Reliable Transport Performance in Mobile Environments

by

Martin McSweeney

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 2001

Abstract

Expanding the global Internet to include mobile devices is an exciting area of current research. Because of the vast size of the Internet, and because the protocols in it are already widely deployed, mobile devices must inter-operate with those protocols. Although most of the incompatiblities with mobiles have been solved, the protocols that deliver data reliably, and that account for the majority of Internet traffic, perform very poorly. A change in location causes a disruption in traffic, and disruption is dealt with by algorithms tailored only for stationary hosts.

The *Transmission Control Protocol* (TCP) is the predominant transport-layer protocol in the Internet. In this thesis, we look at the performance of TCP in mobile environments. We provide a complete explanation for poor performance; we conduct a large number of experiments, simulations, and analyses that prove and quantify poor performance;and we propose simple and scalable solutions that address the limitations.

Acknowledgements

Many people contributed in many ways to this thesis.

First, I thank my supervisor, Dr. Jay Black, for motivation, direction, guidance, and funding. He allowed me to pursue my areas of interest, and I leave with a wealth of knowledge in those areas.

Next, I thank all the members of the Shoshin research group for making me think, a little bit deeper, about many topics. Every member has contributed in at least a small way. A special thanks goes to our sys-admin, and fellow graduate student, Paul Ward, for accommodating me with the network configurations required.

Finally, I thank my parents for their continuing support, and for taking the time to proof-read the final draft.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the 1990s, growth was explosive in the amount of information on the Internet, and in the number of users accessing that information. Those growths were fueled by increasing bandwidths, higher data rates between PCs and Internet Service Providers (ISPs), and increasing computing power on PCs and web servers. Today, users can download efficiently a rich variety of HTML documents, high-resolution images, and multimedia applications.

In more recent years, there has been massive growth in the demand for high-quality wireless voice connectivity. Users of cell-phones have grown to expect connectivity at any time and anywhere, and, in many ways, service providers have met those expectations. The demand for wireless broadband data services, on the other hand, has been less explosive because of low wireless bandwidths, and because a true ubiquitous computing infrastructure is not yet deployed. Currently, most wireless devices on the Internet are limited to slow data rates, and to web documents that are manipulated and down-sized by wireless communication stacks such as the Wireless Access Protocol (WAP) [3].

A ubiquitous computing infrastructure allows users to access the Internet anytime, anywhere. Users of mobile devices browse the World-Wide Web (WWW) and access documents, images, and applications in the same manner as PC users do today. A change in connectivity is transparent, and the user does not notice a degradation in network performance while using a mobile device. Increasingly higher wireless data rates will open the doors for the deployment of this computing infrastructure.

Expanding the global Internet to include wireless and mobile devices is an exciting area of current research. Because of the vast size of the Internet, and because the protocols in it are already deployed widely, mobile devices must inter-operate with those protocols. Although most of the incompatibilities have been solved, the protocols that deliver data reliably, which generate the majority of Internet traffic, perform very poorly when a host is mobile. A change in location causes a disruption in traffic, and disruption is dealt with by algorithms tailored only for stationary hosts.

In this thesis we identify the fundamental challenges for reliable transport in mobile environments, and we quantify and improve the performance of reliable transport in mobile environments.

The rest of this chapter provides background and an outline for the rest of the thesis. Next, we present the Internet architecture and the protocols that deliver data reliably. Section 1.3 presents the mobile environment. In Section 1.4, we outline the main challenges for delivering data reliably and efficiently in mobile environments, and in Section 1.5, we discuss the contributions to current research and the thesis organization.

1.2 The Internet and Reliable Transport

The Internet is a massive heterogeneous system where the Internet Protocol (IP) [43] is used to provide host-location and routing services over a variety of link-layer technologies (e.g. Ethernet, ATM, FDDI).

Every host on the Internet is assigned an IP address. The address is broken into a network number, or sub-network number, and a host number. When a host sends a datagram, it addresses the datagram to the IP address of the destination host, and when a router receives the datagram, a routing-table look-up is performed and the datagram is forwarded to the appropriate next-hop router. At the destination network, the datagram is delivered directly to the destination host.

IP provides *best-effort* service; although reliable delivery is attempted, it is not guaranteed—a datagram can be lost, dropped, or corrupted for a variety of reasons. Reliability is guaranteed at the layer above IP, the transport layer. A reliable-transport protocol monitors transmitted datagrams and performs loss-recovery in the absence of time-sensitive acknowledgments from the receiving host.

The Transmission Control Protocol (TCP) [44, 49] is the dominant reliable-transport protocol in the Internet. Numerous applications require reliability, including HTTP [17], `ftp` [45], and e-mail [26], and most of those applications use

TCP. It is estimated that 83% of all datagrams in the Internet are TCP packets [29]. As well as delivering reliably and efficiently, TCP also performs congestion control when the network is limited. This control is a fundamental reason why the Internet has scaled.

The next section discusses how IP and TCP have been integrated into mobile environments.

1.3 The Mobile Environment and Network-Layer Mobility

A mobile environment is one where a host moves from one network to another. Typically, the environment consists of wireless devices communicating with wireline base stations. The base stations act as next-hop IP routers, and are typically organized in a cellular topology with an IP backbone.

The fundamental challenge for network-layer mobility is that hosts are identified by static IP addresses, not by location. If a host migrates, datagrams destined for it are delivered incorrectly to the old network.

Mobile IP [37] solves this challenge and is the current protocol supported by the Internet Engineering Task Force (IETF) [2] standardization body for providing network-layer mobility. Mobile IP allows a mobile host to move from one network to another, and still be both identifiable and locatable. Datagrams that reach the previous network of the mobile are re-directed to the new network.

1.4 The Problem

In a wireline network, TCP packets are lost primarily because of congested router queues. Those types of losses are called *congestion-induced*. TCP assumes that all losses are congestion induced.

When losses are *movement-induced*, or a result of mobility, TCP also assumes congestion occurred. That incorrect assumption causes TCP to perform very poorly in mobile environments. Unfortunately, the inherent temporary disconnections in mobility protocols, including Mobile IP, result in high probabilities that movement-induced losses occur during mobility.

1.5 Thesis Contributions and Organization

To current research in the area of TCP performance in mobile environments, this thesis contributes

1. A complete explanation for poor performance;
2. A large number of experiments, simulations, and analyses that prove and quantify poor performance; and
3. Simple and scalable solutions that can improve performance significantly during Mobile IP handoffs.

We focus on the performance of a mobile Internet browser, and we assume mobile devices act primarily as clients, not servers.

The thesis organization is as follows. In Chapter 2, we present the details of Mobile IP and TCP, as well as a set of tools that are used to analyze performance.

In Chapter 3, we identify the reasons for poor performance in mobile environments, and we quantify, through experiments and simulations, the effects that mobility can have on TCP. Chapter 4 shows, through experimentation, why TCP performance is poor with Mobile IP, and Chapter 5 shows how to improve that performance. The thesis is concluded in Chapter 6.

Chapter 2

Background

2.1 Overview

This chapter presents an overview of the material referenced throughout the thesis. Mobile IP is presented in Section 2.2, and TCP in Section 2.3. Tools used in later experiments are presented in Section 2.4. For a more in-depth coverage of these topics, the reader is referred to Perkins [39] and Stevens [49].

2.2 Mobile IP

In the Internet, stationary hosts, for the most part, maintain IP addresses that do not change. These static addresses are important for many reasons, including location and identification. Protocols such as TCP and UDP [41] associate a host with a single IP address, and work only if the address does not change. For a host to migrate to a new network, however, and still be locatable, a new IP address must be acquired.

Mobile IP is a protocol that solves this challenge because it allows a host to change networks and still be both locatable and identifiable. With Mobile IP, a mobile host (M) is identified by a fixed IP address on its home network, but located by a temporary care-of-address on the network where it resides. Datagrams that reach the home network of a mobile are tunneled, by IP-in-IP encapsulation [36], to the care-of-address.

In Mobile IP, special routers, or mobility agents, are used to provide mobility services for a mobile. At a foreign network, a *foreign agent* (FA) provides a care-of-address, and operates as the end-point of a tunnel from the home network. When a foreign agent receives datagrams, it decapsulates them and forwards them directly to the mobile. At a home network, a *home agent* (HA) maintains bindings between home mobiles and care-of-addresses, and intercepts and tunnels datagrams to foreign agents.

When a mobile migrates into a foreign network, several functions must be performed. First, the mobile must discover the existence of foreign agents in the network, and must realize that it has migrated. Second, the mobile must communicate this new location information to its home agent so that the mobility binding can be updated. Finally, at the home agent, the mobile must be authenticated and authorized to use mobility services. This entire process is called a handoff.

The rest of this section describes Mobile IP and the above functions in more detail.

2.2.1 Agent-Discovery Mechanisms

A mobile uses *agent advertisements* and *agent solicitations* to discover and monitor the presence of mobility agents.

Agent advertisements are ICMP messages [15], similar to ICMP router advertisements [42], that are broadcasted periodically by mobility agents. A mobile realizes it is near an agent when it receives an advertisement.

Advertisements contain fields that indicate the address of the sending agent, the agent's available care-of-addresses, and the advertisement *lifetime*. The lifetime is used by a mobile to monitor the presence of an agent. If a lifetime expires before another advertisement arrives, the mobile concludes that the agent is no longer reachable. The recommended rate for sending agent advertisements is one per second, with an advertisement lifetime of three seconds [37].

Agent solicitations are ICMP messages sent from mobiles to mobility agents. A mobile sends a solicitation when it is searching for an agent. A mobility agent responds to a solicitation with an agent advertisement.

2.2.2 Movement Detection

Before a mobile can register a new foreign agent with its home agent, it must first realize that it has migrated from one network to another. Perkins [39] defines three primary mechanisms by which a mobile detects migration: Prefix Matching (PM), Eager Cell Switching (ECS), and Lazy Cell Switching (LCS). The mechanisms are based on the arrival of agent advertisements, and on the lifetimes of the advertisements.

Prefix Matching

Prefix Matching requires that mobility agents include a special *prefix-length extension* in the advertisements. Using this extension, a mobile can compare the network prefixes between two advertisements: the advertisement from its current agent, and the advertisement from a different agent. If the prefixes are identical, the two foreign agents reside on the same subnet, and the mobile has not migrated. If different, the mobile can conclude that it is moving into a new network, and can register the new agent.

The use of this strategy requires that adjacent foreign agents include the extension, and that care is taken if adjacent agents have the same prefix-length in their network addresses. Because of these and other problems [39], Prefix Matching is an undesirable choice for movement-detection.

Eager Cell Switching

Eager Cell Switching is the most aggressive strategy. It is based on the concept that moving entities tend to follow straight-line trajectories, and that changes in trajectory are gradual [39]. With this in mind, a mobile that moves into one network will continue moving into that network and away from its previous network.

With ECS, a mobile registers an agent immediately when a *new* advertisement is received. An advertisement is *new* if it comes from an agent that the mobile has not seen before, or at least in the last three seconds, the recommended advertisement lifetime.

Lazy Cell Switching

Lazy Cell Switching is based on the lifetime of advertisements. A mobile using LCS registers an agent only when the lifetime of its current agent expires. Therefore, LCS is most suitable in situations where a mobile changes directions repeatedly [39]. LCS assumes that a new advertisement does not necessarily imply that the mobile will continue moving into the new network.

2.2.3 Registration

After a mobile decides to use the mobility services of a foreign agent, it must communicate that information to its home agent. The home agent must update the mobility binding and configure a tunnel to the new care-of-address. This process is called registration.

As shown in Figure 2.1, registration involves the exchange of two messages: *registration requests* and *registration replies*.

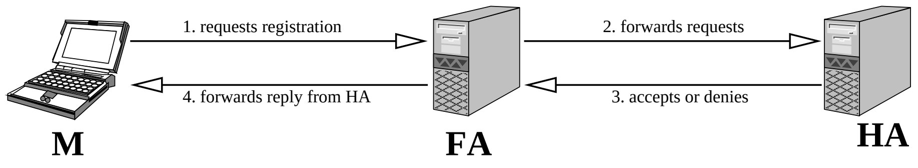

Figure 2.1: Mobile IP registration

A registration-request message is a UDP datagram that originates at a mobile. The message contains the address of the foreign agent, and the care-of-address the mobile will use. If the foreign agent is willing to service the mobile, it forwards the message to the home agent. If the home agent is also willing, it updates the mobility binding, configures a tunnel, and replies with a registration-reply message. The reply message indicates whether registration was successful, and indicates how long the home agent will service the mobile in that location. Upon receiving the reply message, the foreign agent configures the end-point of the tunnel, and forwards the message to the mobile. After the mobile processes the message, registration and the handoff are complete.

2.2.4 Security

Users of mobile devices will be required to pay for Internet access, just as household PC users now pay their local Internet Service Providers. Although the logistics of servicing mobiles in foreign domains are not yet established formally, the IETF Mobile IP working group [1] is developing a model to *authenticate* mobiles to foreign agents, *authorize* services to mobiles in foreign domains, and *account* for the services used by mobiles. This is known as the Accounting, Authentication, and Authorization services (AAA) [19]. The model introduces two new entities into Mobile IP: the AAAF and the AAAH. The AAAF is a server in a foreign domain responsible for handling AAA services on behalf of foreign agents in that domain. The AAAH is a home domain server which authenticates and authorizes mobiles to AAAFs.

Four secure relationships exist in the AAA model: between the foreign agent and the AAAF, between the AAAF and the AAAH, between the AAAH and the mobile, and between the AAAH and home agent. When a mobile node enters a foreign domain, it must send its credentials to the AAAF via the foreign agent. The

AAAF verifies these credentials with the AAAH. If the mobile is authenticated, the AAAF approves the foreign agent to service the mobile. Because a major component of the latency involved in performing AAA services is the round-trip time between the foreign network and the home network, it is proposed that these services be integrated into the registration process through mobility extensions in the registration messages [12].

Another important aspect of security is the securing of user payload. If a mobile expects the same level of security while roaming as it does while at home, the home agent and the foreign agent must secure the tunnel for the mobile. This approach, not yet standardized, involves the use of IP security protocols (IPSec) [25] with an infrastructure for managing security keys.

2.3 Transmission Control Protocol

The Transmission Control Protocol (TCP) offers reliable, connection-oriented, and in-order delivery of data for application-layer programs. It is the predominant transport layer protocol in the Internet.

TCP is implemented not only to deliver reliably, but also to deliver efficiently. Special mechanisms are used to control the rate of transmission and to discover the limits, or capacity, of the network. TCP respects those limits, and throttles the transmission rate when congestion occurs.

The rest of this section describes many details of TCP, and is organized into three parts. First, we discuss the semantics of TCP and how data is delivered reliably. Second, we discuss the congestion-control mechanisms of TCP, and how data is delivered efficiently when congestion occurs. In the third part, we discuss a special feature of TCP, called *persist mode*, that empowers a receiver to throttle transmission at the sender.

2.3.1 Reliable Transport

For reliable transport, TCP uses a *sliding window* mechanism. The window, or *send window*, slides over a linear representation of the data to transfer. The sender transmits the data that the window covers, and waits for the receiver to acknowledge that data. To keep track of transmissions and acknowledgments, TCP breaks the data into *segments*, and numbers them sequentially. When an acknowledgment arrives, the sender *shifts* the window right, usually by the number of segments that were acknowledged, and new segments can then be transmitted. Typically, during data transfer, a receiver acknowledges every other segment.

The size of the send window is equal to the socket-buffer size at the receiver. When a receiver sends an acknowledgment, or ACK, it advertises that size. At any time, the receiver can shrink the send window by advertising a small buffer.

To transmit data, TCP passes the segments to the network layer, or IP. IP fragments the segments into sizes that are suitable for transmission across the network. Those fragments, with TCP and IP headers, can be called either *packets* or *datagrams*. The size of a packet is limited by the *maximum transmission unit* (MTU) of the network.

TCP uses two mechanisms to guarantee reliability. The first is the duplicate acknowledgment scheme, or *fast-retransmit*. When a segment arrives out of order, the receiver sends a duplicate acknowledgment for the last segment acknowledged. When a sender receives three duplicates, it concludes that a segment was lost, and it retransmits the oldest unacknowledged one.

However, if there are not enough segments in transit to generate three duplicate acknowledgments, or if more than one segment in a window is lost, the sender must rely on a different mechanism to recover: *retransmission timeouts.*

When a segment is transmitted, the sender estimates how long it should wait for an acknowledgment before concluding that the segment was lost. If an acknowledgment is not received by that estimated time, a retransmission timeout occurs and the segment is retransmitted. Because a retransmission timeout indicates that multiple segments may have been lost, and because the sender cannot deduce which segments were lost, a timeout re-sets all timers and begins retransmitting the entire send window.

A sender must exercise caution, however, when estimating the time to wait for an acknowledgement. If the time is too short, a retransmission may be unnecessary and may add extra load to the network. If too long, on the other hand, the wait can have a serious impact on efficiency. To solve this dilemma, a sender calculates a reasonable timeout value, or RTO, by sampling the round-trip times (RTT) of the connection. For each send window, a sender calculates the time that elapses between transmitting a particular segment and receiving the acknowledgement. The RTO is then a running calculation of a multiple of a smoothed estimate of the mean round-trip time [44], and of a measure of variance using a smoothed mean difference of the samples [22]. To avoid spurious and unnecessary retransmissions, RFC 2988 [35] recommends that senders keep a one-second lower-bound on the RTO.

If a retransmission from a retransmission timeout is lost, either the network is very congested and needs relief, or a problem exists at the receiving host. Therefore, consecutive timeouts are separated exponentially in time until an upper-bound of 64 seconds is reached.

2.3.2 Congestion Control

Congestion occurs when routers are overloaded and cannot keep up with incoming traffic. When this happens, routers are forced to drop datagrams. In the wireline Internet, it is estimated that 99% of all TCP segment losses are due to congestion [22].

Early versions of TCP used reliability schemes that were plagued by unnecessary retransmissions, and that contributed heavily to network congestion [16]. As the Internet grew in the 1980's, it was brought to a stand-still several times because of a series of congestion collapses [31]. In response, a congestion-control scheme [4, 22] was introduced in 1988. The scheme incorporated two important algorithms into TCP: *slow-start* and *congestion avoidance.*

Slow-Start

At the start of a connection, a sender must determine the capacity of the network. Although the sender can initially transmit a large burst of segments, few segments will reach the receiver if the network is congested. Senders are discouraged from doing this, however, because losses will result in the retransmission of the entire burst, and because transmitting a burst of segments when the network is limited is equivalent to " pouring gasoline on fire" [22].

In theory, the amount of data a sender can have in transit equals the *bandwidth-delay product* of the network. If transmitting at the maximum rate, and a segment is being placed on the network as another one leaves the network, TCP is said to be operating at *equilibrium*. The goal of slow-start is to reach that equilibrium point—quickly, but without an initial large burst of segments.

Slow-start introduces two new state variables: the *congestion window* and the *slow-start threshold*. The congestion window represents the knowledge of network capacity, and is used to control the transmission rate. The size of the send window is modified to become the minimum of the congestion-window size and the receive-buffer size. The slow-start threshold acts as a safe upper-bound on slow-start to avoid a situation where an extremely large burst of segments is inadvertently transmitted. When slow-start reaches that bound, congestion avoidance takes over.

Initially, a sender possesses little knowledge of capacity, and the congestion window is set to one segment-size. When that segment is acknowledged, the window grows to two segment-sizes, and when those are acknowledged, to four segment-sizes. The exponential growth continues until either the receive-buffer size is reached (in which case the sender continues transmitting at that rate), the slow-start threshold is reached, or segment losses occur.

If segment losses occur, the sender can conclude that the capacity of the network is somewhere between half the congestion-window size (or the size at the last exponential increase), and the congestion-window size. Therefore, when a retransmission timeout eventually occurs, the sender sets the slow-start threshold to half the congestion-window size, sets the congestion-window size to one segment-size, and again invokes slow-start. This time, however, slow-start should end before losses occur a second time.

After the slow-start threshold is reached, slow-start ends and congestion avoidance begins.

Congestion Avoidance

Because congestion avoidance is invoked when the congestion window is near network capacity, growth of the window becomes much slower. During congestion avoidance, a sender increases the window linearly. In many TCP implementations, the window grows by a segment size every other round-trip time, or every time a full window of segments is acknowledged. When the window reaches network capacity, a growth should cause only one segment to be lost, and fast-retransmit can recover that loss quickly. When fast-retransmit is invoked, the slow-start threshold is again adjusted, but the congestion window drops to only half its current size. In this way, the long wait for a retransmission timeout is avoided.

2.3.3 Persist Mode

A receiver may want to pause a connection for a variety of reasons. An application-layer program may stop taking TCP data, for example, or the machine of the receiver may be slow or overloaded.

A receiver can stop a sender from transmitting by advertising progressively smaller receive-buffer sizes. The advertisement of a buffer of size zero is called a *zero-window-size acknowledgement*, or ZWSA.

Figure 2.2 shows how a receiver closes the send window. In the example, the window size initially is equal to the receive-buffer size, or 5 segments, and segments 14 through 18 are transmitted. During normal operation, the acknowledgement for segment 14 advertises a buffer of 5 segments, and the left and right edges of the send window shift right by one segment-size. That would then trigger the transmission of segment 19. But here, the sender advertises a buffer of only 4 segments. As a result, only the left edge of the window is adjusted, and the sender is prevented from transmitting segment 19. The acknowledgement for segment 16 advertises an even smaller buffer, and for segment 18, a buffer of size zero.

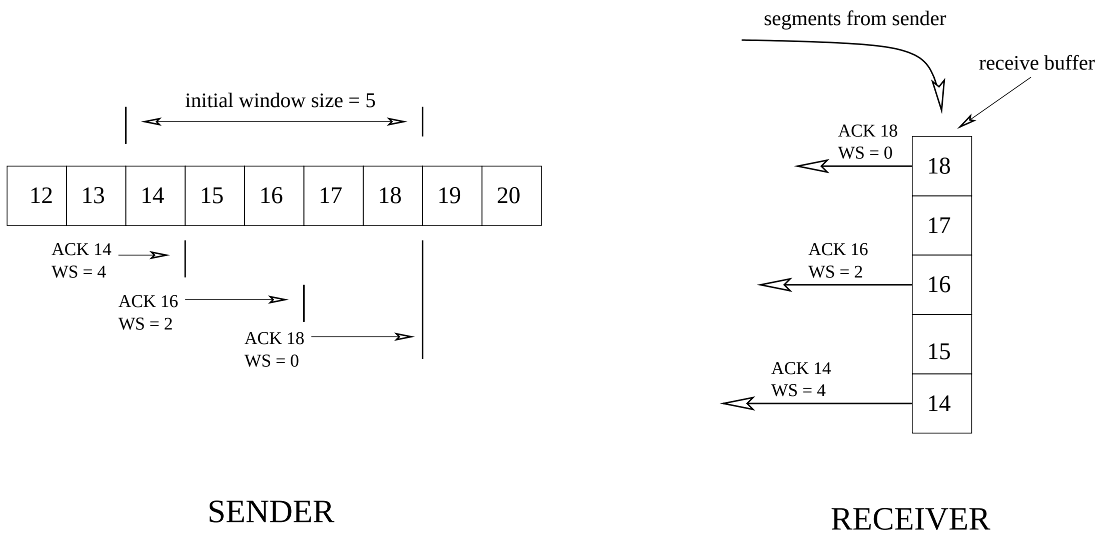

Figure 2.2: Sending a zero-window-size acknowledgement

When a sender receives a zero-window-size acknowledgement, it enters a state called persist mode. In persist mode, a sender cannot transmit any data and waits for an acknowledgement that re-opens the send window. Because acknowledgements are not delivered reliably, the sender periodically transmits *window probes* to generate them [10]. Window probes usually contain one byte of data, at most. a receiver can send a zero-window-size acknowledgement prematurely, although not recommended [10, 44], effectively invalidating the transmission of in-transit segments. Because this can confuse a sender, TCP implementations are required to expect that incorrectly-implemented receivers can send premature zero-window-size acknowledgements, and are required to recover if a zero-window-size acknowledgement is sent [10, 44]. In some implementations, a premature zero-buffer advertisement is simply ignored by the sender.

An important feature of persist mode, in many implementations, is that all acknowledgement timers are frozen. Therefore, while in persist mode, a sender does not incur retransmission timeouts, and the congestion window is not affected. When an acknowledgement re-opens the send window, the sender resumes transmitting at the rate existing prior to the zero-window-size acknowledgement.

Chapter 5 discusses persist mode, and premature zero-window-size acknowledgements, in more detail.

2.4 Network Tools for Performance Analysis

Many tools were used in this thesis to analyze transport and network layer performance. A few of those tools are presented in this section.

Netfilter

Netfilter [9] is a framework for packet manipulation that is built into Linux 2.4

kernels.

The framework defines *hooks* at various points in the IP protocol stack. Kernel modules can register at the hooks and intercept packets at the different stages of stack traversal. Packets can either originate at the intercepting machine, be destined for the intercepting machine, or be passing through the intercepting machine (in the case of a router or firewall).

The user-space tool `iptables`—a replacement for the Linux `ipfwadm` and `ipchains` firewall tools—can add and delete rules that affect the types of packets for which the kernel listens. The rules also define the fate of a packet that is intercepted. A packet can be dropped, usually for security reasons, re-injected back into traversal, or queued for user space. In the latter case, the kernel passes the packet to a listening user-space process via a *Netlink* socket. In user space, before re-injecting the packet, the process can examine and manipulate its contents.

There are many reasons for intercepting and manipulating packets. Packet manipulation, for example, can be used to control the uplink traffic rate on a busy wireless link [27]. An intercepting machine can lie in the path between the wireless link and the wired network. Using `iptables`, the kernel on that machine can be told to listen for traffic between two hosts: a mobile and a stationary. When intercepting the traffic, the advertised buffer size in the acknowledgements can be lowered.

In this thesis, we used Netfilter mostly to delay traffic and to emulate long delay paths between two networks that are otherwise within milliseconds of each other.

Test-TCP

Test-TCP [30], or `ttcp`, is a user-space utility for analyzing network-layer performance. Like `ftp`, `ttcp` uses TCP to transfer data between two hosts. Unlike `ftp`, however, no hard-disk accesses are needed—all data is taken from main memory. As a result, `ttcp` is more precise when analyzing network-layer performance.

We used `ttcp` mostly to generate TCP workload. The utility provides a simple user interface, and the source code can be modified easily to change data sizes and buffer sizes.

Tcpdump, tcptrace, and xplot

Tcpdump [23] is a widely used packet-capture tool for watching traffic on a link-layer interface. The interface driver passes a copy of packets to `tcpdump`. If a packet matches the criteria given by a user, via command-line parameters, the packet's header information is displayed. An example execution is as follows:

```
...
[bear] tcpdump -i eth0 proto TCP and host belle.
tcpdump: listening on eth0
18:29:11.592558 bear.53158 > belle.ftp: P 4627:4667(40) ack 2135 win 27800 (DF)
18:29:11.592984 belle.ftp > bear.53158: P 1:41(40) ack 40 win 9648 (DF)
18:29:11.593064 bear.53158 > belle.ftp: . ack 41 win 27800 (DF)
18:29:11.593276 belle.ftp > bear.53158: P 41:129(88) ack 40 win 9648 (DF)
18:29:11.593338 bear.53158 > belle.ftp: . ack 129 win 27800 (DF)
18:29:11.593494 belle.ftp > bear.53158: P 129:185(56) ack 40 win 9648 (DF)
18:29:11.593548 beard.53158 > belle.ftp: . ack 185 win 27800 (DF)
...
```

In this example, `tcpdump`, executed on host *bear*, is listening on interface *eth0*, and is watching TCP traffic that has a source or destination IP address of *belle*. The output includes sequence numbers, acknowledgement numbers, timestamps, and advertised buffer sizes. This example is simple, and `tcpdump` accepts a multitude of filtering arguments to watch different kinds of traffic, and to produce more verbose output.

`Tcpdump` can be used to obtain critical statistics about TCP connections. It can be used, for example, to monitor round-trip times, to watch retransmissions, and to observe the size of the congestion window.

Written by Shaun Ostermann at Ohio University, `tcptrace` [33] is a utility that can parse `tcpdump` output and collect those critical statistics. The utility provides features, for example, to produce graphs that plot, across time, segments transmitted, segments acknowledged, and round-trip time estimates. The graphs are viewed in `xplot` [48].

Figures 2.3 and 2.4 show examples of these graphs. In the first figure, the graph shows the transmitted and acknowledged segments at the sender. The x-axis represents time, and the y-axis represents the sequence number space of the connection. The arrows and diamonds represent the times that particular segments were transmitted. The line below tracks the acknowledgements, and the line above tracks the receive-buffer size. The letter "R", seen in the middle of the plot, indicates a retransmission, and the number "3" indicates a third duplicate acknowledgement.

Figure 2.4 is another graph of the same connection. The y-axis represents the number of outstanding bytes, and it approximates, in bytes, the congestion-window size. The initial fast increase near the beginning of the plot is the result of slow-start; the sudden drop near the middle is the result of the retransmission; and the gradual rise after that is the result of congestion avoidance.

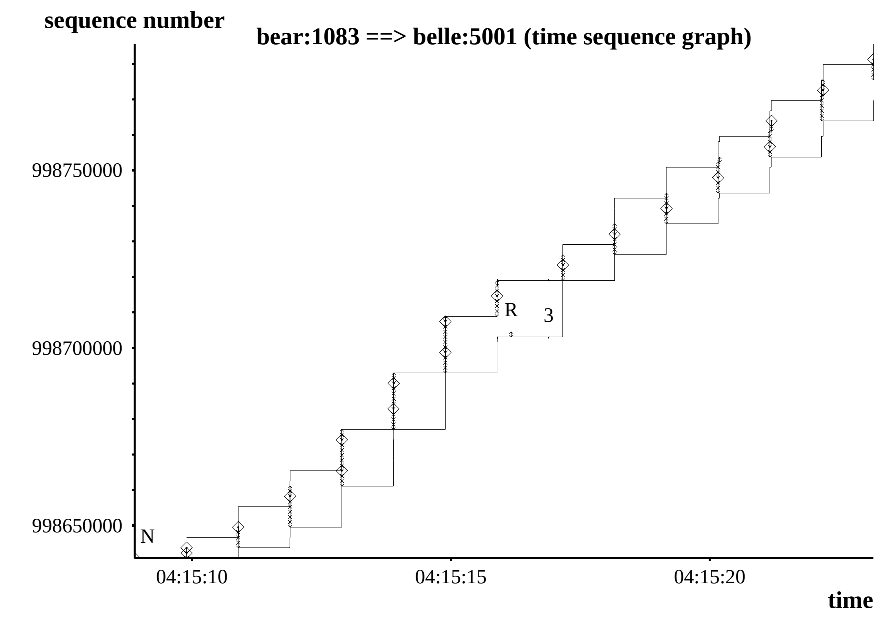

Figure 2.3: Tcptrace time-sequence graph

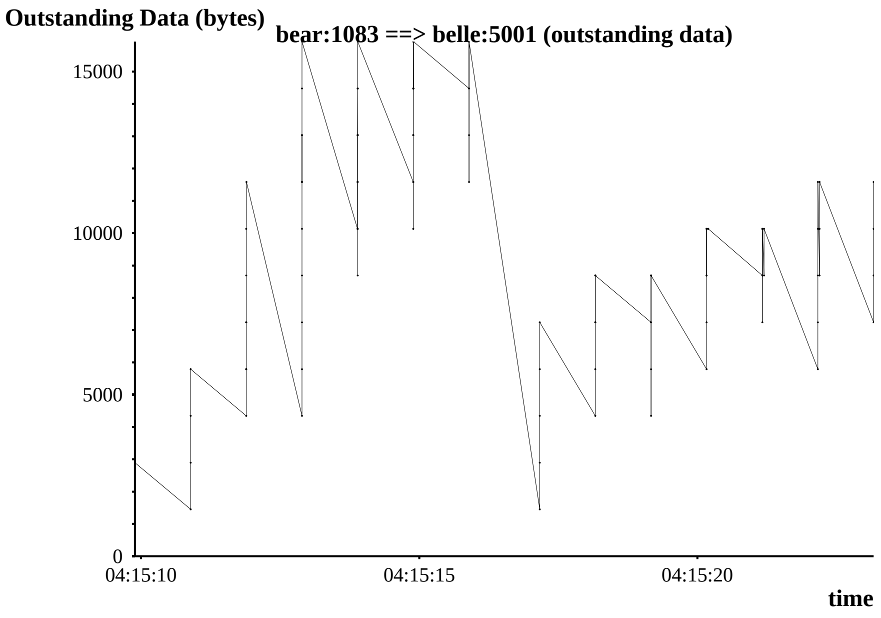

Figure 2.4: Tcptrace outstanding-data graph

Chapter 3

TCP Performance with Mobile Receivers

3.1 Overview

As we mentioned earlier, TCP is tailored for wireline networks. When congestion occurs and packets are dropped, a sender stops transmitting, waits for a retransmission timeout, and invokes slow-start. Although congestion degrades throughput, the reactions of a sender prevent further and more serious degradations because they allow the network to recover. If senders did not implement congestion control, network queues would not empty and there would be congestion collapses [31].

Clearly, performance is affected negatively if congestion control is invoked when there is no congestion. The reactions of a sender degrade performance because the transmission rate is throttled well below the capacity of the network. This is the case for receivers in mobile environments. A temporary disconnection caused by a handoff between two base stations can cause packet losses, and those losses are interpreted as being congestion-induced.

Previous research indicates two primary reasons for performance degradations with mobile receivers [13, 18, 28]. First, there is the pause in communication caused by waiting for a retransmission timeout. Figure 3.1 shows a trace, generated at the sender, of the acknowledged segments during a connection with a mobile receiver. In the figure, the handoff causes packet losses, forcing the sender to wait for a retransmission timeout. However, the timeout does not occur until long after the short handoff completes. The time between the handoff completing and the retransmission timeout occurring is unnecessary, and many round-trip times elapse during it.

Although the TCP specification states how retransmission timeouts should be implemented, the length of time a connection can lie idle unnecessarily can be much worse, or better, than we might expect. Incorrect implementations, programming errors, and deliberate violations of the specification can alter the expected behavior of a sender [34]. One implementation, for example, might retransmit very aggressively, resulting in short idle times, while another might have a high lower-bound on the time to wait for a retransmission timeout, resulting in long idle times. These behaviors, and the behavior we can expect in general, can be determined by experimenting with different TCP implementations.

The second primary reason for TCP performance degradations in mobile environments is the incorrect invocation of congestion control. The transmission rate is throttled by slow-start and congestion avoidance, and recovering the old trans-

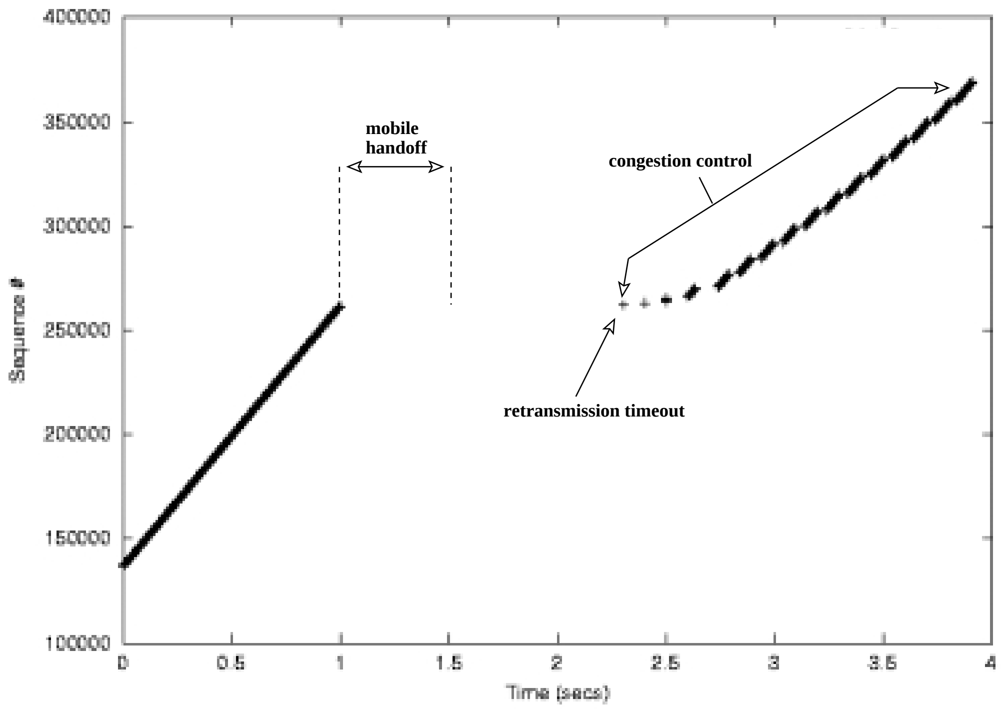

Figure 3.1: A handoff triggering a retransmission timeout and congestion control

mission rate, or bringing the rate to its maximum, is a slow process. During that process, many segments that could be delivered successfully by the network are not transmitted and, as a result, many round-trip times are lost unnecessarily. Figure 3.1 shows how the transmission rate is very low after a retransmission timeout.

The extent to which congestion control impacts a connection depends largely on the maximum size possible for the congestion window. If the maximum size is large, congestion control can throttle the sender unnecessarily for a very long time. If small, on the other hand, the congestion window reaches its maximum size quickly and performance degradations should be less noticeable.

The length and impact of congestion control also depends on the size of the congestion window when a retransmission timeout occurs. If the window is small when a retransmission timeout occurs, the slow-start threshold can be set to a very small value. In such a case, an extremely long time can elapse before the congestion window reaches its maximum size.

The purpose of this chapter is to determine, through experiments and simulations, the degradations in performance we can expect from retransmission timeouts and congestion control. Previous research is extended by looking at the behavior of different TCP implementations, by simulating behavior with different maximum congestion-window sizes, and by comparing results to an implementation that does not invoke congestion control. The next section examines retransmission timeouts, and Section 3.3 examines congestion control.

3.2 The Effects of Retransmission Timeouts

Several factors combine to determine how long a sender waits before incurring a retransmission timeout. A dominant factor is the RTO calculation. As discussed in Section 2.3, the RTO is calculated from round-trip time samples and from the variance in those samples; it is the sender's estimation of the time to wait for an acknowledgement before concluding a particular segment was lost. If the round-trip time is large, the RTO is also large. If the round-trip time is small, the RTO is limited to the sender's RTO lower-bound.

The extent to which the RTO corresponds to the actual time of a retransmission timeout depends on the *heartbeat timer* of the sender's TCP implementation. This timer periodically triggers the kernel interrupts that check for RTO expirations on unacknowledged segments. If the timer granularity is high, heartbeats are frequent and timeouts occur close to the set times. However, if the granularity is low, timeouts may occur hundreds of milliseconds from the set times.

Figure 3.2 illustrates the consequence of an implementation that uses a low-granularity timer. In this example, the granularity is 500 msecs. The RTO is 1100 msecs, and the timeout is set at time 150 to occur at time 1250. However, the next heartbeat after time 1250 occurs at time 1500, which is 250 msecs longer than the set timeout.

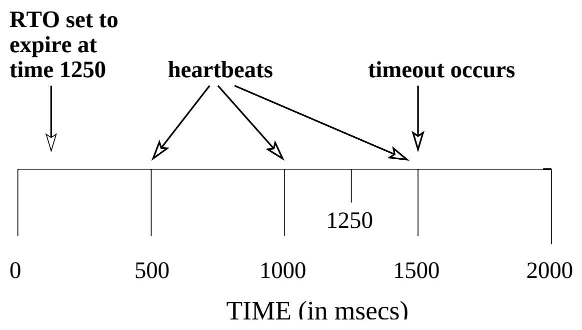

Figure 3.2: The heartbeat timer of TCP

TCP implementations are not allowed to set timeouts of less than 2 heartbeats [10]. If an implementation used a 1-heartbeat timeout, the timeout could inadvertently occur within a few milliseconds of when it was set.

The number of successive timeouts also influences the time to wait for a retransmission timeout. The RTO is doubled with each retransmission timeout, and many successive timeouts can result in very long pauses.

The above factors can lead to long and unnecessary communication pauses before handoff-affected TCP connections incur retransmission timeouts. Although long pauses are beneficial for congested networks, they can leave connections on mobiles inactive unnecessarily. A short pause, on the other hand, can cause a connection to re-start more quickly, but short pauses are discouraged because they introduce the possibility of unnecessary retransmissions, and because those retransmissions can add extra load to an already congested network.

To determine the general behavior of TCP implementations, and to determine if pauses are long and can affect connections recovering from handoffs negatively, we looked at the retransmission-timeout behavior of four current implementations: FreeBSD 2.7, SunOS 5.6, AIX 4.3, and Linux 2.2. Those implementations were chosen because they are used widely in research and because they were readily available to us.

To construct real Internet scenarios, we analyzed implementation behavior with different connection round-trip times: 60, 100, 200, 500, and 1000 milliseconds. Those times were chosen because the inherent characteristics of the Internet cause round-trip times to fluctuate depending on travel distance, time of day, and congestion levels. Using the network utility `ping`, we found that on-campus RTTs were within 15msecs, that continental RTTs largely fell in the range between 50 msecs and 200 msecs, and that oceanic journeys showed RTTs of up to 1000 msecs. Assuming that mobiles are constrained to the low-bandwidth characteristic of wireless links, and are subject to the latency involved in mobility mechanisms (IP tunneling for example), the round-trip time between a mobile and a server is at least tens of milliseconds.

The rest of this section describes those experiments and the results.

3.2.1 Experiments

We used the network configuration shown in Figure 3.3 and Table 3.1. All of the servers were, at most, two on-campus IP hops from the receiver, and the base RTT was 2 msecs. The delayer host was used to delay segments and acknowledgements by an equal amount in order to reach the targeted round-trip times.

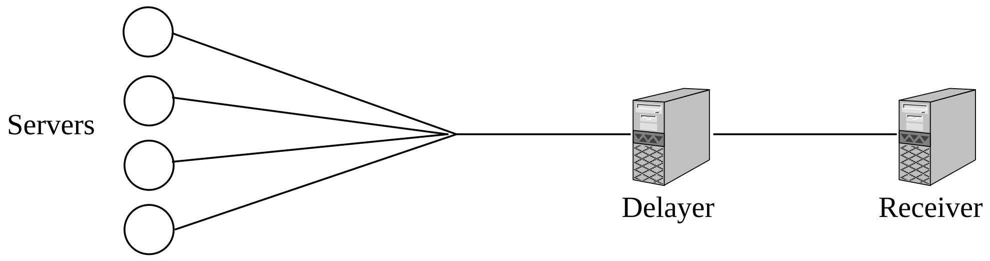

Figure 3.3: Network for analyzing retransmission timeouts

Host	**OS**	**Architecture**	**Purpose**
Linux Server	Linux 2.2	Intel i686	TCP sender
Sun Server	SunOS 5.6	SPARC sun4u	TCP sender
FreeBSD Server	FreeBSD 2.2.7	Intel i386	TCP sender
AIX Server	AIX 4.3	IBM RS6000	TCP sender
Delayer	Linux 2.4	Intel i686	delays TCP traffic between server and receiver
Receiver	Linux 2.2	Intel i686	TCP receiver

Table 3.1: Hosts used in the retransmission-timeout experiments

Each experiment used `ttcp` to generate the workload between the server and the receiver. When the connection sustained a lifetime long enough to ensure that the

RTO had stabilized to reflect the round-trip time, the IP interface on the receiver was disabled. This resulted in serial timeouts at the sender. We examined both the times of initial timeouts, corresponding to the original RTO and calculated from the time a retransmitted segment was originally transmitted, and the times of second timeouts, corresponding to the exponential back-off and calculated from the time of the initial timeout.

In all, one hundred connections were traced: five for each combination of round-trip time and implementation. The results for each combination were averaged. The traces were generated using `tcpdump`, and the receive-buffer size was 64KB.

3.2.2 Results

The first and second timeouts are shown in Tables 3.2, 3.3, 3.4, and 3.5, for Linux 2.2, AIX 4.3, FreeBSD 2.7, and SunOS 5.6 respectively.

RTT (msecs)	**First Timeout (msecs)**	**Second Timeout (msecs)**
60	193	400
100	196	400
200	312	640
500	680	1380
1000	1340	2680

Table 3.2: Linux 2.2 retransmission timeouts

Of the four implementations, Linux exhibits the most aggressive behavior. Linux 2.2 implementations have a lower-bound of 200 msecs on the RTO and have a heartbeat-timer granularity of 10 msecs. Because of these, and because Linux does not conform to the specification of the RTO calculation, the initial timeouts for small round-trip time connections occur after 200 msecs. Even with a 500-msec RTT,

RTT (msecs)	First Timeout (msecs)	Second Timeout (msecs)
60	1334	1505
100	1780	1602
200	1860	2000
500	2422	2000
1000	2410	2500

Table 3.3: AIX 4.3 retransmission timeouts

RTT (msecs)	First Timeout (msecs)	Second Timeout (msecs)
60	1126	2000
100	1621	3000
200	1487	3000
500	1843	2000
1000	2343	2000

Table 3.4: FreeBSD 2.2.7 retransmission timeouts

RTT (msecs)	First Timeout (msecs)	Second Timeout (msecs)
60	394	799
100	389	800
200	403	840
500	764	1430
1000	1286	2580

Table 3.5: SunOS 5.6 retransmission timeouts

the initial timeouts occur after only 680 msecs. Successive timeouts with Linux are coupled closely to the doubling of the RTO. The SunOS implementation also exhibits aggressive behavior, and appears to have a timer granularity of 200 msecs.

The AIX implementation, on the other hand, has the most conservative behavior. With a small round-trip time connection, initial timeouts occur after 1.3 seconds, growing to 2.4 seconds with larger round-trip time connections. AIX's timer granularity is 500 msecs.

FreeBSD performs like AIX, but we noticed strange behavior with second timeouts. With 100 and 200 msec RTTs, the second timeouts occur three seconds after the first timeout. However, with the larger round-trip times, the second timeouts occur only two seconds later. We did not investigate the reasons for this.

3.2.3 Discussion

These experiments show that timeout behavior is very implementation-dependent, and is highly variable. Depending on the sender, a connection with a small round-trip time can be re-started very quickly, or very slowly, by a retransmission timeout. The behavior of a sender can affect the performance on mobiles differently. Linux's "broken" retransmission behavior can cause problems for congested network queues [34], but can benefit connections recovering from handoffs: if the handoff time is short, the connection will re-start very quickly. Other implementations, however, can leave connections recovering from handoffs idle unacceptably long. If the amount of data to transmit is small, a long wait can impact the overall transfer rate greatly.

Successive timeouts can cause extremely long communication pauses. If losses occur at an untimely instant, and if the handoff time is long enough to *just* miss the arrival of the first retransmission, up to three seconds can elapse before the next retransmission timeout occurs.

3.3 The Effects of Congestion Control

If a handoff causes packet losses, the sender will eventually incur a retransmission timeout. After the timeout, slow-start and congestion avoidance throttle the rate for a long time at possibly well below the capacity of the network.

The main reason congestion control can last unacceptably long for connections recovering from handoffs is because of the congestion-avoidance phase. Slow-start is fast, and it can grow the transmission rate rapidly, but congestion avoidance grows the congestion window very slowly. In many implementations, that growth is only one segment-size every other round-trip time. In some cases, we found that AIX increases the congestion-window size approximately once every four round-trip times during congestion avoidance.

The length of congestion avoidance, in comparison to the length of slow-start, depends on the slow-start threshold (`ssthresh`). The threshold determines when slow-start ends and congestion avoidance begins, and it is set to half the congestion-window size when a retransmission timeout occurs. Therefore, even if the transmission rate is near its peak when a handoff occurs, slow-start will recover only half that rate before congestion avoidance is invoked. If the handoff occurs when a connection is in the initial slow-start phase, the threshold can be set to a very small value. In such a case, congestion avoidance is invoked early in the recovery phase. This becomes a serious problem for a sender having a mistaken perception of congestion.

If the handoff and retransmission timeout occur when the congestion window is at its maximum size possible, the extent to which congestion control affects performance depends on what that size is. If the maximum size possible is large, a long time can elapse before the sender is again transmitting at the optimal rate, and many segments that should have been transmitted are not. If the size is small, on the other hand, the congestion-window size is recovered quickly, and performance degradations are less significant.

The rest of this section quantifies the degradations in performance that are caused by the incorrect invocation of congestion control. First, through two experiments, we show the effects of slow-start and congestion avoidance with a receive-buffer size of 13 segments. In one experiment, we emulate a handoff when the sender reaches the maximum transmission rate. In the other, a handoff is emulated when the sender is in the initial slow-start phase of the connection. We look at how the transmission rate is affected, and we make comparisons to a mobile-aware sender.

Second, we simulate congestion control at a sender that uses different receive-buffer sizes. We look at the number of round-trip times that an incorrect invocation of congestion control adds to the lifetime of a connection, and we compare those results to simulations of a sender that does not invoke congestion avoidance, and to simulations of a sender that does not invoke any congestion-control mechanisms.

In the simulations, we assume that the sender's transmission rate is limited by the receive-buffer size, and not by the capacity of the network.

3.3.1 An Analysis with Experiments

The network configuration we used for the two experiments is shown in Figure 3.4. Both sender and receiver ran Linux 2.2, and a host was placed between them to delay traffic and achieve a one-second round-trip time. A handoff was emulated at the delayer by dropping an entire window of segments. In the first experiment, a window was dropped after the transmission rate peaked, and in the second, the second window of segments during slow-start was dropped. The workload was generated using `ttcp`, the receive-buffer size was approximately 18KB (or 13 segments), and the initial slow-start threshold was greater than the receive-buffer size.

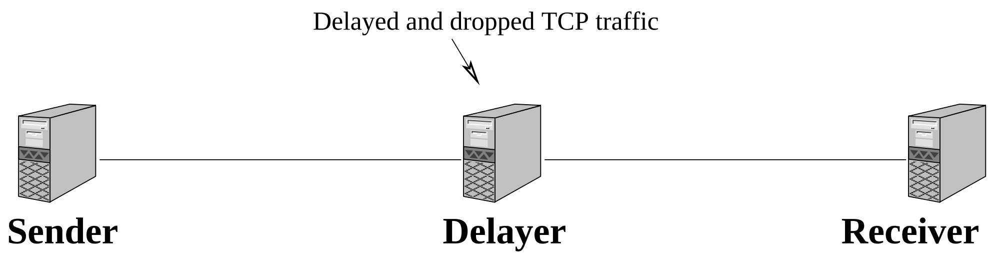

Figure 3.4: Network for analyzing congestion control

Figure 3.5 and Table 3.6 show the events of the first experiment. The connection quickly reaches the maximum transmission rate at time 31.95. The window of segments transmitted nine seconds later is dropped, emulating the handoff. After a pause, a retransmission timeout occurs at time 41.36 that re-starts the connection, and that reduces the slow-start threshold to approximately 9KB. Because of this, slow-start lasts only three round-trip-times and congestion-avoidance lasts 12 round-trip-times. The congestion window recovers its maximum size at time 56.35. During the 15 round-trip-times after the retransmission timeout, 120 segments are transmitted and acknowledged—75 less than what would have been transmitted had no losses happened. Recovering that difference adds 5.76 (75 divided by 13) round-trip times to the lifetime of the connection.

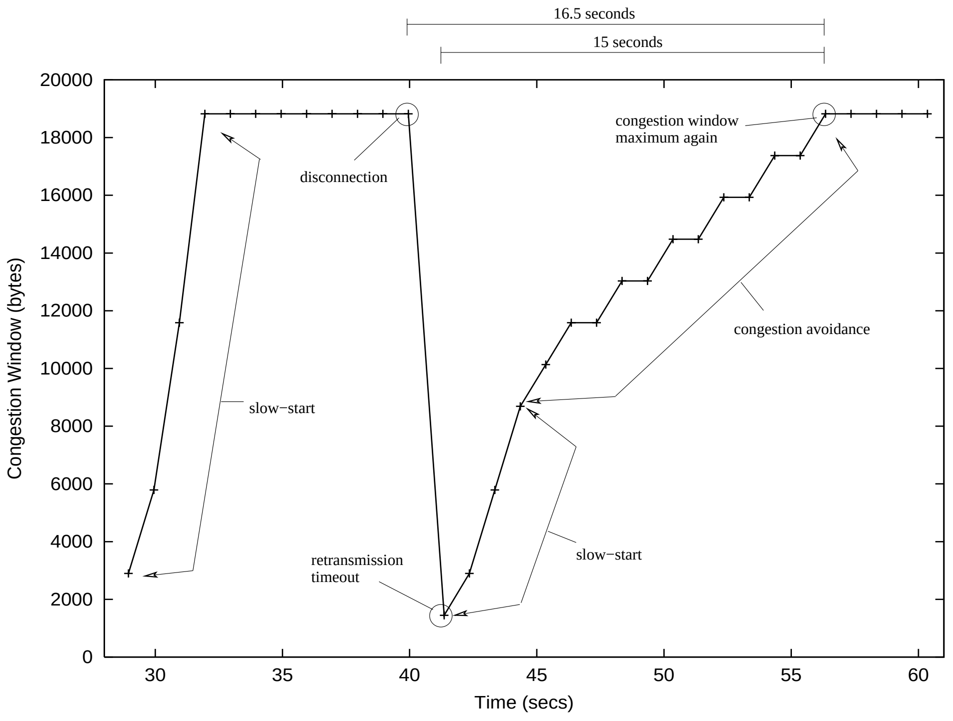

Figure 3.5: Handoff emulation when transmission rate is maximum

The events of the second experiment are shown in Figure 3.6 and Table 3.7. The window of segments in the second round-trip-time is dropped, emulating the handoff. Although it is recommended that initial RTO values be 3 seconds [35], this sender waits an unusually long 6 seconds before incurring a retransmission

Time (secs)	**CWIN (bytes)**	**Event**
28.95	2896	slow-start
31.95	18824	slow-start ends and transmission rate is maximum
39.95	18824	window of segments is dropped
41.36	1448	retransmission timeout and slow-start
42.35	2896	ACK for the retransmission
44.35	8688	slow-start ends and congestion avoidance begins
56.35	15928	congestion avoidance ends

Table 3.6: Sequence of events corresponding to Figure 3.5

timeout. Because the congestion window is small when the timeout occurs, the slow-start threshold is reduced to only 3KB. As a result, slow-start ends after only one round-trip-time, and 17 round-trip-times elapse in congestion avoidance before the transmission rate reaches its maximum. During slow-start and congestion avoidance, only 124 segments are transmitted and acknowledged, adding 9.46 round-trip times to the connection's lifetime. Note that the dip shown in the congestion window at time 12.21 does not indicate that the window shrank, but rather illustrates an implementation detail of Linux which prevents a sender in these circumstances from transmitting new data until all segment losses are recovered.

Time (secs)	**CWIN (bytes)**	**Event**
03.72	2896	slow-start
04.72	5792	window of segments is dropped
10.21	1448	retransmission timeout and slow-start
11.21	2896	ACK for retransmission and congestion avoidance begins
29.30	18824	congestion avoidance ends

Table 3.7: Sequence of events corresponding to Figure 3.6

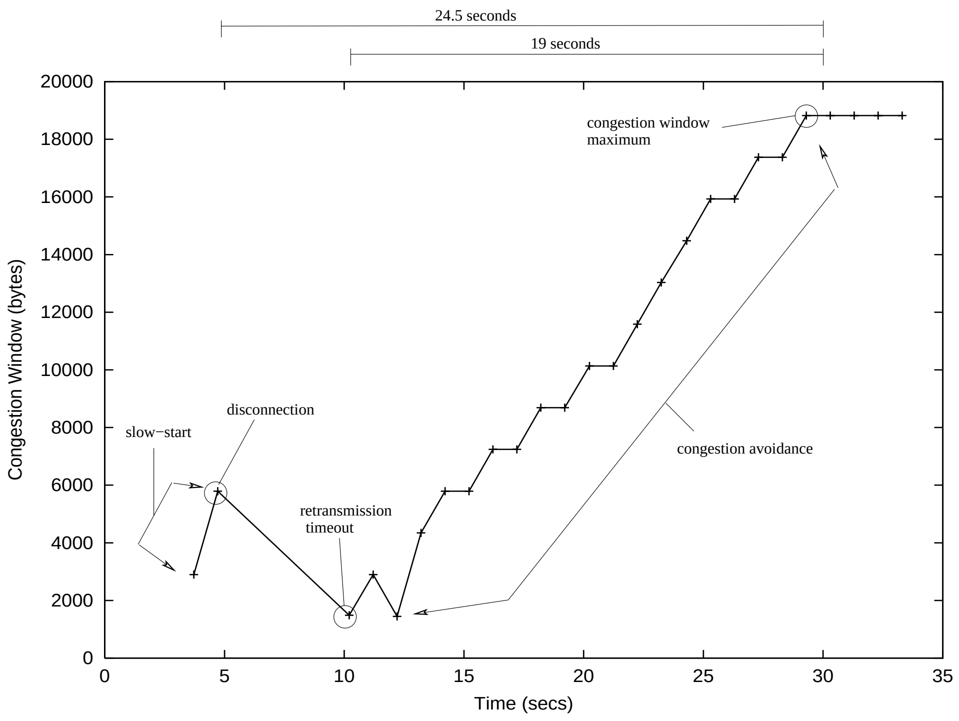

Figure 3.6: Handoff emulation during slow-start

Even with a small receive-window size of 18KB, these experiments show that congestion control can continue long after a retransmission timeout, and show that round-trip times are unnecessarily added to the lifetime of a connection. If senders are somehow implemented to recognize movement-induced losses, however, slow-start and congestion avoidance can be avoided and new algorithms can possibly take their places. One solution is to extend the slow-start phase to bypass the congestion-avoidance phase [28]. In such a solution, the recovery process would be quick and degradations in performance would be less noticeable. Caution must be exercised, however, because adjacent wireless cells can have extremely different network characteristics.

If the sender in our experiments had bypassed congestion avoidance with slow-start, the results would have been much different. In the first experiment, 158 segments instead of 120 would have been transmitted and acknowledged during the 15 round-trip-times of recovery—a 32% increase in the transmission rate. In the second experiment, 210 segments instead of 124 segments would have been acknowledged during recovery—a 70% increase in the transmission rate.

Another solution to quick recovery from handoffs is to resume with the old rate of transmission once a retransmission timeout re-starts the connection. This solution bypasses both slow-start and congestion avoidance. A disadvantage, however, is that a closely-spaced burst of segments is forced into a network that could have become congested during the handoff [21].

3.3.2 An Analysis with Simulations

In order to simulate properly a sender's behavior during congestion control, and in order to estimate, given the receive-buffer size, the number of round-trip times that an incorrect invocation of congestion control adds to the lifetime of a connection, a few assumptions must be made:

- a retransmission timeout always occurs when the congestion window is maximum, or equal to the receive-buffer size;
- the slow-start threshold is always half the receive-buffer size; and
- during congestion avoidance, the sender increases the congestion window by one segment-size every other round-trip time.

With those assumptions in mind, we estimate the added round-trip times by calculating the number of segments transmitted during slow-start and congestion avoidance, subtracting that sum from the number of segments that could have been transmitted in the same time frame, and dividing the difference by the receive-buffer size.

To begin, we calculate the number of segments transmitted during congestion control. Because slow-start begins at one segment-size, and because the growth is exponential, the sum of segments transmitted during slow-start is simply the addition of powers of 2. To calculate the number of segments transmitted during congestion avoidance, we add all of the segments transmitted between the slow-start threshold and the receive-buffer size. Therefore, the number of segments transmitted during congestion control can be expressed as

$$n\_segs\_cc = \sum_{i=0}^{\lceil \log_2 ssthresh \rceil} 2^i + \sum_{i=ssthresh, i<rsize} 2i$$

where `rsize` is the receive-buffer size, and `ssthresh` is approximately half the receive-buffer size, or

$$ssthresh = \lfloor rsize/2 \rfloor$$

To calculate the number of segments that would have been transmitted if no losses occurred, we must first calculate the number of round-trip times that elapse during congestion control. In slow-start, the number of round-trip times is equal to the number of exponential increases that occur in the congestion window, and in congestion avoidance, the number of round-trip times is equal to the slow-start threshold multiplied by two, or simply the receive-buffer size. Therefore, the number of round-trip times in congestion control can be expressed as

$$nrtts = \lceil \log_2 ssthresh \rceil + rsize$$

and the total number of segments that could have been transmitted is then $nrtts * rsize$. Therefore, the number of added round-trip times because of an incorrect invocation of congestion control is calculated by subtracting `n_segs_cc` from $nrtts * rsize$, and dividing the result by the receive-buffer size:

$$added\_rtts = (nrtts * rsize - n\_segs\_cc)/rsize$$

With minor modifications to above equations, we can also calculate the added round-trip times when congestion avoidance is bypassed, and when both slow-start and congestion avoidance are bypassed. In the latter scenario, sub-optimal performance occurs only during the initial round-trip time in which the only segment transmitted is the one from the retransmission timeout.

Figure 3.7 shows the results of simulations with receive-buffer sizes of up to 256KB. As the buffer size grows, the number of round-trip times that are lost because of slow-start and congestion avoidance increases quickly. With a buffer size of 256KB, an incorrect invocation of congestion control will add over 70 round-trip times to the lifetime of a connection. Even with a modest buffer size of 64KB, over 20 round-trip times are lost. In the graph, the abrupt increases after buffers sizes that are powers of 2 are due to an extra round-trip time being added to slow-start.

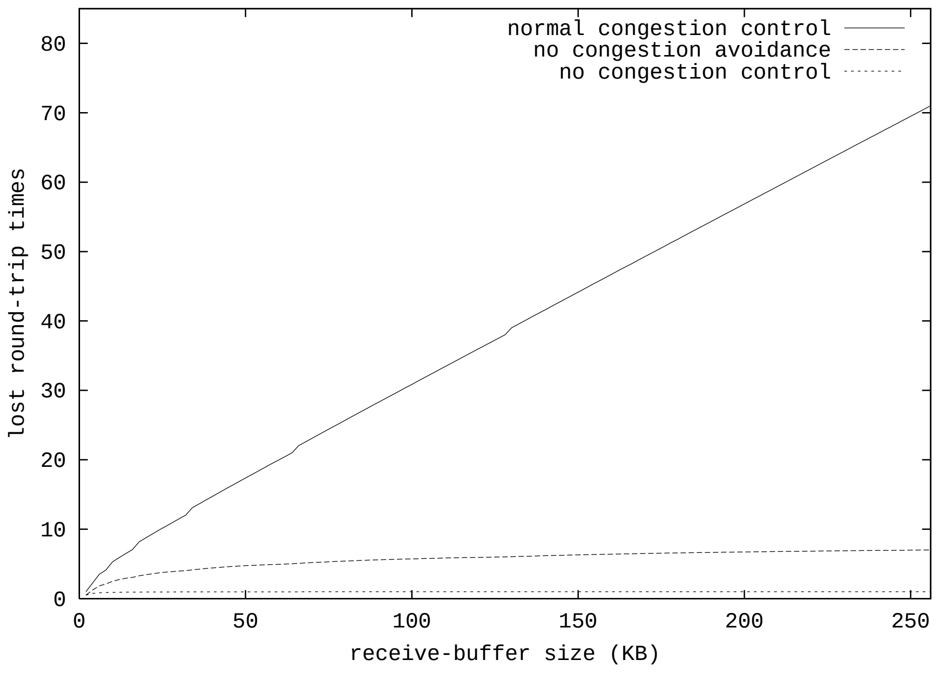

Figure 3.7: The effects of congestion control

When congestion avoidance is bypassed by slow-start, the number of lost round-trip times decreases dramatically. The differences are a clear indication that congestion avoidance, and the re-evaluation of the slow-start threshold, affect performance very negatively. When the buffer size is 64KB, only 5 round-trip times are lost, and when the size is 256KB, only 7 round-trip times are lost. Those represent improvements of 75% and 90% respectively over a sender that invokes congestion avoidance.

When no congestion-control mechanisms are invoked after a retransmission timeout, the decrease in performance is negligible and only adds one round-trip time at most to a connection's lifetime.

These simulations not only show the dramatic impact on performance when congestion control is invoked incorrectly, but show that the performance of a TCP connection can be significantly improved by not reducing the slow-start threshold when losses are movement-induced. If senders are implemented to recognize movement-induced losses, special mechanisms can be used for re-evaluating the threshold, and the threshold need not be reduced by the full amount. This is discussed further in Chapter 6.

Another point worth mentioning is that previous research has shown typical web transfers to be between 4KB and 64KB [5, 6], and that the full effects of congestion control are realized only if there is a large amount of data to transfer. Therefore, a typical web transfer that is affected by a handoff will end soon after a retransmission timeout and early into the congestion-control phase, and the effects of congestion control should be negligible. The effects should be noticeable, however, during the transfer of larger files, such as high-resolution images or multimedia applications.

3.4 Chapter Summary

This chapter has shown that TCP connections can be affected very negatively for a long time after a handoff completes. Retransmission timeouts can cause long periods of unnecessary inactivity, and congestion control can throttle the transmission rate for long after a retransmission timeout occurs. One result is very clear: unnecessary pauses and incorrect invocations of congestion control can add many round-trip times to the lifetime of a connection. In the next chapter we look at how connections are affected by Mobile-IP handoffs.

Chapter 4

TCP Performance with Mobile IP

4.1 Overview

If a TCP connection is affected by a handoff, the sender will suffer a long period of time where the transmission rate is effectively halted. During the actual disconnection, or handoff, any segments that are transmitted will not reach the mobile. After the handoff completes, transmission is idle until the sender incurs a retransmission timeout.

The purpose of this chapter is to quantify how long Mobile IP handoffs leave connections halted, and to examine how that impacts overall transfer rates.

Mobile IP handoffs are inherently long for three reasons. First, mobility agents send agent advertisements very infrequently. This is to avoid overloading wireless links unnecessarily; base stations send advertisements more frequently because link-layer, or micro, handoffs occur more often [46]. Second, there can be long propagation delays between the home network and the foreign network. Those delays can add a considerable amount of time to the registration process. The third reason why Mobile IP handoffs are long is because of the registration process itself. The mobile must be authenticated, tunnels must be configured, and routes must be updated.

The unnecessary wait for a retransmission timeout can also be very long. A short wait will often be the result of the coincidental occurrence of a retransmission timeout soon after a handoff completes. However, depending on the RTO calculation and on the sender implementation, the wait can be extended significantly.

The rest of this chapter is organized as follows. In the next section, we show how long Mobile IP handoffs are for the best-case, worst-case, and average-case scenario. In Section 4.3, we describe experiments we ran with Mobile IP. The experiments quantify how long a connection can lie idle unnecessarily after a handoff. To gain a strong understanding of the effects of handoffs under varying circumstances, we varied round-trip times and sender implementations in the experiments. Sections 4.4 and 4.5 discuss the results and the overall impact of handoffs. The chapter ends with a discussion of related work, and a summary.

4.2 Handoff Times

A handoff in Mobile IP begins when the mobile migrates out of one network and into another, and ends when registration is complete at the mobile. For our purposes, however, we consider the handoff to be complete when the home agent transmits the registration-reply message. This indicates that the home agent is ready to forward datagrams to the new location, and the forwarded datagrams will reach the mobile as long as registration is successful at the other end.

In most cases, the dominant factor in the handoff time is how long it takes for the mobile to realize that it has migrated, and to realize that it should register a new care-of-address. As discussed in Section 2.2, one of three movement-detection mechanisms is used to detect migration and to discover a new foreign agent. In this section, we look at the handoff times of Eager Cell Switching and Lazy Cell Switching. The Prefix Matching strategy is known to have very similar performance to that of LCS [18].

For each movement-detection mechanism, we consider three scenarios: the best-case, worst-case, and average-case handoff time. Again, a handoff starts when the mobile first migrates, and ends when the home agent sends the registration reply. To simplify the results, the following assumptions are made:

- L = the delay for link-layer mobility;
- T = the one-way transit time from the mobile to the home agent;
- C = the software processing time of the request messages and the latency involved in performing AAA services and tunnel setup; and
- agent advertisements are sent at a rate of one per second, with a three-second advertisement lifetime.

Eager Cell-Switching Handoff Times

With ECS, the handoff time is dependent largely on how soon an agent advertisement arrives after link-layer movement completes. This is because the mobile initiates registration the moment it receives that advertisement. Because advertisements are sent once per second, the mobile can receive an advertisement either immediately after a link-layer switch, or up to a second after the switch (the wait could be more than one second because the actual transmission times of advertisements are randomized to avoid synchronization and collisions with other agent advertisements [39]).

Figure 4.1 illustrates the best-case handoff with ECS. Initially, the mobile is registered with foreign agent FA1. Near time 3.0, however, the mobile migrates into the domain of a different foreign agent, FA2. Immediately after this, the mobile receives an advertisement from FA2. Because the mobile is using ECS, a registration-request message is sent once that advertisement is processed. Therefore, the fastest possible handoff time is $L + T + C$.

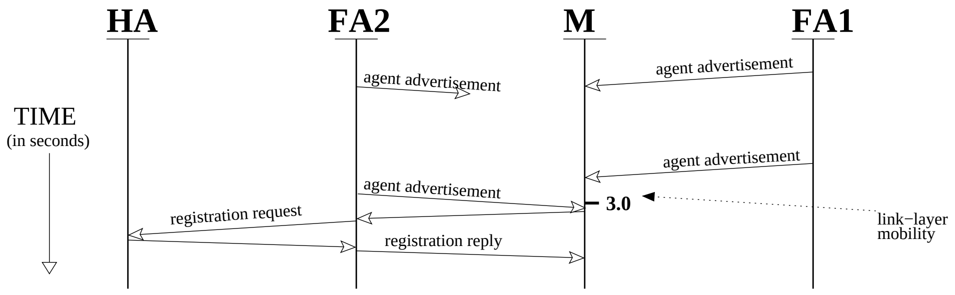

Figure 4.1: ECS best-case handoff time

At worst, a mobile can wait for 1000 msecs after link-layer movement before an agent advertisement arrives. Figure 4.2 illustrates this scenario. Therefore, the ECS worst-case handoff time is approximately 1000 $msecs + L + T + C$.

On average, the mobile receives an agent advertisement 500 msecs after link-layer mobility. The ECS average-case handoff time is approximately 500 $msecs + L + T + C$.

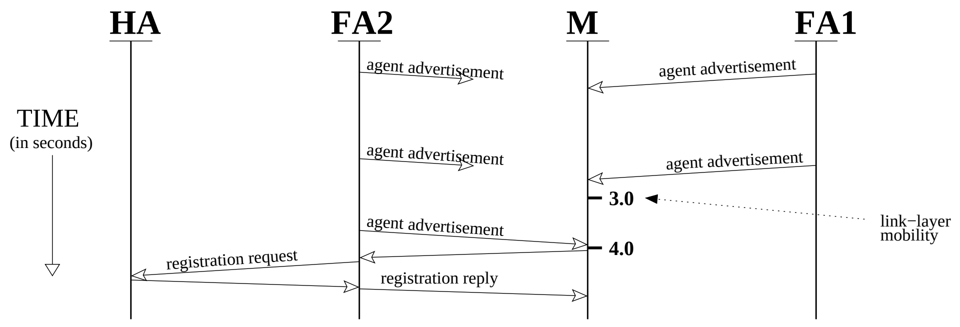

Figure 4.2: ECS worst-case handoff time

Lazy Cell-Switching Handoff Times

With the LCS strategy, the mobile does not change foreign agents until expiry of the lifetime of the advertisement from its current agent.

Figure 4.3 illustrates the best-case handoff with LCS. At time 2.0, the mobile receives an agent advertisement from its current agent, FA1. Because the advertisement lifetime is three seconds, the advertisement does not expire until time 5.0—the earliest time the mobile is allowed to begin registration with a new agent. At time 3.0, immediately before another advertisement arrives, link-layer mobility begins. At time 5.0, the advertisement from FA1 expires, and the mobile begins registration with the new foreign agent. Therefore, at best, the LCS handoff time is approximately 2000 $msecs + T + C$.

Figure 4.4 illustrates the worst-case scenario using LCS. Again, the mobile receives an agent advertisement at time 2.0. However, link-layer mobility begins immediately after this, leaving the mobile idle until time 5.0. Therefore, the worst-case handoff time is approximately 3000 $msecs + T + C$.

In the average-case scenario, link-layer mobility begins approximately 500 msecs after the last agent advertisement. The mobile can receive datagrams for 500 msecs

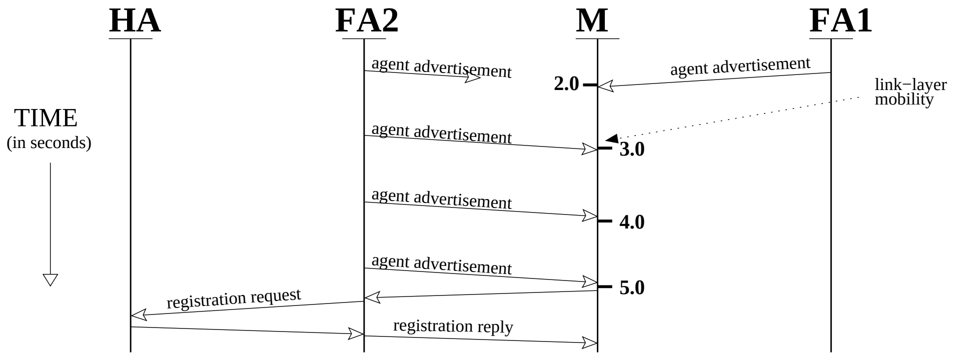

Figure 4.3: LCS best-case handoff time

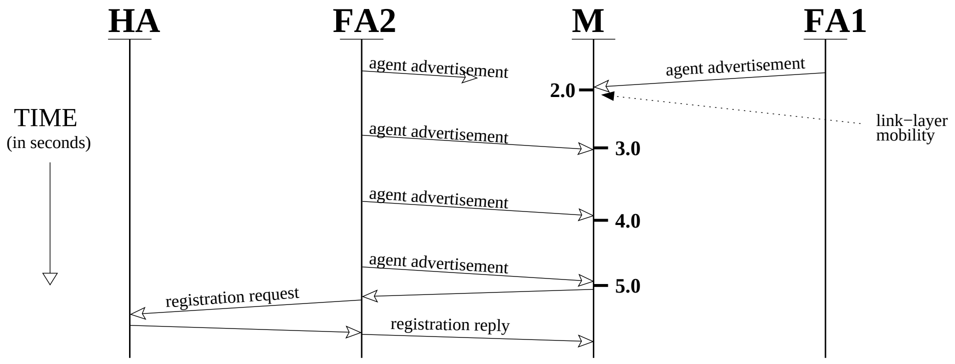

Figure 4.4: LCS worst-case handoff time

longer than in the worst-case scenario, but for 500 msecs less than in the best-case scenario. The approximate average-case handoff time using LCS is 2500 $msecs + T + C$.

Summary

Table 4.1 summarizes the handoff times for ECS and LCS. Only in the ECS best-case does the one-way transit time dominate the equation. In all other scenarios, the pause caused by the movement-detection mechanism is the dominant factor. The LCS strategy is plagued by the long wait for the advertisement to expire, which is 2000 msecs at best.

Switching	Best-Case	Worst-Case	Average-Case
ECS	$L + T + C$	$1000 msecs + L + T + C$	$500 msecs + L + T + C$
LCS	$2000 msecs + T + C$	$3000 msecs + T + C$	$2500 msecs + T + C$

Table 4.1: ECS and LCS Mobile-IP handoff times

4.3 Experiments with Mobile IP Handoffs

Using the ECS and LCS handoff times, and with a Mobile IP implementation developed by our research group, we ran a series of experiments to quantify how long those times halt connections, including the time between a handoff and a retransmission. In each experiment, we used `ttcp`, with a receive-buffer size of 18KB, to start a connection between a server and a mobile. Midway through the connection, we triggered a Mobile IP handoff. If packet losses occurred, we tracked the retransmission timeouts at the server, and we quantified how long the connection remained idle unnecessarily after the handoff completed.

Figure 4.5 and Table 4.2 show the network configuration and the hosts we used for the experiments. The two servers, Linux and AIX, were chosen for their aggressive and conservative behaviors respectively. The round-trip time between any two hosts was negligible, and the delayers were in place to emulate round-trip times of 60, 200, 500, and 1000 msecs between the mobile and the servers. Those times were chosen for the same reasons as given in Section 3.2. For simplicity, we did not vary the round-trip time during a connection—we are interested in generalizing results for connections with small round-trip times on average, and for connections with large round-trip times average. The second delayer emulated a round-trip time of 60 msecs between the home network and the foreign network, and the first delayer delayed packets further to achieve the larger round-trip times. All segments, acknowledgements, and registration messages were delayed by equal amounts at the delayers (e.g. one-way transit times were symmetric in both directions). Acknowledgements were reverse-tunneled.

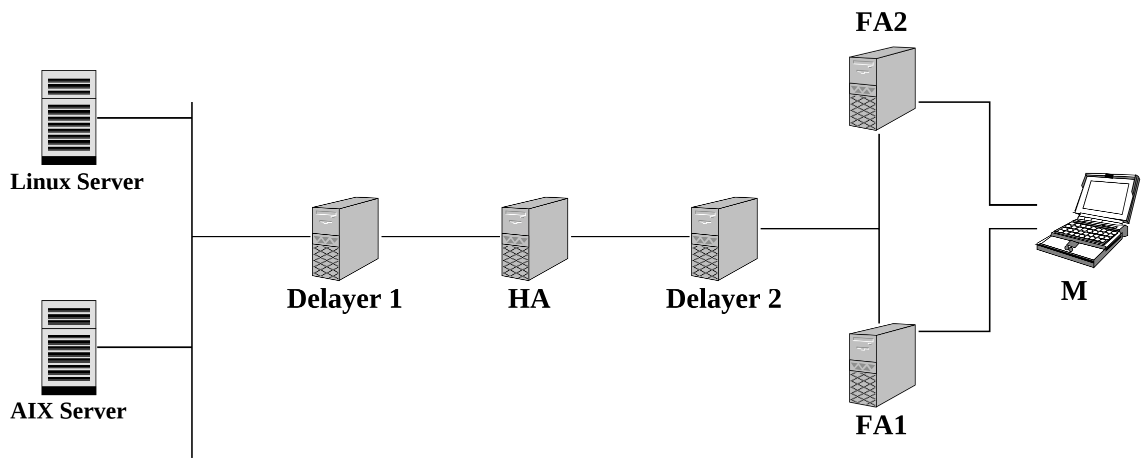

Figure 4.5: Network configuration for experiments with Mobile-IP handoffs

The mobile was connected directly to each foreign network via Ethernet, and was configured to communicate on only one network at a time. Linux bridging software made that possible. To trigger a handoff, the communicating interface was disabled, and the other one was enabled. The switch took 10 msecs, approximating an optimal link-layer switch in a wireless environment.

To emulate the different handoff times, the Mobile IP software on the mobile was modified. After the mobile switched networks, it was blocked for the amount of time necessary to achieve the desired handoff scenario. With average-case ECS,

Host	**OS**	**Architecture**	**Purpose**
Linux Server	Linux 2.2	Intel i686	TCP sender
AIX Server	AIX 4.3	IBM RS6000	TCP sender
Delayer 1	Linux 2.4	Intel i686	delays TCP traffic between HA and the TCP server
Delayer 2	Linux 2.4	Intel i686	delays TCP and UDP traffic between HA and FAs
HA	Linux 2.4	Intel i686	Home Agent
FA1, FA2	Linux 2.2	Intel i686	Foreign Agents
M	Linux 2.4	Intel i686	Mobile, TCP receiver

Table 4.2: Hosts used in Mobile-IP handoff experiments

for example, the mobility process was put to sleep for 500 msecs. Once un-blocked, the mobile immediately broadcasted an agent solicitation, forcing an agent advertisement from the new foreign agent. The latency in this message exchange was negligible.

Figure 4.3 shows the approximate handoff time for each handoff scenario. The shortest handoff time, ECS best-case, is 50 msecs. That accounts for link-layer switch delay, transit time from the mobile to the home agent, and the software overhead in processing the registration. The other handoff times also include those delays.

Figure 4.6 shows how we calculated the unnecessary idle times caused by retransmission timeouts. After the home agent sent the registration reply message, we measured the elapsed time before a retransmission arrived. That elapsed time shows how much sooner the sender should have retransmitted. As shown in the figure, even though the retransmission timeout can occur before mobility completes, mobility can complete before the retransmission arrives at the home agent.

Handoff Scenario	**Handoff Time (msecs)**
ECS Best-Case	50
ECS Average-Case	550
ECS Worst-Case	1050
LCS Best-Case	2050
LCS Average-Case	2550
LCS Worst-Case	3050

Table 4.3: Handoff times in the experiments

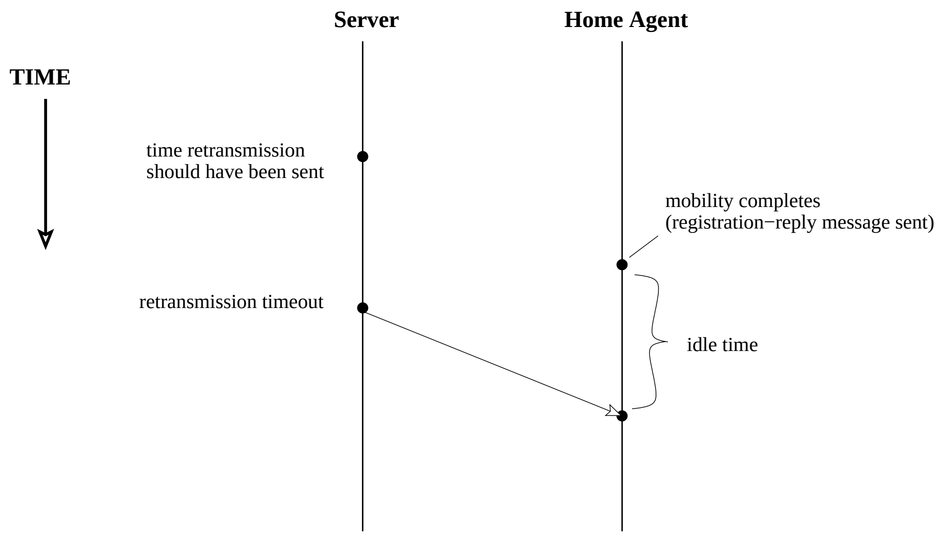
Server
Home Agent
TIME
time retransmission should have been sent
mobility completes (registration−reply message sent)
retransmission timeout
idle time

Figure 4.6: Calculating idle time

To obtain averages for each combination of handoff time, round-trip time, and server implementation, 10 handoffs were executed per combination. We computed the average number of round-trip times that elapsed during a handoff, and the average number of round-trip times that elapsed during the unnecessary idle time. Although round-trip times are variable during a connection, we believe that computing the averages in this manner reflects closely the impact a handoff has on a connection.

4.4 Results and the Overall Impact of Handoffs

The shortest handoff time had the least effect on the connections. When the round-trip time was greater than 60 msecs, a handoff time of 50 msecs very rarely caused packet losses. This was because the handoff time was a small fraction of the large round-trip time, and because the receive-buffer size was only 18KB. However, when a connection was affected, Linux and AIX reacted very differently. When the round-trip time was 60 msecs, AIX waited an average of 1449 msecs, or 24 round-trip times, more than it should have before incurring a retransmission timeout. Linux, on the other hand, waited an average of only 119 msecs, or 2 round-trip times. Combined with the handoff time, AIX suffered a loss of approximately 25 round-trip times, while Linux suffered a loss of only 3 round-trip times. That large difference shows clearly how the aggressive behavior of Linux can be beneficial when recovering from a handoff.

Figures 4.5 and 4.4 show the remaining results for Linux and AIX, respectively. For each combination of handoff time and round-trip time, the tables show the average number of timeouts, the average number of lost round-trip times during the handoffs, and the average number of lost round-trip times after the handoffs complete (Idle time). The last column of each table shows the total number of lost round-trip times.

In nearly all cases, the handoff causes packet losses, and the sender incurs at least one retransmission timeout. As the handoff time increases, so does the number of retransmission timeouts. In one case, Linux incurs 5 retransmission timeouts on average.

Handoff time	RTT	Timeouts per handoff	Handoff time (rtts)	Idle time (rtts)	Total (rtts)
550	60	2.1	9.16	2.18	11.35
550	200	2	2.7	2.46	5.21
550	500	1	1.1	0.53	1.63
550	1000	1	0.55	0.33	0.88
1050	60	3	17.5	5.68	23.18
1050	200	3	5.25	6.80	12.05
1050	500	2	2.1	2.71	4.81
1050	1000	1	1.05	0.61	1.66
2050	60	4	34.16	16.90	51.07
2050	200	3	10.25	2.35	12.60
2050	500	2	4.1	0.67	4.77
2050	1000	2	2.05	2.28	4.33
2550	60	4	42.5	7.43	49.93
2550	200	3.8	12.75	10.94	23.69
2550	500	2.7	5.1	4.08	9.18
2550	1000	2	2.55	1.77	4.32
3050	60	5	50.83	51.98	102.82
3050	200	4	15.25	11.36	26.61
3050	500	3	6.1	5.12	11.22
3050	1000	2	3.05	1.52	4.57

Table 4.4: Linux results

Handoff time	RTT	Timeouts per handoff	Handoff time (rtts)	Idle time (rtts)	Total (rtts)
550	60	1	9.16	21.22	30.38
550	200	1	2.75	6.55	9.30
550	500	1	1.1	3.23	4.33
550	1000	.5	0.55	0.74*	1.29
1050	60	1.1	17.5	8.27	25.77
1050	200	1	5.25	5.68	10.93
1050	500	1	2.1	2.28	4.38
1050	1000	1	1.05	1.75	3.80
2050	60	2	34.16	13.13	47.30
2050	200	1.5	10.25	3.46	13.71
2050	500	1.3	4.1	1.61	5.71
2050	1000	1.1	2.05	0.71	2.76
2550	60	2	42.5	16.75	59.25
2550	200	2	12.75	8.11	20.86
2550	500	1.9	5.1	2.70	7.80
2550	1000	1.3	2.55	0.83	4.38
3050	60	2.8	50.83	41.83	92.67
3050	200	2.4	15.25	8.79	24.05
3050	500	2	6.1	1.78	7.88
3050	1000	2	3.05	1.86	4.91

* only trials with retransmission timeouts are used in the average

Table 4.5: AIX results

The number of retransmission timeouts also shows the difference between Linux and AIX. While AIX incurs less than two timeouts on average, Linux incurs more than two on average. Only in a few cases does Linux incur less than two timeouts. Although Linux does not reduce the slow-start threshold aggressively, some implementations halve the threshold with each successive retransmission timeout, and a large number of retransmission timeouts would then have a dramatic impact on the length of congestion control.

In both implementations, the length of time the sender is idle unnecessarily varies. At worst, Linux waits over 50 round-trip times after the handoff completes before incurring a retransmission timeout. Even in the optimal scenarios where the handoff times are short and the round-trip times are small, the idle period can last several round-trip times. In some cases, the idle time produces a larger effect than the handoff time. In one case with AIX, for example, 6.55 round-trip times are lost after a handoff, while only 2.75 round-trip times are lost during the handoff.

The total number of round-trip times lost is large in nearly all scenarios. Even with a small round-trip time, seconds can elapse before a connection is re-started. In the worst cases, both implementations lose approximately 100 round-trip times on a connection. Losing that number of round-trip times can have a devastating impact on the overall transfer rate.

4.5 Discussion

In general, the impact of these communication pauses on the overall transfer rate depends on the round-trip time. When a connection has a small round-trip time, it is expected to have a very high rate of transfer per second. If a handoff occurs, that rate per second can be lowered significantly, and the connection can last much longer than expected. The impact is greatest when the amount of data to be transferred is small.

Connections with large round-trip times, on the other hand, should be affected less noticeably by handoffs. When the average rate of transfer per second is very low, a handoff has less of an impact on that rate. But, again, if the amount of data to be transferred is small, the overall transfer rate can drop significantly.

A final point worth mentioning is congestion control. Aside from implementations that re-adjust the slow-start threshold with each successive timeout, the length of congestion control is independent of the disconnection time. However, if a large amount of data must be transferred, the effects of congestion control can be much worse than the effects of a communication pause. If a connection must transfer a large amount of data, and if a handoff occurs midway through that transfer, congestion control can throttle the rate for a very long time. Short transfers are less affected because they end soon after a retransmission timeout occurs.

Table 4.6 shows an example of the number of round-trip times lost with Linux when congestion control is included in the equation. Even with these modest buffer sizes, connections lose well over 20 round-trip times, on average, because of a handoff. In most cases, the largest portion of the lost round-trip times is incurred in congestion control. As mentioned earlier, however, files transferred on the web are typically between 4KB and 64KB in size, and those transfers would end long before congestion control could have a serious effect on the overall transfer rate.

Handoff time	RTT	Extra rtts 16KB buffer	Extra rtts 32KB buffer	Extra rtts 64KB buffer	Extra rtts 128KB buffer
550	60	18.41	23.38	32.37	49.36
550	200	12.27	17.24	26.23	43.22
550	500	8.69	13.66	22.65	39.64
550	1000	7.94	12.91	21.9	38.89
1050	60	30.24	35.21	44.2	61.19
1050	200	19.11	24.08	33.07	50.06
1050	500	11.87	16.84	25.83	42.82
1050	1000	8.72	13.69	22.68	39.67
2050	60	58.13	63.1	72.09	89.08
2050	200	19.66	24.63	33.62	50.61
2050	500	11.83	16.8	25.79	42.78
2050	1000	11.39	16.36	25.35	42.34
2550	60	56.99	61.96	70.95	87.94
2550	200	30.75	35.72	44.71	61.7
2550	500	16.24	21.21	30.2	47.19
2550	1000	11.38	16.35	25.34	42.33
3050	60	109.88	114.85	123.84	140.83
3050	200	33.67	38.64	47.63	64.62
3050	500	18.28	23.25	32.24	49.23
3050	1000	11.63	16.6	25.59	42.58

Table 4.6: Linux results with buffer sizes of 16KB, 32KB, 64KB, and 128KB

4.6 Comparisons to Related Work

In the most relevant research, Fikouras *et al* [18] also studied the impact of Mobile IP handoffs on TCP performance. They found that, at worst, the handoff times with ECS and LCS are approximately 2.6 seconds and 6 seconds respectively. They also concluded that TCP connections can suffer idle periods after handoffs of approximately 6.5 seconds with LCS, and 3.4 seconds with ECS. In their experiments, the round-trip time between the home agent and the mobile was approximately 20 msecs. There was no indication of the round-trip time of the TCP connections.

The results of that study, however, do not reflect *typical* worst cases accurately; the experiments had several limitations:

- After movement was detected by the mobile, the Linux implementation on the mobile suffered a two-second configuration delay when changing the IP default route.
- IP datagrams originating at the TCP sender had the *don't fragment* flag marked. The TCP retransmission arriving after a completed handoff was dropped because the tunnel maximum transmission unit (MTU) between the home agent and the new foreign agent was less than the previous MTU. This caused an ICMP *Datagram Too Big* message to be sent to the sender. The dropped retransmission resulted in yet another timeout at the sender, increasing the idle period after a handoff significantly.
- The Mobile-IP software on the mobile operated in granularities of one second. Therefore, an advertisement could be delayed at the mobile for nearly one second before being processed by the Mobile-IP process.

These limitations would not occur in a real mobile environment. The two-second configuration delay on Linux occurs because the kernel flushes the routing cache at two-second intervals. This delay is eliminated by modifying the file *min_delay* of the *proc* file-system on Linux. Also, it is unreasonable to assume that a mobile should detect and respond to events at one-second intervals. The process for mobility should block when waiting for events, and un-block promptly when an important event occurs, such as the arrival of an agent advertisement. Regarding the tunnel MTU problem, this should occur only if adjacent networks have different MTUs. If adjacent cells did not have identical MTUs, the dropped retransmission for MTU discovery would become a regular phenomenon. This can have a devastating impact on the transfer rate of a connection, and would motivate network implementators to have identical MTUs in adjacent cells.

Our experiments result in a more realistic evaluation of TCP performance with Mobile IP. In the experiments, there are no unnecessary configurations that could cause long delays in the kernel or Mobile-IP process, and adjacent cells have identical MTUs. Also, the experiments considered a variety of scenarios by using different handoff times, round-trip times, and server implementations.

4.7 Chapter Summary

This chapter presented the overall effects of Mobile IP handoffs on TCP connections. In a wide variety of scenarios, we have shown that connections can be affected very negatively by a handoff. Although a connection is halted during an actual handoff, it will often remain idle long after the handoff completes. That idle period is unnecessary, and it can lower the overall transfer rate significantly.

In the next chapter, we discuss the future role of Mobile IP and propose solutions for improving performance after handoffs complete.

Chapter 5

TCP Enhancements for Wide-Area Mobility

5.1 Overview

Mobile IP is simple and scalable, and is the network-layer-mobility protocol supported by the IETF. The frequent use of Mobile IP for handoffs, however, has serious performance implications. Because the handoffs are inherently long, and because TCP connections can be affected very negatively, frequent handoffs can cause very unpredictable and unreliable performance on mobiles. For those reasons, Mobile IP should not be used every time a mobile changes locations.

Some of the current research in network-layer mobility focuses on solutions that very rarely use Mobile IP for handoffs [38, 46, 52]. These solutions, called *micro-mobility* solutions, assume that most mobility is confined to limited geographic areas, and that all mobility functions should be handled within those areas. The areas, or *wide-area networks*, can be as large geographically as a university campus or a metropolitan city-centre.

Micro-mobility solutions use sophisticated protocols that set up and maintain mobile-specific routes within the wide-area network. In the networks, a foreign agent is at the root of a hierarchical structure of base stations, customized routers, and other mobility-support stations. When a mobile moves from one base station to another, routes are updated quickly and packets in transit are re-directed to the new base station. The protocols provide fast and lossless handoffs, and avoid the problems with network-layer handoffs.

When a mobile migrates from one wide-area network to another, however, Mobile IP is needed. The mobile must acquire a new care-of-address and must be authenticated. Figure 5.1 illustrates the distinction between *micro-mobility*, which is mobility within a wide-area network, and *macro-mobility*, which is mobility between wide-area networks.

To provide a TCP-lossless handoff in Mobile IP, segments must be cached at the old location and re-directed to the new location after the handoff completes. This can be done either by the old foreign agent, or by a mobility-support station at the edge of the old network. The foreign agent, for example, temporarily caches segments. If informed that the mobile has moved, the foreign agent forwards the cached segments, and any subsequent segments, to the new location.

This scheme, however, has two major drawbacks. First, scalability is a problem. Wide-area networks are large, in part, to optimize transport-layer performance, and mobility-support stations may be servicing tens of thousands of mobiles. If there are

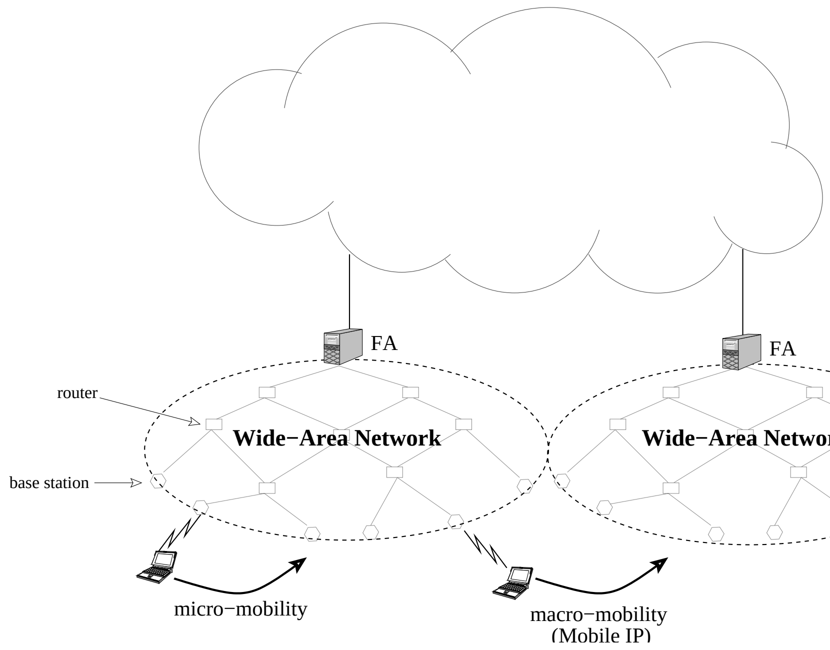

Figure 5.1: Micro-mobility and macro-mobility

also tens of thousands of TCP connections, it is almost impossible for those stations to have sufficient resources to monitor and cache all of the segments. Furthermore, the resources used could have been used to meet other requirements, and the overall performance within the wide-area network can be affected.

The second major drawback is that there is no guarantee that the resources used will justify the performance gained. Regardless of whether segments are cached, a handoff can often be long enough to cause a sender to incur a retransmission timeout, and a timeout causes congestion control. And if congestion control is invoked, a cached window of segments may save only one round-trip time for the connection. One large benefit, however, is that by forwarding segments immediately after a handoff, the long pause caused by a retransmission timeout is eliminated.

In this chapter, we present simple and scalable solutions that can provide the same benefits as described above, but that do not require large data caching. Rather than trying to prevent retransmission timeouts that are often unavoidable, we cache only a single segment, and use that segment to re-start a connection as soon as the handoff completes. This eliminates the communication pauses caused by timeouts, and can save many round-trip times for a connection.

The rest of the chapter is organized as follows. In the next section, we discuss work related to improving transport-layer performance in mobile environments. In Section 5.3, we show how an intermediary host can monitor TCP connections, cache retransmissions, and forward the retransmissions when handoffs complete. Section 5.4 shows how we force a sender into persist mode when a handoff is impending and, when the handoff completes, to re-start the connection immediately. In that scheme, the sender does not incur retransmission timeouts and, in some implementations, congestion control is prevented. Section 5.5 summarizes the chapter.

5.2 Related Work

This section presents related work that involves caching, buffering, and manipulating segments. Most of that work improves performance when handoffs are between base stations, and it is included here in order to provide a complete picture of possible solutions. To our knowledge, little work is specific to Mobile IP handoffs.

Routing Protocol (1995)

Developed with Snoop [8] at the University of California at Berkeley, Routing Protocol [47] provides lossless and low-latency handoffs between base stations. A mobile is assigned a temporary IP-multicast address by its home agent, and each base station within range of the mobile joins that multicast group. Segments destined for the mobile are encapsulated, either by the home agent or by the sending host (in the case of Mobile IP route optimization [40]), and sent to the multicast address. Base stations in the group that are not servicing the mobile buffer the segments temporarily. Therefore, when a handoff occurs, the new base station has a copy of the segments that would have been lost during the handoff.

Routing Protocol has three major drawbacks. First, buffering segments at various base stations presents a scalability problem. If the base stations are servicing many mobiles, or if the mobiles have large receive buffers, buffering requires great storage capacity. Accessing and storing the segments can also be complex.

The second drawback is that multicasting traffic can degrade performance within a domain. Unless appropriate care is taken, resources can be used inefficiently: if a base station joins the multicast group too soon, resources may be used well ahead of when they are needed. Also, it is possible that a buffering base station never actually services the mobile, resulting in an unnecessary use of resources.

The third major drawback is that managing the multicast groups is difficult because address management must be handled across the Internet.

DFA Hierarchical Mobility Management (1999)

This management approach [50] is very similar to Routing Protocol, but the multicast group is managed from a domain foreign agent, or DFA, and not from the home agent or the sending host. The foreign agent administers an aggregation of subnets and base stations, and forwards segments to the multicast address. Although this approach eliminates the problems associated with multicasting management across the Internet, it shows the same inefficiency and scalability problems of Routing Protocol.

I-TCP (1995)

I-TCP [7] is a split-connection solution. When a mobile requests a TCP connection, the base station splits the connection into two separate connections: one between the server and the base station, and the other between the base station and the mobile. The base station acknowledges data on behalf of the mobile, and recovers the losses caused by handoffs. In this way, mobility is shielded from the sender.

A major drawback of I-TCP is that it requires large resources to handle connection states, and the transferring of states between base stations can be complex and time-consuming. Another drawback is that end-to-end TCP semantics are compromised; because a base station can acknowledge data that the mobile has not received, if failure occurs, the base station must terminate or re-set the connection at the sender.

Hierarchical Mobility Management (1996)

In this scheme [14], the base station servicing a mobile caches unacknowledged segments. When the mobile migrates, the new base station sends a message to the old base station, and the old base station forwards the cached segments to the new location. Although acknowledgements clear the cache, the size of the cache can quickly grow large during a handoff. This becomes a scalability problem when the number of mobiles serviced is large also.

M-TCP (1997)

M-TCP [11] is an entirely different approach for improving TCP performance. In M-TCP, if migration is detected, the base station, on behalf of the mobile, sends a zero-window-size acknowledgement. This forces the sender into persist mode. When the handoff completes, the base station, or the mobile, sends an acknowledgement that re-opens the send window. That re-starts the connection quickly and, in some implementations, prevents congestion control.

Because a sender disregards window sizes in duplicate acknowledgements, however, the zero-window-size message sent must acknowledge new data. To achieve this, M-TCP buffers acknowledgements, and if migration is detected through the absence of expected acknowledgements, a buffered acknowledgement is changed to a zero-window-size acknowledgement and forwarded to the sender. To prevent the sender from stalling, the base station holds back only the last byte of each acknowledgement. When a new acknowledgement arrives, the byte is acknowledged normally, and the next last byte is buffered. Special mechanisms are used to ensure that the last acknowledgement from a segment burst is not delayed.

M-TCP is a scalable solution and can recover losses quickly. However, breaking acknowledgements can cause re-packetization delays at the sender and, as we will show in Section 5.4, trying to exploit persist mode has its own disadvantages.

Freeze-TCP (2000)

Freeze-TCP [20] is similar to M-TCP, but zero-window-size acknowledgements are sent from the mobile itself. When the mobile senses an impending handoff (by analyzing base-station signal strength), all acknowledgements are changed to advertise a zero buffer. When the handoff completes, the mobile immediately sends an acknowledgement that re-opens the send window.

Freeze-TCP has an advantage over M-TCP and other solutions, if the IP payload is encrypted. In Freeze-TCP, however, complexity is moved from the base station to the mobile. That may be a difficult choice for power-conscious manufacturers of mobile devices.

The solutions we propose in the next two sections overcome some of the limitations of this related work.

5.3 Retransmission Caching

5.3.1 Enhancement Overview

The goal of this enhancement is to cache, during a handoff and at an intermediary host, the retransmission from a retransmission timeout, and to forward that retransmission when the handoff completes. As long as the sender incurs an initial and unsuccessful retransmission timeout, the connection is re-started at the earliest possible time, and there is no unnecessary pause in communication.

To perform the caching effectively, the intermediary host must distinguish between normal segments and segments from retransmission timeouts. To do this, the host monitors the highest sequence number seen. If a segment arrives that has a sequence number less than that, or out-of-order, that segment was either re-ordered by the network, duplicated by the network (which we assume is very rare), from a fast-retransmit, or from a retransmission timeout. In all cases, the segment is cached. The next course of action depends on what arrives next:

1. **A higher-sequenced segment arrives.** In this case, it is most likely that the connection is proceeding normally, and the cache is cleared.

2. **The same segment arrives.** This is a strong indication that the sender incurred a consecutive retransmission timeout. The cache is maintained, and the segment is forwarded to the mobile.

3. **A mobility-update message arrives.** This indicates that the mobile moved, and that the cached segment likely came from an initial retransmission timeout. The segment is forwarded to the mobile's new location, the cache is cleared, and the connection is re-started without waiting for the next retransmission timeout.

In the latter or third case, if the segment was not from a timeout, or if communication had already resumed before the mobility-update message arrived, forwarding the segment will at worst generate a duplicate acknowledgement.

5.3.2 Deployment

This enhancement can be deployed easily at either the home agent or the foreign agents. Only one segment, at most, needs to be cached for each TCP connection.

Home-Agent Caching

If segments en-route to the mobile pass through the home network, the home agent can carry out the caching. If a Mobile IP handoff occurs, the home agent can forward a cached segment to the new location immediately after the registration reply is sent. As long as registration is successful at the foreign network, the segment should reach the mobile soon after registration completes.

Foreign-Agent Caching

If segments en-route to the mobile do not pass through the home network, as with Mobile IP route optimization [40], the foreign agent can monitor and cache segments. This requires, however, that the old foreign agent is notified immediately of the new location of the mobile.

5.3.3 Experiments

We deployed and experimented with the home-agent caching method. A user-space process monitored the sequence numbers of segments, and cached the segments from retransmission timeouts. If a handoff occurred, and if a segment was in the cache, the cached segment was sent to the new location. If no segment was in the cache, it was likely that either no losses occurred or the initial retransmission timeout did not occur.

If caching had been used for the experiments in Section 4.3, the results would have been much different. Table 5.1 shows the possible performance gains for the connections to the Linux server. The third column shows the number of round-trip times lost because of the handoff and because of the unnecessary wait for a retransmission timeout. The fourth column shows how the lost round-trip times decrease with caching, and the fifth column shows the savings in round-trip times and in percentages. The results show that the number of lost round-trip times can be limited to the length of the handoff, and can be significantly reduced. In a couple of cases, however, caching had no effect because the initial retransmission timeout had yet to occur.

Handoff time	RTT	No caching (rtts)	Caching (rtts)	**Savings (rtts)**
550	60	11.35	9.16	**2.17 (19%)**
550	200	5.21	2.7	**2.46 (47 %)**
550	500	1.63	1.63	0
550	1000	0.88	0.88	0
1050	60	23.18	17.5	**5.68 (25 %)**
1050	200	12.05	5.25	**6.80 (56 %)**
1050	500	4.81	2.1	**2.71 (56 %)**
1050	1000	1.66	1.05	**0.61 (37 %)**
2050	60	51.07	34.16	**16.90 (33 %)**
2050	200	12.60	10.25	**2.35 (19 %)**
2050	500	4.77	4.1	**0.67 (14 %)**
2050	1000	4.33	2.05	**2.28 (53 %)**
2550	60	49.93	42.5	**7.43 (15 %)**
2550	200	23.69	12.75	**10.94 (46 %)**
2550	500	9.18	5.1	**4.08 (44 %)**
2550	1000	4.32	2.55	**1.77 (41 %)**
3050	60	102.82	50.83	**51.98 (51 %)**
3050	200	26.61	15.25	**11.36 (43 %)**
3050	500	11.22	6.1	**5.12 (46 %)**
3050	1000	4.57	3.05	**1.52 (33 %)**

Table 5.1: Caching for a Linux sender

5.3.4 Discussion

Caching retransmissions to improve performance is simple and scalable. Only one segment is cached per TCP connection, and no transfer of states is needed between agents. Furthermore, as we explained in Section 4.5, web transfers usually involve small amounts of data, and a connection gains the most benefit when no unnecessary pause occurs in communication.

The major limitation with caching retransmissions, however, is that it works only if the initial timeout occurs before mobility completes. If the timeout does not occur before then, caching has no benefit: if a segment other than one from a retransmission timeout is forwarded, the generated acknowledgement will, at most, result in the transmission of one new segment, and the long wait for a retransmission timeout will eventually occur anyways. Linux is aggressive, incurs many timeouts, and can benefit greatly. But other implementations, such as AIX, are less aggressive, and consecutive timeouts may often occur. In those implementations, the caching scheme will have little effect. The next section describes a scheme that can be used when there is no initial retransmission timeout.

5.4 Zero-Window-Size Acknowledgements

5.4.1 Enhancement Overview

To avoid the wait for the initial retransmission timeout, the sender needs acknowledgements for all lost segments. Without buffering a large amount of data during the handoff, and without seriously compromising TCP semantics, this requirement is impossible. But by sending a premature zero-window-size acknowledgement, and thereby forcing persist mode, the sender can be *tricked* into thinking that the lost segments should never have been transmitted, and therefore should not have timeouts set. Therefore, after the handoff completes, transmission can resume immediately, and normally, by sending an acknowledgement, new or duplicate. In this way, the connection is re-started quickly, and the consequences associated with retransmission timeouts are avoided. This is the strategy adopted by M-TCP and Freeze-TCP.

The difficulty with zero-window-size acknowledgements, however, is that they **must** be new, not duplicate—a sender disregards the window size in duplicate acknowledgements. M-TCP overcomes this problem by delaying a part of the last acknowledgement seen, and Freeze-TCP changes regular acknowledgements to zero-window-size acknowledgements. In both cases, new data is acknowledged.

But M-TCP and Freeze-TCP do not always work. It is possible that segments are lost and that no acknowledgement is available for a zero-window-size acknowledgement. This scenario is shown in Figure 5.2 where an intermediary host, as in M-TCP, buffers the last acknowledgement seen. But when the acknowledgement for segment 18 is buffered, no subsequent acknowledgements are expected. Therefore, to prevent the sender from stalling or waiting, and after a small amount of time elapses, the host forwards the acknowledgement. As a consequence, the buffer is empty when segments 19 through 27 are lost, and the sender eventually incurs a retransmission timeout. Because of the bursty nature of TCP, we believe this situation would occur often.

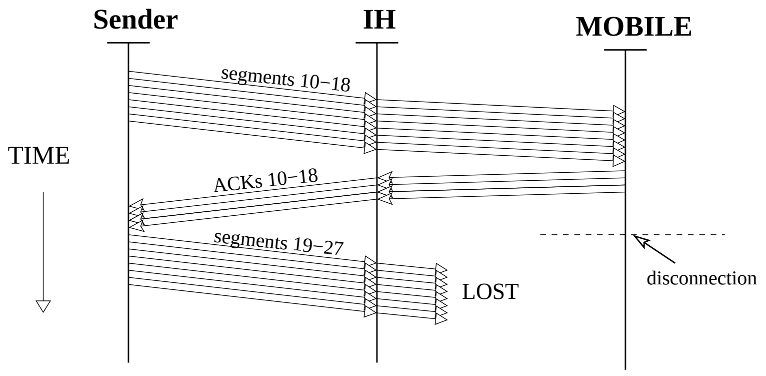

Figure 5.2: A situation where M-TCP and Freeze-TCP do not work

We propose a scheme that is similar to M-TCP and Freeze-TCP, but that overcomes their limitation described above. In the scheme, the intermediary host monitors segments and acknowledgements, and caches the *oldest* unacknowledged segment. Because the host knows when to expect acknowledgements, it can detect a handoff by their absences. If an acknowledgement is detected, and if the oldest unacknowledged segment is in the cache, a zero-window-size acknowledgement is sent for the segment. The acknowledgement is new, and the sender is forced into persist mode. When the handoff completes, the intermediary node forwards the cache to the mobile's new location, and the subsequent acknowledgement re-opens the send window.

Figure 5.3 shows the benefit of this solution. As in the previous figure, segments 19 through 27 are lost during a handoff. This time, however, the intermediary host has cached segment 19 and, when the handoff is detected, sends a zero-window-size acknowledgement for it. When the handoff completes, and when the intermediary host is notified, segment 19 is forwarded to the mobile. The subsequent acknowledgement immediately re-starts the connection.

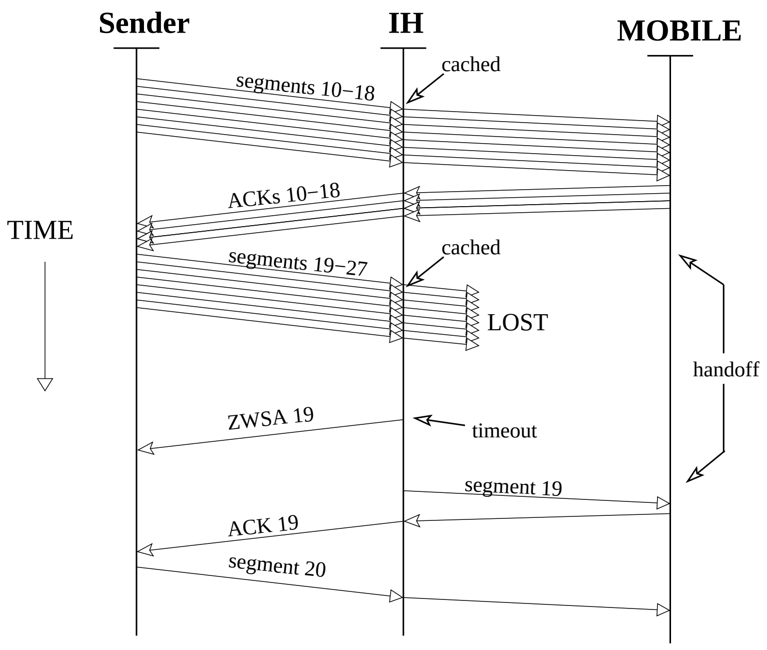

Figure 5.3: Caching a segment for a zero-window-size acknowledgement

This trivial algorithm is shown in Figure 5.4. If there are no outstanding acknowledgements, or if the cache is clear, the next segment is cached, and if a cached segment is acknowledged, the cache is cleared. If a timeout occurs on the cache, a zero-window-size acknowledgement is sent, and when a mobility-update message arrives, the cache is forwarded to the mobile.

This scheme is simple, and it will work well in most scenarios. If a handoff occurs during the arrival of a closely-spaced burst of segments, however, the algorithm can fail. Figure 5.5 shows how this occurs. Initially, segment 11 is cached, and segments 12 and 13 are forwarded normally. A handoff occurs, and although segment 11 reaches the mobile in time, segments 12 and 13 are lost. Segment 11 is now obsolete in the cache because segment 12 is the oldest unacknowledged segment. Although segment 14 is cached because the cache was cleared, it cannot be used for a zero-

```
If (segment arrived)
    If (no ACKs expected) OR (cache is clear)
        cache segment, set timer
Else If (ACK arrived)
    If (ACK >= segment cached)
        clear cache, re−set timer
Else If (timer expired)
    send zero−window−size acknowlegment
Else If (mobility−update message arrived)
    forward cache to new location
```

Figure 5.4: Algorithm for caching the oldest unacknowledged segment

window-size acknowledgement because it is not the oldest and unacknowledged.

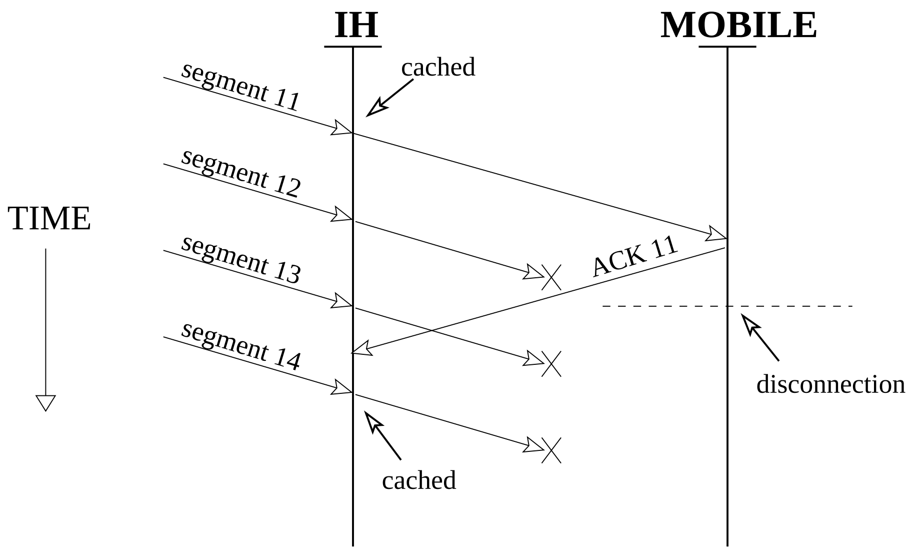

Figure 5.5: A limitation of the zero-window-size acknowledgement scheme

We believe, however, that the situation described here is very rare. In most cases, this scheme will work because handoffs are more likely to start before or after, rather than during, the arrival a very closely-spaced burst of segments.

5.4.2 Deployment

This scheme can be deployed easily at foreign agents. A process on the foreign agent can monitor a connection and cache segments. If a handoff is detected, the foreign agent sends a zero-window-size acknowledgement. As shown in Figure 5.6, a messaging protocol can be established between foreign agents, and the new foreign agent can inform the old foreign agent of the mobile's location.

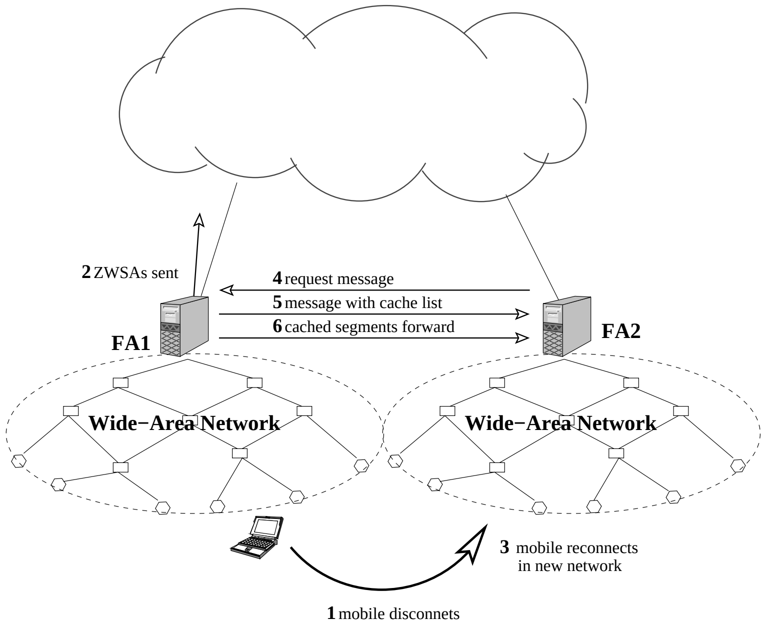

Figure 5.6: Foreign-agent deployment

The major limitation of this scheme is that end-to-end semantics are violated. If the protocol fails for any reason, and if a segment that was cached and used for a zero-window-size acknowledgement is lost, the TCP connection can enter into a seemingly infinite loop where the mobile continually responds to window probes with acknowledgements that are sequenced less than what the sender expects. In that case, the connection must be either terminated or re-set.

To ensure that the mobile receives cached segments reliably, the foreign agents can agree on a simple protocol to deliver them reliably. The old foreign agent, for example, can keep a copy of a cached segment until the new foreign agent acknowledges it, and built-in timeouts can trigger retransmissions at the old foreign agent. And if the old foreign agent fails, the new foreign agent can re-set a frozen connection on behalf of the mobile.

We have yet to develop the details of the messaging and reliability protocols.

5.4.3 Experiments

We experimented with this solution using the network configuration in Section 4.3. A host was placed between the foreign agents and the second delayer in order to cache the segments and to emulate the message-passing protocol.

The AIX server responded well to the zero-window-size acknowledgements, and the sender was forced into persist mode at every handoff. Table 5.2, taken from the results of Section 4.4, shows the performance gains possible with AIX. During the disconnection and idle time, the number of lost round-trip times when the scheme is not used is shown in the third column, and when the scheme is used, in the fourth column. In many cases, this scheme saves over 30% of the disconnection time (or 8 round-trip times), and more than one second, for the connection.

As noted by Goff *et al* [20], AIX processes premature zero-window-size acknowledgements the same way it processes mature ones, and congestion control is prevented. Because of this, not only does a connection benefit because it is re-started quickly, but it also re-starts at the rate of transmission prior to the zero-window-

Handoff time	RTT	Without ZWSAs (rtts)	With ZWSAs (rtts)	**Savings (rtts)**
550	60	30.38	9.16	**21.22 (70 %)**
550	200	9.30	2.75	**6.55 (70 %)**
550	500	4.33	1.1	**3.23 (75 %)**
550	1000	1.29	0.55	**0.74 (57 %)**
1050	60	25.77	17.5	**8.27 (32 %)**
1050	200	10.93	5.25	**5.68 (52 %)**
1050	500	4.38	2.1	**2.28 (52 %)**
1050	1000	3.80	1.05	**1.75 (46 %)**
2050	60	47.30	34.16	**13.13 (28 %)**
2050	200	13.71	10.25	**3.46 (26 %)**
2050	500	5.71	4.1	**1.61 (28 %)**
2050	1000	2.76	2.05	**0.71 (26 %)**
2550	60	59.25	42.5	**16.75 (28 %)**
2550	200	20.86	12.75	**8.11 (39 %)**
2550	500	7.80	5.1	**2.70 (35 %)**
2550	1000	4.38	2.55	**0.83 (19 %)**
3050	60	92.67	50.83	**41.83 (46 %)**
3050	200	24.05	15.25	**8.79 (37 %)**
3050	500	7.88	6.1	**1.78 (23 %)**
3050	1000	4.91	3.05	**1.86 (38 %)**

Table 5.2: AIX results with zero-window-size acknowledgements

size acknowledgement. This essentially obviates most of the devastating effects of handoffs.

Linux, however, reacts very differently. Linux disregards premature zero-window-size acknowledgements and, without it being in persist mode, connections must rely on a retransmission timeout to resume normally. We found, however, that attempting to use the scheme with Linux does not degrade performance because the premature acknowledgements are simply discarded.

In similar experiments, we found that SunOS 5.6 performed like AIX.

5.4.4 Discussion

A sender's reaction to a premature zero-window-size acknowledgement is unpredictable and entirely implementation-dependent. The specification [44] states only that the sender *should* not shrink the send window from the right edge. Therefore, some implementations will respond positively, while others will not respond at all.

If the goal of a solution is to prevent congestion control, however, future TCP implementations may not respond well. A recent proposal by Handley *et al* [21] argues that long pauses in communication invalidate the congestion window. After a long time in persist mode, for example, the sender has an out-dated estimation of congestion levels. Therefore, when the connection resumes, the sender should begin in slow-start, and should have a lower slow-start threshold. Fortunately for mobiles, this proposal decreases the threshold at a rate slower than retransmission timeouts do.

Regardless of how particular implementations react to zero-window-size acknowledgements, we found that, in general, sending these acknowledgements has no negative effects. Moreover, some implementations will benefit, while others will perform as if one of those acknowledgements was never even sent.

5.5 Chapter Summary

This chapter presented the role of Mobile IP in the future Internet, and provided a thorough examination of how we can limit the effects of Mobile IP handoffs on TCP connections. The obvious solutions suffer from scalability problems, and do not necessarily provide enough benefits to justify the resources used.

We proposed two solutions that are simple, scalable, and deployed easily. The solutions require little resources, and the algorithms are trivial. Through experiments and analyses, we showed that TCP performance after a handoff can be improved significantly, and that many round-trip times can be recovered for a connection.

Chapter 6

Summary, Conclusions, and Future Work

6.1 Summary and Conclusions

In this thesis, we explained and quantified the negative effects of mobility on TCP connections, and we showed how to limit those effects.

There are many reasons for poor TCP performance in mobile environments. TCP is geared for the characteristics of wired networks, and is tailored only for environments where congestion is the only cause of segment losses. It can perform well during congestion, and it can tune the rate of transmission to the fluctuations in network conditions. However, the reasons it performs well during congestion are the same reasons it performs poorly in mobile environments.

During normal operation, segment losses cause the sender to incur a long pause in communication, in part, to help relieve congestion. After that pause, a retransmission timeout occurs, and congestion control is invoked. In congestion control, the sender initially transmits at a very low rate, and then probes for more and more bandwidth by slowly growing that rate.

If the segment losses are the result of a handoff, however, the sender also incurs a long communication pause, assuming incorrectly that the pause might benefit the connection. After the retransmission timeout, the sender grows the transmission rate slowly, this time assuming that the network is congested and limited. Those two incorrect assumptions cause TCP performance to be seriously degraded in mobile environments.

In this thesis, we showed that a communication pause can be long, and can last long after a handoff completes. With Mobile IP for handoffs, we showed that many round-trip times can be lost during a pause. The dramatic effects of congestion control were also presented. Congestion control can last unacceptably long and, if the amount of data to transfer is large, can have a devastating impact on the overall transfer rate.

To counter these problems, we proposed two solutions that eliminate the communication pauses, and that can sometimes prevent congestion control. The solutions involve deploying simple protocols at home agents and foreign agents, and they require few resources.

In summary, to current research in the area of TCP performance in mobile environments, this thesis has contributed

1. A complete explanation for poor performance;

2. A large number of experiments, simulations, and analyses that prove and quantify poor performance; and

3. Simple and scalable solutions that can improve performance significantly during Mobile IP handoffs.

The optimal, but improbable, solution for improving performance is to implement senders who differentiate between handoff losses and congestion losses, and to tune senders for mobility in the same way they are tuned for congestion. If mobility is detected, for example, the sender can wait for a message that communicates the end of the handoff. Or, as another example, the sender can be more aggressive when growing the transmission rate during congestion control.

Although re-implementing TCP for mobility can improve performance significantly, a re-implementation is very unlikely to happen in the near future. TCP is widely deployed, extensively used, and widely agreed upon; changing it would be a very long process that could result in serious disruptions of the compatibility of hosts. For at least the next few years, solutions will have to be deployed on the mobiles and the mobility-support stations themselves.

6.2 Future Work

Opportunities exist for improvements and future work. We aim our work at a more intensive examination of the retransmission caching solution in Chapter 5, and at looking at performance with the successor of Mobile IP, Mobile IPv6 [24].

Deployment of Solutions

The solution of caching retransmissions is quite simple, can be deployed easily, and can eliminate many idle pauses in communication. These reasons justify a closer look. Retransmission caching works very well for Linux, but that is due, in part, to the fact that Linux is very aggressive. Our goal is to deploy the solution, and to test it with exhaustive variants. We intend to look at numerous implementations, handoff times, and round-trip times and, if the results are positive, deploy the solution in a real mobile environment. Positive results, however, depend largely on the portion of implementations that have an aggressive retransmission-timeout strategy.

TCP Performance with Mobile IPv6

IPv6 incorporates many features that are beneficial for mobile hosts. One of the most important is that the triangular routing of Mobile IP is eliminated. In Mobile IPv6, tunnels are set up and secured at the sending hosts themselves, and a mobile can deliver a change in address directly to the sending host.

Another important feature is that a mobile does not require the use of a foreign agent. A mobile can use the Stateless Address-Autoconfiguration [51] and Neighbor Discovery [32] mechanisms of IPv6 in order to acquire a care-of-address on a foreign network.

These new features provide new performance issues for TCP. We aim to discover those issues and find solutions for them.

Bibliography

[1] IP Routing for Wireless/Mobile Hosts (Mobile IP). IETF Working Group Charter: http://www.ietf.org/html.charters/mobileip-charter.html.

[2] The Internet Engineering Task Force. http://www.ietf.org.

[3] Wireless Application Protocol (WAP) Forum. http://www.wapforum.org.

[4] M. Allman, editor. *TCP Congestion Control*. RFC-2581, 1999.

[5] M. Arlitt, R. Friedrich, and T. Jin. Workload Characterization of a Web Proxy in a Cable Modem Environment. *ACM SIGMETRICS Performance Evaluation Review*, 27(2):25–36, September 1999.

[6] M. Arlitt and T.Jin. Workload Characterization of the 1998 World Cup Web Site. *IEEE Network*, 14(3):30–37, May/June 2000.

[7] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *15th International Conference on Distributed Computing Systems*, pages 136–143, May 1995.

[8] H. Balakrishnan, S. Seshan, and R.H. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, 1(4):469–481, December 1995.

[9] M. Boucher and R. Russell. The Netfilter Project: Packet Mangling for Linux 2.3+. http://netfilter.samba.org.

[10] R. Braden, editor. *Requirements for Internet Hosts – Communication Layers.* RFC-1122, 1989.

[11] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks. *ACM Computer Communications Review (CCR)*, 27(5):19–43, October 1997.

[12] C. Perkins. Mobile IP Joins Forces with AAA. *IEEE Personal Communications*, 7(4):59–61, August 2000.

[13] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5):850–857, June 1995.

[14] R. Caceres and V. Padmanabham. Fast and Scalable Handoffs for Wireless Internetworks. In *Proceedings of ACM Conference on Mobile Computing and Networking (Mobicom'96)*, 1996.

[15] S. Deering, editor. *ICMP Router Discovery Messages.* RFC-1256, 1991.

[16] K. Fall and S. Floyd. Simulation-Based Comparison of Tahoe, Reno, and Sack TCP. *ACM Computer Communication Review*, 26(3):5–21, July 1996.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leachm, and T. Berners-Lee. *Hyptertext Transfer Protocol – HTTP/1.1.* RFC-2616, 1999.

[18] N. A. Fikoura, K. El Malki, S. R. Cvetkovic, and M. Kraner. Performance Analysis of Mobile IP Handoffs. In *Proceedings of 1999 Asia-Pacific Microwave Conference*, pages 770–773, Singapore, December 1999.

[19] S. Glass, T. Hiller, S. Jacobs, and C. Perkins. *Mobile IP Authentication, Authorization, and Accounting Requirements.* RFC-2977, 2000.

[20] T. Goff, J. Moronski, D.S. Phatakd, and V. Gupta. Freeze-TCP: A True End-to-End TCP Enhancement Mechanism for Mobile Environments. In *Proceedings of the IEEE INFOCOM 2000*, pages 1537–1545, Israel, 2000.

[21] M. Handley, J. Padhye, and S. Floyd. *TCP Congestion Window Validation.* RFC-2861–experimental, 2000.

[22] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM '88*, pages 314–32, 1988.

[23] V. Jacobson, C. Leresd, and S. McCanne. Tcpdump. Available via anonymous ftp from ftp.ee.lbl.gov.

[24] D.B. Johnson and C. Perkins. Mobility Support in IPv6. *IETF Mobile IP Internet Draft*, 2000.

[25] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol.* RFC-2401, 1998.

[26] J. Klensin, editor. *Simple Mail Transfer Protocol.* RFC-2821, 2001.

[27] M. Lioy and J.P. Black. Providing Network Services at the Base Station in a Wireless Networking Environment. *Wireless 97. TR Labs, TRIO, IEEE Canada*, 7(4):59–61, July 1997. Calgary, AB Canada.

[28] P. Manzoni, D. Ghosa, and G. Serazzi. Impact of Mobility on TCP/IP: An Integrated Performance Study. *IEEE Journal on Selected Areas in Communications*, 13(5):858–867, June 1995.

[29] S. McCreary and K. Claffy. Trends in Wide Area IP Traffic Patterns, A View From Ames Internet Exchange. 2000. http://www.caida.org/outreach/papers/AIX0005.

[30] M. Muuss and T. Slattery. TTCP. http://www.ccci.com/learn/tools/ttcp.tar.

[31] J. Nagle. *Congestion Control in IP/TCP Internetworks.* RFC-896, 1981.

[32] T. Narten, E. Nordmark, and W. Simpson. *Neighbor Discovery for IP Version 6 (IPv6).* RFC-1970, 1996.

[33] S. Ostermann. Tcptrace. http://jarok.cs.ohiou.edu/.

[34] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[35] V. Paxson and M. Allman. *Computing TCP's Retransmission Timer.* RFC-2988, 2000.

[36] C. Perkins. *IP Encapsulation within IP.* RFC-2003, 1996.

[37] C. Perkins, editor. *IP Mobility Support.* RFC-2002, 1996.

[38] C. Perkins. Mobile-IP Local Registration with Hierarchial Foreign Agents. *IETF Mobile IP Internet Draft*, February 1996.

[39] C. Perkins. *Mobile IP: Design Principles and Practices.* Addison Wesley Longman, Inc., Reading, Massachusetts, 1998.

[40] C. Perkins and D.B. Johnson. Route Optimization in Mobile IP. *IETF Mobile IP Internet Draft*, November 2000.

[41] J. Postel. *User Datagram Protocol.* RFC-768, 1980.

[42] J. Postel. *Internet Control Message Protocol.* RFC-777, 1981.

[43] J. Postel, editor. *Internet Protocol.* RFC-791, 1981.

[44] J. Postel. *Transmission Control Protocol.* RFC-793, 1981.

[45] J. Postel and J. Reynolds. *File Transfer Protocol (FTP).* RFC-959, 1985.

[46] R. Ramjee, T. La Porta, S. Thuel, K. Varadhan, and S.Y. Wang. HAWAII: A Domain-Based Approach for Supporting Mobility in Wide-Area Wireless Networks. In *Proceedings of Seventh Annual International Conference on Network Protocols, ICNP '99*, Toronto, ON Canada, November 1999.

[47] S. Seshan, H. Balakrishnan, and R. Katz. Handoffs in Cellular Wireless Networks: The Daedalus Implementation and Experience. *Kluwer Journal on Wireless Personal Communications*, January 1996.

[48] T. Shepard. Xplot. Available from ftp://mercury.lcs.mit.edu/pub/shep/.

[49] W. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols.* Addison Wesley Longman, Inc., Reading, Massachusetts, 1994.

[50] C.L. Tan, K.M. Lye, and S. Pink. A Fast Handoff Scheme for Wireless Networks. *Second ACM International Workshop on Wireless Mobile Multimedia*, pages 83–90, August 1999. Seattle, WA USA.

[51] S. Thomson and T. Narten. *IPv6 Stateless Address Autoconfiguration.* RFC-1971, 1996.

[52] A.G. Valko. Cellular IP: A New Approach to Internet Host Mobility. *ACM Computer Communication Review*, pages 50–65, January 1999.