by running the `tcpdump` utility on a third neutral observer machine on the Ethernet side of the router while the experiments were in progress. These periodic packet drops cause the sender to drop it's congestion window, irrespective of what the receiver does. Hence, using the Freeze-TCP mechanism to prevent the window from falling during disconnections is not very effective since it periodically falls anyway (because the intermediate router drops packets).

We optimized the kernel on the router to enhance it's routing performance, but that did not lead to a noticeable change in the overall performance. Then we kept reducing the maximum window size which the receiver advertises. The idea is that with small window sizes, the number of outstanding packets allowed is small, thereby reducing the likelihood of buffer-overflow at the intermediate router. It turns out that the window size (W) required to bring down the router packet drops to a couple of segments once in a while is so small that preventing the window from falling (by using Freeze-TCP) is not effective.

The important point of all the data is that even when Freeze-TCP is not effective, it does not worsen performance by a noticeable amount. The few cases in which it loses, the losses are marginal. This indicates that the overhead due to the Freeze-TCP mechanism is very small: even if it fails to enhance performance, it will at least render baseline (standard TCP like) performance. *This is a win, no-loss situation!*

Finally we would like to point out that for all the data presented, the number of disconnections was kept equal in both base and Freeze-TCP cases, even though the base case typically takes a longer time to execute than the corresponding Freeze-TCP case. In reality, connections with longer transfer times are likely to suffer more disconnections. This means the data is highly pessimistic and illustrates a worst-case scenario.

V. Discussion and Conclusion

We illustrated the Freeze-TCP mechanism to enhance TCP throughput in the presence of frequent disconnections (and reconnections) which characterize mobile environments. It is a true end-to-end signaling scheme and does not require any intermediaries (such as base stations) to participate in the flow control. Furthermore, it does not require any changes to TCP code on the sending side. Changes to TCP code are confined entirely to the receiver side and are easy to implement. It exploits existing mechanisms already provided in the TCP/IP protocols (such as receiver's window size, SACK, etc.) to enhance TCP's performance. Hence there is little or no overhead of any kind. Thus any performance gains are almost "free" (almost no overheads or tradeoffs). Even more important, complete inter-operability with the existing infrastructure is guaranteed.

In order to exploit this mechanism, the network stack needs to be aware of mobility (at least to some extent). For instance, the NIC vendors need to provide some details of their roaming and

Disconnect time	Transfer time (Sec), averaged over 10 runs (120 disconnection events)				Overall Gain
	Triplicate Reconnection (TR)-Acks		No TR-Acks		(Freeze-TCP + TR-Acks
	Standard TCP	Freeze-TCP	Standard TCP	Freeze-TCP	over Normal TCP)
2.6 ms	185.7	178.8 (+ 3.7%)	183.1	179.7 (+ 1.9%)	[+ 2.35%]
30 ms	185.2	181.5 (+ 2.0%)	181.8	177.4 (+ 2.4%)	[+ 0.17%]
0.1 s	190.2	176.7 (+ 7.1%)	178.2	183.0 (– 2.6%)	[+ 0.84%]
0.5 s	178.3	179.3 (– 0.6%)	188.4	189.2 (– 0.4%)	[+ 4.83%]
1 s	196.0	193.7 (+ 1.1%)	198.2	193.7 (+ 2.2%)	[+ 2.27%]
2 s	206.9	208.3 (– 0.7%)	234.4	217.5 (+ 7.2%)	[+ 11.1%]
5 s	243.5	240.1 (+ 1.4%)			

TABLE V
Low Bandwidth data: Local host (3 hops). Number of Disconnections per run = 12.
Interval between successive events = 10 s. Data stream size = 500 KBytes.
Effective throughput of 38.4 Kbps PPP ≈ 4800 Bytes/sec. Warning period = 650 ms ≈ RTT for 1000 byte packets.

Disconnect time	Transfer time (Sec), averaged over 10 runs (100 disconnection events)				Overall Gain
	Triplicate Reconnection (TR)-Acks		No TR-Acks		(Freeze-TCP + TR-Acks
	Standard TCP	Freeze-TCP	Standard TCP	Freeze-TCP	over Normal TCP)
2.6 ms	76.9	75.0 (+ 2.5%)	73.4	78.7 (– 7.3%)	[– 2.18%]
30 ms	81.1	76.8 (+ 5.2%)	74.7	75.6 (– 1.2%)	[– 2.81%]
0.1 s	78.6	76.1 (+ 3.1%)	77.2	80.2 (– 3.8%)	[+ 1.42%]
0.5 s	87.3	86.2 (+ 1.2%)	92.8	93.5 (– 0.7%)	[+ 7.11%]
1 s	95.5	97.9 (– 2.5%)	106.6	103.7 (+ 2.7%)	[+ 8.16%]
2 s	111.4	110.5 (+ 0.8%)	124.0	114.1 (+ 8.0%)	[+ 10.9%]
5 s	153.9	135.8 (+ 11.7 %)	182.9	178.3 (+ 2.5%)	[+ 25.8%]

TABLE VI
Remote host (24 hops, across the country). Number of Disconnections per run = 10.
Interval between successive events = 5 s. Data stream size = 200 KBytes.
Effective throughput of 38.4 Kbps PPP ≈ 3200 Bytes/sec. Warning period = 780 ms ≈ RTT for 1000 byte packets.

handoff algorithms to TCP code developers. In essence, some cross-layer (layers of the networking stack) efforts and information exchanges are needed, which may be a drawback.

At a more fundamental level, the question is whether it is appropriate to restart transmission at the full rate with the old window size upon entering a new, unknown environment? TCP itself implements "slow-start" at the beginning of a connection precisely because it wants to sample the congestion state before it decides to increase the transmission rate. While this is true in general, there are many situations in mobile environments where the disconnections are not caused by any fundamental or substantial changes in the traffic or congestion state of the network. This can happen for instance, due to temporary obstructions or fades that the mobile client may experience. When the temporary obstruction disappears, the mobile client is reconnected in the same cell where the traffic pattern is very likely to be the same as before. In such cases, it should be safe to resume transmission with the same window size as prior to the disconnection. Another possible drawback of Freeze-TCP is that it needs the receiver to predict impending disconnections. However, if a disconnection cannot be predicted the behavior and performance will be exactly that of standard TCP. Simulation results highlight the importance of *pro-active* action/signaling by the mobile-host. The data indicates that in many cases, a *pro-active* mechanisms such as Freeze-TCP can yield better performance than those that simply *react after* a disconnection occurs.

In closing, we would like to point out that true end-to-end techniques will become indispensable when IPSEC or other security mechanisms are employed to encrypt the IP payloads. Drastic changes to protocols could be required in the future since the Internet was not conceived to support mobility, security, Quality of Service (QoS), etc. Trying to incorporate these attributes is therefore a continual "retrofit" job which will perhaps not fix everything. However, as with any retrofit, backward-compatibility and inter-operability with existing infrastructures are of utmost importance and the proposed technique satisfies these criteria.

REFERENCES

[1] Gupta, Vipul, Linux-Mobile-IP developed at SUNY, Binghamton. This was the first implementation available for Linux and has been publicly released on the Web since Aug. 1995 onwards, `http://anchor.cs.binghamton.edu/˜mobileip`.

[2] Perkins, C., RFC 2002: IP Mobility Support, October 1996.

[3] Postel, J. eds, RFC 793: Transmission Control Protocol, September 1981.

[4] Braden, R. eds, RFC 1122: Requirements for Internet Hosts – Communication Layers, October 1989.

[5] H. Balakrishnan, V. N. Padmanabhan, and R. Katz, "Improving Reliable

Transport and Handoff Performance in Cellular Wireless Networks," *Wireless Networks*, vol. 1, no. 4, Dec. 1995.

[6] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. Katz, "A Comparison of Mechanisms for Improving TCP performance over wireless links," in *Proceedings of ACM SIGCOMM'96, Palo Alto, CA*, Aug 1996, pp. 256–269.

[7] Ajay Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," Tech. Rep., Rutgers University, May 1995, `http://www.cs.rutgers.edu/~badri/journal/contents11.html`.

[8] Raj Yavatkar and Namrata Bhagawat, "Improving end-to-end performance of tcp over mobile internetworks," in *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, Dec. 1994, `http://snapple.cs.washington.edu/mobile/mcsa94.html`.

[9] Ramòn Càceres and Liviu Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE JSAC Special Issue on Mobile Computing Network*, 1994, `http://www.cs.rutgers.edu/~badri/journal/contents11.html`.

[10] M. Mehta and N. H. Vaidya, "Delayed duplicate acknowledgments: A proposal to improve performance of tcp on wireless links," Tech. Rep., Texas A&M University, Dec. 1997, `http://www.cs.tamu.edu/faculty/vaidya/Vaidya-mobile.html`.

[11] N. Vaidya, Overview of work in mobile-computing (transparencies), `http://http://www.cs.tamu.edu/faculty/vaidya/slides.ps`.

[12] W. Richard Stevens, *TCP/IP Illustrated, Volume 1*, Addison Wesley, 1994.

[13] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks," *ACM Computer Communications Review (CCR)*, vol. 27, no. 5, 1997.

[14] G. Montenegro and S. Dawkins, "Wireless Networking for the MNCRS, Internet Draft, work in progress," Aug. 1998, `http://www.ietf.org/internet-drafts/draft-montenegro-mncrs-00.txt`.