Improving TCP Performance in Mobile Computing Environments

Michael Stangel   Vaduvur Bharghavan
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
stangel@uiuc.edu   bharghav@crhc.uiuc.edu

ABSTRACT: **In this paper we examine several aspects of performance and connection management during handoffs and interface switching in mobile computing environments. We demonstrate the insufficiency of network layer mobility protocols to preserve high-performance operation of reliable transport protocols such as TCP under certain circumstances. We also introduce a mobility-aware transport, Mobile TCP, that maintains robust operation despite lossy mobile activity. Experimental results are presented and discussed.**

1. Introduction

The de facto TCP/IP protocol standard is incapable of supporting host mobility in the Internet [12]. This is because the IP address of a host plays the dual role of both end-point identification and location identification. While the former role is used by TCP for connection management (a TCP connection is uniquely identified by the 4-tuple <`src addr, src port, dst addr, dst port`>), the latter role is used by IP for routing packets to the host. Thus, when a mobile host changes its point of attachment to the network, its address must change in order to route packets to it successfully. The evolution of TCP/IP networks did not consider the problem of mobility until recently; the TCP connection identifier is defined during connection establishment, and is not expected to change during the lifetime of the connection. Thus, special mechanisms are needed to support mobility while still allowing ongoing TCP connections to transmit and receive packets transparently during and after handoff. In recent years, a number of research efforts have focused on providing network layer support for mobility, resulting in the development of the Mobile IP standard for both IPv4 and IPv6 (see [2] for a survey). The basic approach of all of these proposals has been to provide a two-layer addressing scheme for a mobile host. A mobile host has an invariant *home address* that is used by higher layer protocols such as TCP for addressing and connection management, and a location dependent *care-of address* that is used by IP for routing. The mapping between the home address and the care-of address is done by means of *address translation agents* that may reside in the home network of the mobile host (the network identified by its home address) [11], or may be distributed throughout the network [13], depending on the mobility support protocol. At any rate, the common feature of almost every existing approach is to shield transport layer protocols from host mobility.

While hiding mobility from transport layers enables connection-oriented protocols such as TCP to operate unaltered, it has serious implications on performance, as described in this paper and also in related work [1, 4]. In particular, TCP attributes all packet losses to congestion; thus, when packets are lost during host mobility, TCP mistakenly reacts to these transient losses by throttling its sending rate. Recovering from this unnecessary congestion control can take several seconds, and causes perceptible degradation in the performance of TCP. The fundamental purpose of this paper is to explore the effect of mobility on TCP and to propose mechanisms that enable TCP to distinguish the cause of packet losses and react accordingly. We also propose mobility-aware connection management in TCP not only for improving performance, but also to deliver additional functionality such as supporting seamless interface switching among different network providers. As we show through performance measurements, our approach significantly improves the performance of TCP in a mobile computing environment.

The rest of this paper is organized as follows. Section 2 discusses related work and the motivation for this paper. Section 3 describes the design and implementation of Mobile TCP, our modified mobility aware TCP protocol. Section 4 describes the performance of our protocol and compares it to a mobility unaware TCP that operates over Mobile IP. Section 5 summarizes the results of this paper and proposes directions for future work.

2. Related Work and Motivation

In this section, we first describe related work, and then motivate the design of Mobile TCP.

2.1. Related Work

As described in Section 1, Mobile IP provides a two layer addressing scheme for mobile hosts, which provides invariant connection end-points for TCP (home address) while enabling Mobile IP to route packets to a mobile host at its current point of attachment (care-of address). Correspondent hosts address packets to the home address of a mobile host. A home agent that resides in the home network of a mobile host intercepts packets addressed to the mobile host and tunnels them to the mobile's current care-of address [11]. As long as the home address remains unchanged, the correspondent host's protocol stack and the mobile host's transport protocol remain unaware of the IP endpoint changes that may occur during handoffs and interface switching.

As a result of the above approach, TCP is unaware of mobility and attributes all packet losses to network congestion. Before we describe the effect of mobility on TCP, we briefly review the congestion control mechanisms in TCP. The sender of the data in TCP performs congestion control using feedback from the receiver in the form of cumulative acknowledgements. At any point, a *congestion window* (`cwnd`) determines how many unacknowledged bytes the sender may pump into the network. There are four basic mechanisms to manage `cwnd`: (a) *slow start* to exponentially ramp up `cwnd` during startup in order to quickly reach half the estimated steady state value (identified by `ssthresh`), (b) congestion avoidance with *linear window increase* beyond the threshold value of `ssthresh`, (c) *fast recovery* (accompanying a fast retransmit), during which cwnd and ssthresh are approximately halved (exponential decrease), and (d) reset cwnd upon a timeout waiting for an acknowledgement. These mechanisms are well documented in literature [7, 8, 12]. Note that a reset is followed by a slow start, and also that fast recovery essentially halves the congestion window and causes a linear increase back to a stable value[1].

Mobility impacts the above congestion control mechanisms in some subtle ways. Consider data transmission from a stationary correspondent host to a mobile host (called a downlink transmission). Upon handoff, some data packets are typically lost (between the handoff and the notification of this handoff to the home agent) but the acknowledgment packets are not. If the transmitting host supports fast retransmits, three consecutive duplicate acknowledgments from the mobile host will induce the correspondent host to retransmit the lost packet and begin fast recovery. Successive packet losses may trigger fast recovery multiple times, thus causing the sender to severely throttle its cwnd and then recover *linearly* to its original value [6]. If the transmitting host does not support fast retransmits, it must wait for a timeout period that is commensurate with the round-trip time of the connection before detecting the packet loss, which then triggers retransmission and slow start. In either case, successive lost packets result in a very small congestion window and an equally small slow start threshold, thus triggering linear recovery from a severely throttled channel.

Consider data transmission from a mobile host to a stationary correspondent host (called an uplink transmission). Upon handoff, some acknowledgement packets are typically lost though the data packets are not. The loss of several ACKs can result in an artificial inflation of the retransmission timer, but in general the connection is nicely restored after one or two acknowledgments are delivered, unless the loss of the acknowlegments causes a timeout. This is largely due to the cumulative nature of ACKs, meaning that any lost ACKs need not be retransmitted. If the lost ACKs do result in a retransmission timeout, the mobile host will reduce its congestion window to one and return to slow-start. This degrades performance, but is expected to happen no more than once per handoff.

From the above discussion, it is clear that TCP needs to distinguish the causes of packet loss and react accordingly; in particular, charging every packet loss to congestion is unreasonable in a mobile computing environment, where packet losses may occur as a result of wireless channel errors, handoffs, interface switches, or congestion. Several recent research papers have addressed these issues. For example, snoop protocols have been proposed to handle wireless channel error [1]. Multicasting support has been proposed to reduce packet losses due to mis-routing [9]. More relevant to this paper, the work of Caceres et al. has identified several of the issues raised above and proposed the following solutions [4]. First, the authors of [4] note that the 500ms granularity of the TCP timers causes unnecessary delay in retransmission timeout, and suggest finer granularity timers. However, increasing the resolution of TCP's timers will also result in greater sensitivity to genuine variations in round-trip time. In particular, loss of acknowledgements during uplink transmissions may now cause the same problems associated with downlink transmissions.

Second, [4] proposes using fast retransmit whenever a handoff is detected by the mobile host. The advantage of this approach is that TCP at the correspondent host need not be modified to support mobility—the mobile host can simply transmit three successive duplicate acknowledgements upon detecting the handoff (or interface switch). This approach also has several problems: it does not prevent the unnecessary congestion control associated with triggering fast recovery multiple times, it unnecessarily throttles uplink connections that may not have lost any data, and finally, it does not account for triangular routing—it is possible that the notification of handoff takes longer to reach the home agent than the duplicate acknowledgements take to reach the correspondent host, thus the retransmitted packets may again be misrouted. The fundamental problem is that the above mechanisms still cause TCP to throttle its sending rate upon packet losses that occur as a result of mobility. As networks increase available bandwidth and applications grow more demanding, it will become critical to avoid the transmission backoff that accompanies TCP's congestion avoidance during mobile transitions.

2.2. Motivation for this work

The motivation for this work is based on both the observed limitations of related work above, and the desire to provide additional functionality to address the requirements of future mobile computing environments. We first discuss the latter, and then revisit the former to motivate our work.

Our experience clearly demonstrates the need to provide mobility management at the transport layer for several reasons. First, mechanisms that depend on Mobile IP for transparent mobility management will necessarily have to work within the constraints of triangular routing. The congestion window of a TCP connection will therefore be affected if there is congestion in the region of the home agent, even if there is no congestion in the path between the correspondent host and the mobile host. Furthermore, both the round trip time and the variation in round trip time of a connection will increase. IPv6 alleviates this problem of triangular routing by enabling a mobile host to notify a correspondent host upon change of its care-of address. Second, as the number of overlapping wireless network providers increase, it will be possible, and even desirable, to dynamically switch between network interfaces and service providers in order to overcome congestion, fades, and bursty channel errors (similar to on-the-fly channel hopping in cellular telephones). Since different network providers will typically assign different home addresses for administrative and accounting reasons, Mobile IP will not be able to support such interface switching functionality. Of course, there is no reason why a transport connection cannot span multiple network connections with different end-point addresses—the requirement of a single endpoint network address for a transport connection is simply an artifact of the design of TCP/IP. Thus, we propose to provide connection management in Mobile TCP in order to handle both the problems with triangular routing and the ability to switch interfaces with different home addresses.

A handoff between neighboring base stations of the same network may have quite different characteristics than an interface switch to a new network (e.g. Ethernet to WaveLAN). When shutting down a previously-used interface, some packets may be lost if they have been queued by TCP but not yet transmitted on the interface. This loss will depend on the existence and size of transmission buffers in the network layer and the hardware itself. Furthermore, a new network may have different delay and bandwidth

[1]This paper deals with TCP-Reno. Improvements to TCP-Tahoe are slightly different, and are the focus of ongoing work.

characteristics, causing the existing estimates for round-trip time, congestion window and slow-start threshold to be invalid. Finally, since base stations of neighboring or even overlapping cells of different networks are unlikely to be physically close to each other (e.g. a switch from CDPD to RAM), schemes such as clustering, hierarchical handoff and base station multicasting are unlikely to be viable in domain-switching scenarios. It is important to keep the above requirements in mind when designing both connection management and performance improvement techniques at the transport layer.

In order to improve the performance of TCP during and after a handoff or upon interface switching, we have already alluded to the importance of distinguishing between the various forms of packet loss. *When packets are lost due to a handoff or interface switch, retransmission should be allowed to occur without congestion control backoff and the resulting performance degradation.* Furthermore, handoffs between two neighboring cells of the same network may be able to use the `cwnd`, `ssthresh` and `rtt` values in the previous cell as hints, but interface switching from one network to another cannot assume that any of the parameters in the previous network are valid. Thus, the transmission in the new network should proceed as with a new connection, i.e. perform slow start.

The next section describes our approach to address the above issues in Mobile TCP.

3. Design and Implementation of Mobile TCP

In this section, we describe the design and implementation of Mobile TCP. We first describe the mechanisms for performance improvement upon handoffs by distinguishing the different causes of packet loss and reacting appropriately, and then describe the mechanism for connection management.

3.1. Performance Management

In order to improve the performance of TCP upon handoff, we introduce the following mechanisms to address the issues raised in Section 2.

1. The mobile host must discover an interface switch or a handoff and notify the correspondent host, so that both end-points are aware that subsequently observed losses of the data in transit should be (optimistically) attributed to mobility rather than congestion.

2. Once the end-points are aware of mobility-related activity, the sender 'marks' the data that is in its retransmit queue (i.e. data that have been sent but are unacknowledged). A lost packet in this range triggers retransmission, but not the corresponding congestion control mechanism.

3. Upon an interface switch, the sender resets the `cwnd`, `ssthresh`, `rtt` and `RTO` values, and proceeds to do a slow start as though it were initiating a new connection.

4. While the slow start mechanism above may work well for handoffs just as it does for interface switching, we can use the stable values from the previous cell as hints. We use the same `rtt` values in the new cell, and set the `cwnd` and `ssthresh` to half of `cwnd`'s old value in order to perform congestion avoidance.

Note that mechanism (4) above is not equivalent to the fast retransmit and fast recovery mechanism proposed in [4], since mechanism (2) ensures that fast recovery will not be triggered for any of the packets that are marked.

Discovery and notification of an interface change or handoff is performed as follows. When the IP endpoint at the mobile host changes, the `ioctl` call corresponding to the downing of an interface or changing of its address is trapped by the kernel device control code. All the sockets that use the affected interface are notified, and a *switch detection* routine is invoked. This causes the mobile host to notify the correspondent host via an option field in the next Mobile TCP packet. Note that if operating on top of IPv6, Mobile TCP can make use of the IPv6 correspondent host notification mechanism instead of using a special option.

During a handoff or an interface switch, packets may be lost due to misrouting or network congestion in any segment between the endpoints. Since outdated routing is the most likely cause of packet loss, Mobile TCP optimistically treats all packet losses during the switching period as the result of mobility rather than congestion. The duration of this optimistic period need only be for the range of packets that might have been lost, which is bounded by the last packet for which an acknowledgment was received and the last packet queued to the IP layer. This range may be as small as a single packet or as large as dozens of packets for connections with large delay-bandwidth products. While the optimistic approach may cause Mobile TCP to react incorrectly to congestion-related losses in the new network, note that either slow start or congestion avoidance is used to expand the congestion window, thus avoiding overloading the new network. On the other hand, erroneously initiating congestion control mechanisms for up to a dozen consecutive packets proves disastrous for the performance of TCP.

As a result of the above approach, packet losses are treated differently depending on whether Mobile TCP detects an interface switch, a handoff, or congestion (the default case).

3.2. Connection Management

Mobile TCP preserves end-to-end semantics by dynamically changing end-point addresses for ongoing connections (thereby subsuming the functionality of Mobile IP).

At each host, Mobile TCP creates a list of all local end-point addresses during device initialization. During connection setup, each end-point specifies its list of known local addresses to its peer by means of a special TCP option. The current address that is used for connection setup is called the 'logical source' (LOGSRC), and the remaining addresses are called 'additional source' (ADDSRC) addresses. Each host maintains the list of addresses that its peer may use during the connection. While on-the-fly additions are not supported in our current implementation (the special option is only supported for the SYN packet), future implementations will support the ability to dynamically add or delete addresses from the list. This will be critical for cases where the mobile host enters a foreign network and acquires a care-of address using a protocol such as DHCP. An authorization mechanism is suggested, however, since a malefic host could 'hijack' an ongoing TCP connection by dynamically sending its address as an ADDSRC and initiating an interface change notification.

As described in Section 3.1, mobile behavior is detected at the mobile host by trapping the `ioctl` call when an interface is changed, upon which Mobile TCP initiates a handoff check mechanism for all sockets that were using this interface. The correspondent host also detects the mobile host's interface switch or handoff, either through explicit notification or implicitly through reception of packets from an ADDSRC address. Incoming packets are passed through the hash table lookup to translate their source IP address to the corresponding logical address, if needed. The first time this operation succeeds, the receiving host concludes that the remote host has changed its interface and updates its socket endpoint appropriately. The remote host calculates which packets it has lost to the network and retransmits those as discussed in Section 3.1.

Normal operation resumes once both hosts have modified their socket endpoints and all marked packets have been transmitted out of the retransmit queue.

4. Performance Measurements

In this section, we first describe the testbed, and then the performance results of a subset of the tests performed.

4.1. Testbed Description

In order to test the performance of our approach, we have instrumented Mobile TCP in two testbed environments. When we began this project in mid-1996, the first testbed consisted of two hosts running an early version of Mobile TCP on Linux 1.3.71. We performed interface switching between a point-to-point 10Mbps Ethernet interface and a 100Kbps SLIP interface, using a 100-MHz 486DX4 laptop with a 3Com 3c589 PCMCIA ethernet adapter, and a 100-MHz Pentium desktop using a 3Com 3c509 ethernet card. The primary motivation for presenting results from this testbed is to show the baseline performance of a stripped down Mobile TCP/IP stack.

The second testbed has been operational since June 1997, and is shown in Figure 1. The testbed consists of 3 backbone subnets, each served by a 200MHz Pentium Pro gateway. 10Mbps point-to-point connections join the gateways, *radha* to *durga* and *atri* to *durga*. *radha* also serves as the gateway of a 2Mbps AT&T WaveLAN subnet. We instrumented *durga* such that it could introduce a 100ms delay for all packets passing through it—thus simulating the effect of a long-haul connection and increasing the delay bandwidth product of connections. For all our performance measurements, the static host was connected to *durga*'s subnet via an Ethernet interface, while the mobile host (the same laptop described in the first testbed) had two interfaces—one each to the Ethernet subnet of *atri* and the WaveLAN subnet of *radha*. All the hosts run Linux 2.0.31, and the end hosts run the Mobile TCP/IP stack.

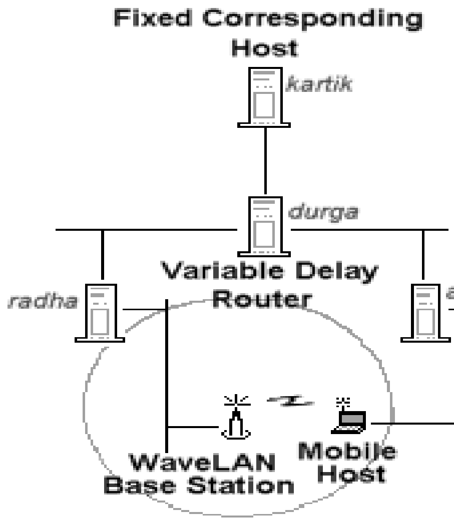

Fig. 1. Mobile TCP Testbed

Trial 1. Our first experiment used the first testbed (described above) to test an interface switch from 10Mbps Ethernet to SLIP, using a low-bandwidth FTP utility. In this early version of Mobile TCP, an interface switch is detected only once retransmission has been initated by an RTO timeout (Linux 1.3.71 does not support fast retransmits). A packet capture of that experiment is shown in Figure 2. This diagram depicts sequence number transmitted versus time; each tick mark on the graph represents a single packet transmitted from the mobile client to the fixed server. Gaps in the graph represent time periods of transmission silence, whereas two tick marks with the same sequence number indicate a retransmission. It can be seen from Figure 2 that the interface switch is initiated at about 1.4 seconds. Nearly two seconds elapse before the new interface is stabilized, and the retransmission timeouts begin to transmit the packets lost during the switch.

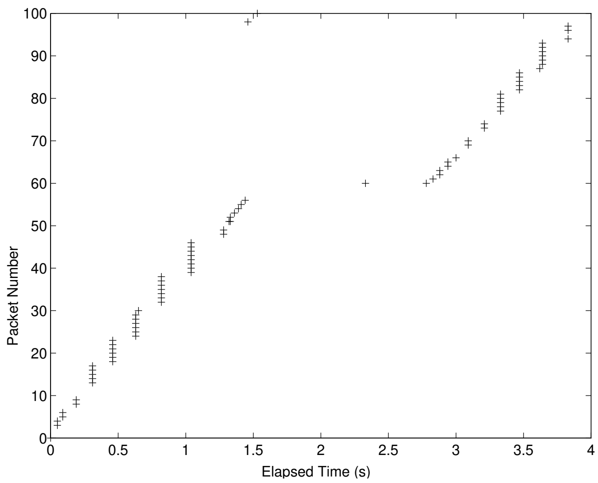

Fig. 2. Trial 1: Low-Bandwidth Interface Switch without Fast Retransmits (Linux 1.3.71)

4.2. Performance Results

In this section, we present results from five experiments. The first experiment is a non-fast-retransmit trial used to baseline strict TCP performance under Linux 1.3.71. The second trial compares modern (i.e. fast-retransmitting) TCP to Mobile TCP in a typical LAN environment. The third trial introduces a 100ms delay, to simulate long-haul networks such as satellite WANs. Trial 4 isolates the impact of fast retransmits in Linux 2.0.31, and the last experiment explores the effect of failing to perform a slow-start on a new network.

Several difficulties were encountered under Linux 1.3.71 which motivated the upgrade to version 2.0.31. Most importantly, the newer version supports fast retransmits, which will be shown to be critical to a quick recovery following mobile behavior. Also, this newer version of Linux implements a read-only socket cache, which improves TCP performance but requires additional complexity in the socket update code. Trials 2 through 5 were conducted using the new testbed, and a high-bandwidth FTP utility. All comparisons to TCP are accomplished by concealing IP changes from the transport, similar to the Mobile IP schemes described in Section 2.

Trial 2. Mobile TCP performance is compared to standard TCP when little network delay is present. TCP and Mobile TCP performed similarly, as shown in Figure 3. Note that both protocols recover in about 200ms, although Mobile TCP's slow-start begins earlier.

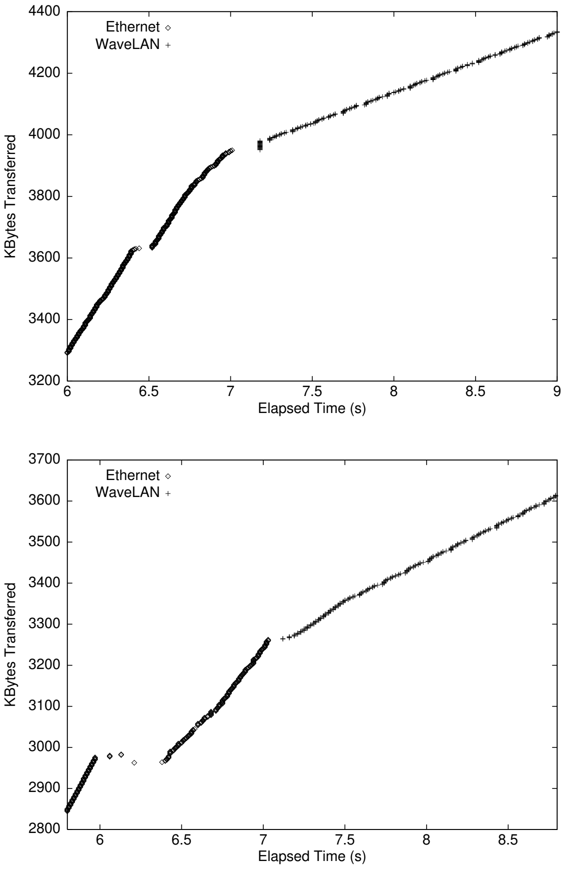

Fig. 3. Trial 2: Performance of TCP (above) compared to Mobile TCP (below) when Network Delay is Negligible

Trial 3. A network delay of 100ms was then introduced, simulating long-haul networks, and Trial 2 was repeated. This had the effect of inducing a loss of about 20 packets in the network. Figure 4 compares TCP to Mobile TCP. TCP's congestion window shrinks to 1, and it must regain linearly, resulting in a recovery time of about 2.3s. Mobile TCP also resets its congestion window to 1, but recovers exponentially in about 1.5s—35% faster than TCP.

Trial 4. The impact of fast retransmits on TCP was isolated by disabling fast retransmits and conducting Trial 3a again. Figure 5 depicts the results (note the different time scales). Clearly fast retransmits are crucial to TCP's 2.3 second recovery in Trial 3. Indeed, observation of this behavior originally motivated the Mobile TCP project, before Linux offered support for fast retransmits.

Trial 5. The performance effects of eliminating Mobile TCP's slow-start was then measured, by simply blasting the lost packets onto the network, ignoring retransmission backoff. Figure 6

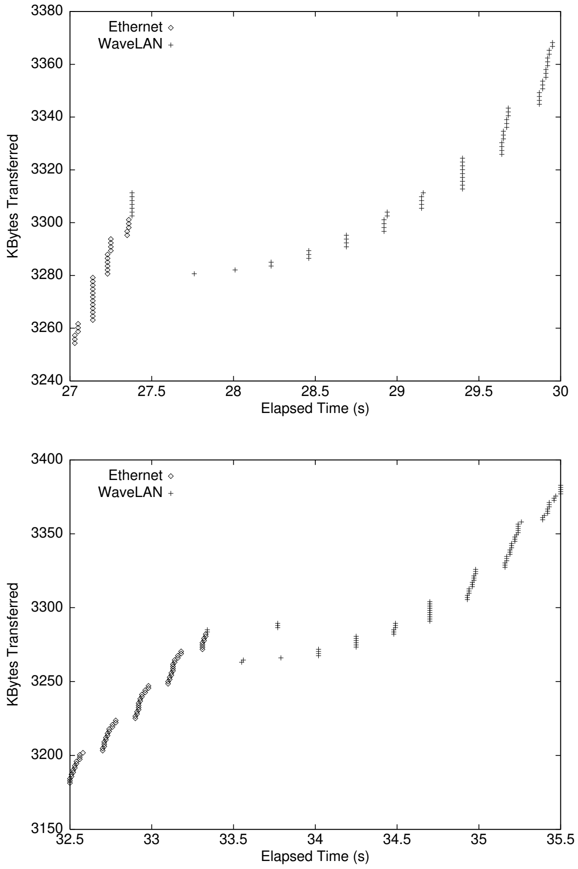

Fig. 4. Trial 3: Performance of TCP (above) compared to Mobile TCP (below) when Network Delay is 100ms

shows the results from this approach. Notice that even though some packets from the first blast overloaded the WaveLAN network, total recovery time is reduced to about 1 second, 87% faster than the original TCP trial. Despite this performance improvement on the unloaded WaveLAN network, this approach is not generally recommended. The effects of overloading the network could have serious implications to other users of the WaveLAN cell, and is likely to be unpredictable even in the isolated case.

5. Summary

Traditional TCP/IP has been shown to be deficient in coping with mobile behavior. Network-layer proposals preserve connection-oriented transport protocols by hiding changes to the IP endpoints. We have demonstrated that this approach allows TCP to make invalid assumptions about packet loss, which can result in significant performance degradation. We have also introduced network-switching scenarios for which protocols based on Mobile IP fail to preserve the network connection, and motivated the need to distinguish this form of mobility from same-network handoff and traditional congestion. The proposed Mobile TCP protocol distinguishes between these forms of packet loss, and reacts in the manner most appropriate to the situation. By accomodating endpoint changes in the transport, new forms of mobility are made possible while simultaneously minimizing route exposure and overall network congestion.

While our approach does address many of the performance problems associated with handoff and interface switching, we have not addressed several fundamental issues relating to efficient and robust transport protocols in mobile computing environments. In

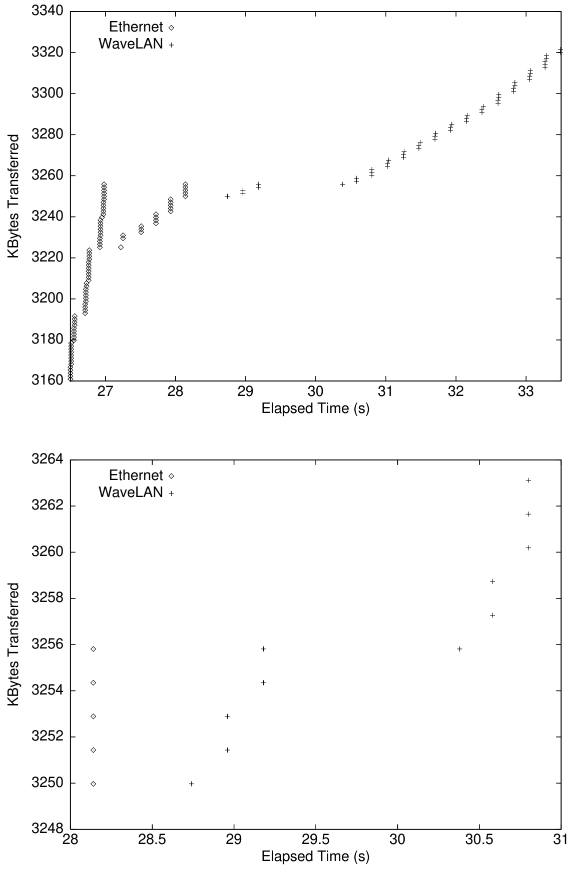

Fig. 5. Trial 4: Effect of Disabling Fast Retransmits when Network Delay is 100ms

particular, we believe that new congestion control schemes need to be developed in order to handle long-haul connections with some wireless links due to the presence of bursty wireless channel error, and differences in channel characteristics upon handoff and interface switching. In particular, none of the congestion control mechanisms listed in Section 2.1 will work well for the above situations. As the realm of networking grows in new and exciting dimensions, so too should the protocols on which it depends.

6. Acknowledgments

The authors wish to thank Vijay Gupta for his initial work on Mobile TCP and time spent in consultation, and Dane Dwyer for his spirited insights and tireless persistence in the lab.

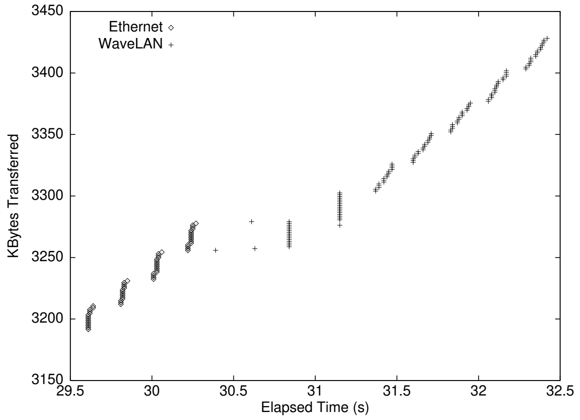

Fig. 6. Trial 5: Mobile TCP Performance without Slow-Start

7. References

[1] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *Proceedings of ACM SIGCOMM*, August 1996.

[2] P. Bhagwat, C. Perkins, and Satish K. Tripathi. "Network Layer Mobility : an Architecture and Survey," *IEEE Personal Communication*, v.3 no. 3, June 1996.

[3] K. Brown and S. Singh. "A Network Architecture for Mobile Computing," *IEEE Network*, September 1995.

[4] R. Caceres, and L. Iftode. "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," *IEEE Journal on Selected Areas in Communications*, Special Issue on Mobile Computing Networks, 1994.

[5] R. Caceres and V. Padmanabhan. "Fast and Scalable Handoffs for Wireless Internetworks," *Proceedings of ACM Mobicom*, November 1996.

[6] J.C. Hoe. "Improving start-up behavior of a congestion control scheme for TCP," *Proceedings of ACM Sigcomm*, August 1996.

[7] V. Jacobson. "Congestion Avoidance and Control," *Proceedings of ACM Sigcomm*, August 1988.

[8] P. Karn and C. Partridge. "Improving Round-Trip Time Estimates in Reliable Transport Protocols," *Computer Communication Review*, v. 17, no. 5. August 1987.

[9] J. Mysore and V. Bharghavan. "A New Multicasting-Based Architecture for Internet Host Mobility," *Proceedings of ACM Mobicom*, September 1997.

[10] C. Perkins and D. Johnson. "Mobility Support in IPv6," *Proceedings of ACM Mobicom*, November 1996.

[11] C. Perkins, Editor. "IP Mobility Support," *Network Working Group, RFC 2002*, October 1996.

[12] W.R. Stevens, *TCP/IP Illustrated, Volumes 1 and 2*, Addison-Wesley Publishing Company, 1994.

[13] F. Teraoka, K. Uehara, H. Sunahara, and J. Murai. "VIP: A Protocol Providing Host Mobility," *Communications of the ACM*, v.37, no. 8, August 1994.