TCP Enhancements for Heterogeneous Networks[1]

Stephen M. West Nitin H. Vaidya

Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
Phone: (409) 845-0512

E-mail: {swest, vaidya}@cs.tamu.edu

Technical Report # 97-003

April 9, 1997

[1] Research reported is supported in part by Texas Advanced Technology Grant 009741-052-C (also known as 999903-052).

Abstract

Heterogeneous networks are composed of a mixture of wired and wireless links. Current transmission control protocol (TCP) implementations have been designed to work well in networks made up exclusively of wired links. In this environment, error rates are very low and packet losses may be attributed almost entirely to network congestion. The proper response for a TCP sender when packets are lost in this situation is to decrease throughput in order to prevent the network congestion from becoming more severe. Unlike wired links, wireless links often have significant error rates, and therefore the assumption that all losses are due to congestion is no longer valid for heterogeneous networks. Since current TCP implementations are unable to distinguish between wireless and congestive losses, they use a conservative approach and still assume that all losses are due to congestion. This approach can often cause unnecessary reductions in throughput, and may result in lower goodput due to retransmission of packets that have already been successfully received. The focus of this project has been to review in more detail what packet loss situations cause TCP to perform poorly and how wireless and congestive losses may be differentiated in order to improve TCP's performance. The test set-up for the project consisted of three personal computers running FreeBSD with the first PC acting as a fixed host, the second as a base station, and the third as a mobile host. The fixed host and base station were connected by a LAN, and the base station and mobile host were connected by wireless network cards. Single error and burst error situations were created on the wireless link using a Poisson-distributed bit error model. Results such as throughput and goodput were then measured using a packet filter at the mobile host and a set of customized TCP trace code at the sender.

Table of Contents

1.0 Introduction to The Problem

Most of the reliable transport layer protocols in existence today were designed when networks were composed solely of wired links. In the time period since these protocols were initially designed, networks have begun to shift from traditional wired systems to heterogeneous systems composed of a mixture of wired and wireless links. Recent annual growth rates in wireless applications have been between 35 and 60 percent per year, and it is anticipated that over 100 million users will have some version of wireless personal communications system (PCS) by the end of the century [1]. As this trend towards wireless systems continues, it becomes more and more important to insure that the transport layer protocols being used are tuned to perform well both in the traditional wired networks and in heterogeneous networks.

Reliable transport layer protocols like TCP (Transmission Control Protocol) which are in use today have been specifically tuned to work well in traditional wired networks. In this environment, error rates are well below one percent and packet losses may be attributed almost entirely to network congestion[2]. The proper response for a TCP sender when packets are lost in this situation is to decrease throughput in order to prevent a congestive collapse in which useful network throughput grinds to a halt. Unlike traditional networks, wireless systems may have intermittent periods of high error rates[3]. Heterogeneous networks therefore have a mixture of congestive losses and wireless losses due to errors. Existing TCP implementations are unable to distinguish between these two types of losses, so they use a conservative approach and assume that all losses are due to congestion. This approach ensures proper response to congestion, but can result in congestion prevention measures being invoked unnecessarily when errors occur on a wireless link. These measures in turn may cause reductions in throughput, and may result in retransmission of packets that have already been successfully received.

Two different classes of solutions have been proposed to improve the performance of transport protocols such as TCP for bulk data transfers within heterogeneous networks. The first class adds functionality to existing transport protocols so that senders are able to distinguish between congestive and wireless losses. The second class attempts to hide wireless losses from the TCP sender. Hiding the losses may be accomplished in several ways including the use of split connections, reliable wireless link-layers, and forward error correction. Specific implementations of both classes along with their advantages and disadvantages will be covered in more detail in section two of the paper.

1.1 Overview of The Project and Paper

The work performed for this project and presented in the rest of the paper can be divided into three major stages. The first stage which is presented in section two of the paper involves understanding how packet losses due to errors or congestion affect the performance of existing TCP implementations. Also, a review of some of the recently proposed methods for improving TCP in heterogeneous networks is presented. TCP was chosen for this study because it is widely

used, has many characteristics that are typical of a reliable transport layer protocol, and it is easy to obtain TCP source code for experimentation. Also, because of the widespread deployment of TCP, it is likely that in the near term incremental upgrades to TCP will be favored over entirely new transport protocols.

The second stage of the project involved creating an environment that would allow Dr. Vaidya and those of us working with him to modify the TCP/IP protocol stack, run experiments with the modified code, and measure the results. This stage, which is highlighted in section four of the paper, took the most time because it required a detailed understanding of certain portions of the TCP/IP protocol stack and the FreeBSD operating system. A set of three personal computers were used as the test system with one functioning as a fixed (wired) host, another as the base station (wired to wireless router), and a third as the mobile (wireless) host. The ability to generate controlled bit error and burst error situations was then added to the operating system kernel on these machines. Next, methods were devised to obtain performance metrics at the sender and receiver. Then, application programs were written to transfer test data. Finally, a configuration program was written which allowed the error model parameters and the TCP enhancement features to be changed inside the kernel without recompiling the kernel's source code.

The third stage of the project involved making modifications to the TCP/IP portion of the FreeBSD kernel in order to improve performance. The changes made to the kernel are highlighted in section four of the paper and discussed in detail in the appendices. The first step involved taking an implementation of the Berkeley snoop protocol (see section 2.6) which was designed to run on the BSDI operating system and porting it over to the FreeBSD operating system. This provided a useful way to become familiar with the FreeBSD kernel and to learn techniques others had used when adding enhancements to TCP. Also, since the Berkeley snoop protocol offers a significant performance improvement over existing TCP implementations on heterogeneous networks, it served as a good baseline for comparison when testing future enhancements. After the snoop protocol was functioning properly, a partial acknowledgment protocol [6] proposed by Dr. Vaidya and those of us working with him was implemented. This protocol, which is covered in the third section of the paper, is intended to be used at base stations that are already running snoop. Its performance is reviewed in section five of this paper.

2.0 Background and Summary of Related Work

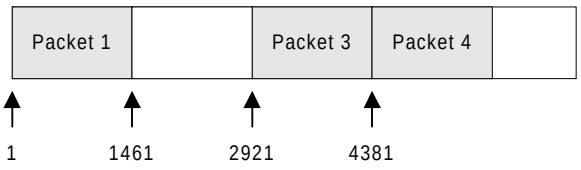
This section first takes a closer look at basic TCP functionality. Then it investigates how TCP reacts to congestion and why this reaction can result in poor performance when errors are interpreted as congestion. Finally, it covers various approaches which have been proposed to improve the performance of TCP in heterogeneous networks. All of these new proposals are backwards compatible and many of them use options negotiation to determine whether the enhancements will be used or not. If either the sender or the receiver does not understand the option, the enhancement is not used.

2.1 Basic TCP Functionality

TCP was designed to function as a connection oriented transport layer protocol capable of operating on top of an unreliable network layer which may lose packets or deliver packets out-of-order. It in turn delivers a reliable, sequenced stream of bytes from one end of the connection to the other. TCP obtains reliability through the use of positive acknowledgments (ACKs) with retransmission. TCP also uses a variable sized sliding window to accomplish flow control and to allow efficient utilization of network bandwidth. The receiver dictates the usable window size so that it can regulate the flow of data from the sender in a manner that doesn't overfill its buffers. The sender is then able to transmit up to a full window's worth of packets before requiring an acknowledgment. The window size needed to make efficient use of available bandwidth increases with the bandwidth of the connection, and the delay in the path. The available bandwidth multiplied by the delay is referred to as the delay-bandwidth product.

The acknowledgments used by TCP are cumulative and provide the sender with the sequence number of the next packet that the receiver expects. ACKs are only sent in response to a packet being received rather than at specific intervals. Thus, if no packets are arriving, no ACKs will be sent. Cumulative acknowledgments are efficient in the sense that an ACK does not have to be sent for every packet received. They are also ambiguous because they do not explicitly inform the sender of any packets which have been lost or damaged. Thus, in Figure 1 below, when packet one arrives, an ACK with a sequence of 1461 will be sent. When packets three and four arrive, an ACK with a sequence of 1461 will be sent again. The sender can not tell from these cumulative acknowledgments that packet three and four were successfully received.

Figure 1 - Receiver's Window (Packet 2 Lost)



The time between when a packet is sent and when its ACK arrives back at the sender is called the round-trip-time (RTT). The RTT is measured by TCP and used to calculate a value for the retransmission timer. The retransmission timer is set when a packet is transmitted and if a time-out occurs at the sender before an ACK for the packet is received, the packet is sent again. This feature ensures reliability because it allows TCP to detect losses and recover from them.

2.2 TCP with Congestion Control

This subsection covers the two TCP versions which introduced congestion control measures and are prevalent in the Internet today. While they handle congestion in a way that results in network stability, their reaction to congestive losses can result in less than optimal throughput even in the

absence of wireless errors. Their throughput becomes even lower with a mixture of congestive and wireless losses since they can not distinguish between the two cases.

2.2.1 TCP Tahoe

TCP Tahoe refers to a version of TCP introduced in the BSD operating system in 1988. This was the first TCP implementation to include the congestion control mechanisms and round-trip-timing enhancements proposed by Van Jacobson in his paper "Congestion Avoidance and Control"[2]. These new algorithms were introduced in response to congestive collapses which began occurring on the Internet in 1986 and caused throughput to drop in some cases by a factor of a thousand. The goal of these mechanisms is to ensure that a TCP connection is able to reach a state of equilibrium and that the connection obeys the "conservation of packets principle" once it is in equilibrium. This principle states that once a connection has reached equilibrium, it should only transmit a packet on the network when it receives feedback indicating that a packet has left the network. The connection reaches equilibrium by probing the network for available bandwidth and adjusting a newly proposed sender congestion window. In TCP Tahoe, the window used by the sender is taken as the minimum of the receiver window and this new congestion window.

The first of the mechanisms added by TCP Tahoe is Slow-Start. This algorithm is invoked when a connection is first established or anytime a packet loss is detected. Its purpose is to ensure that the connection actually reaches equilibrium. It sets (reduces) the sender's congestion window to a single packet in size. In the Slow-Start phase, the sender's congestion window is increased by one packet for each ACK received. Slow-Start is somewhat of a misnomer since the congestion window is actually being expanded exponentially each round-trip-time. Without Slow-Start, a new connection will send an entire window of data all at once which may overwhelm intermediate gateways and lead to a cycle of dropped packets and retransmission.

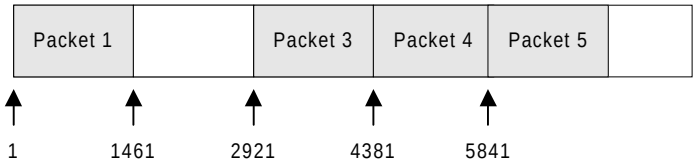
The second mechanism added by TCP Tahoe is an improved method for round-trip-time estimation. The earliest versions of TCP used a smoothed round-trip-time estimator which was a simple low-pass filter with a constant RTT variance factor (beta) of two. The problem with this algorithm is that it adapts slowly to large changes in round-trip-times and this can cause the RTT to be underestimated. This in turn will cause the retransmission timer to expire even though the original packet has not been lost. The packet will then be sent a second time with the first packet still in the network. This wastes bandwidth and can lead to congestion because it violates the conservation of packets requirement. The new RTT algorithm proposed by Jacobson uses a simple estimate of the round-trip-time variance rather than a constant beta factor of two. This allows it to more rapidly adjust to large timing changes and prevents erroneous timer expirations. Jacobson also showed that for proper stability an exponential backoff should be used when a retransmission timer expires and the packet is resent.

The third mechanism added by TCP Tahoe is Congestion-Avoidance. The purpose of this algorithm is to ensure that a sender cuts its throughput in half when a loss occurs since the loss is assumed to be due to congestion. Although Congestion-Avoidance is a separate concept from Slow-Start, they are implemented as a single procedure in practice. When a loss occurs, a

threshold variable called *ssthresh* is set to half of the congestion window or receiver's window - whichever is smaller. Then the congestion window is set to one to initiate Slow-Start. The connection stays in the Slow-Start phase and increases the congestion window by one packet for every ACK received until the congestion window reaches *ssthresh*. Then the connection enters the Congestion-Avoidance phase in which it increases the congestion window by one packet for each full window of data successfully transmitted and acknowledged. This approach results in exponential decreases and linear increases.

The final mechanism introduced in TCP Tahoe was Fast-Retransmission. If a packet arrives out-of-order (i.e. before the packet ahead of it in the sender's transmission sequence) it means that either it took a faster path or the packet before it was lost. The Fast-Retransmission algorithm requires the receiver to immediately send an ACK when an out-of-order packet is received. When packet one arrives in Figure 2 below, an ACK will be sent for sequence number 1461. Packets three, four, and five are considered out-of-order packets and because ACKs are cumulative, the sequence number acknowledged in each case will be 1461. A TCP Tahoe sender equipped with FAST-Retransmission will interpret three duplicate ACKs in a row as an indication of a packet loss. It will retransmit the missing packet and invoke the Slow-Start and Congestion-Avoidance measures. The algorithm is called Fast-Retransmit because it allows the sender to recover from a packet loss without waiting for the retransmission timer to expire.

Figure 2 - Receiver's Window (Duplicate ACKs)



Advantages of Tahoe

Tahoe has the most basic congestion and loss mechanisms of all the modern TCP implementations. However, the Fast-Retransmit algorithm introduced with TCP Tahoe may provide the single most significant performance improvement when packets are lost due to congestion. Fast-Retransmit is often able to detect packet losses in a matter of several milliseconds on a LAN and within roughly a hundred milliseconds on a WAN. Without Fast-Retransmit, the sender is forced to wait for a retransmission timer to expire. Since these timers are designed to provide a relatively loose upper bound, they often have values on the order of several seconds for even the fastest of LANs. Therefore, Fast-Retransmit can save on the order of several seconds each time a loss occurs and this translates into enormous gains in throughput.

Disadvantages of Tahoe

While Fast-Retransmit makes Tahoe perform drastically better than a TCP implementation whose sole means of loss detection is retransmission timers, it obtains significantly less than optimal performance on high delay-bandwidth connections because of its initiation of Slow-Start (which

TCP Reno discussed below avoids). Also, in the case of multiple losses within a single window, it is possible that the sender will retransmit packets which have already been received[7].

2.2.2 TCP Reno

TCP Reno refers to a version of TCP introduced in the BSD operating system in 1990. It has all of the features of TCP Tahoe plus a new algorithm called Fast-Recovery which builds on the Fast-Retransmit mechanism. With Fast-Recovery, three duplicate ACKs are still interpreted as a packet loss due to congestion, but the fact that these duplicate ACKs made it to the sender means congestion is moderate rather than severe. If congestion were severe, the majority of the packets would be lost and it is unlikely that three duplicate ACKs would be received. Therefore, the sender retransmits the lost packet, and invokes the Congestion-Avoidance algorithm, but does not initiate Slow-Start. This effectively causes the sender to cut its throughput in half without performing the initial ramp up.

In TCP Reno, the Fast-Recovery phase lasts from the time the loss is detected, until an ACK for the retransmitted packet is received. The hardest part of the algorithm to understand is a temporary inflation of the congestion window which occurs while in the Fast-Recovery phase. Each duplicate ACK received at the sender indicates that another packet has left the network and according to the conservation of packets concept, a new one may be introduced. Therefore, the congestion window which was cut in half upon entering the Fast-Recovery phase is immediately increased by three packets to account for the three initial duplicate ACKs. Then, each subsequent duplicate ACK causes the congestion window to increase by one packet in size. When the ACK for the retransmitted packet arrives, this temporary inflation is removed and the congestion window is again half the size it was prior to the packet loss. This temporary window inflation allows the connection to continue to make progress by maintaining the clocking relationship between the ACKs coming back and the data being sent. Without this temporary inflation, the congestion window would fill up, and the connection would stall.

Advantages of Reno

The key advantage to TCP Reno over TCP Tahoe is that it introduces a way to maintain the clocking of new data with the duplicate ACKs. This allows TCP to directly cut its throughput in half without the need for a Slow-Start period to reestablish the clocking between data and ACKs. This improvement has the most noticeable effect on long delay-bandwidth connections where the Slow-Start period lasts longer and large windows are needed to achieve optimal throughput. On a wireless LAN where losses are due to errors, the Slow-Start period is very short, and even the smallest of congestion windows makes efficient use of available bandwidth.

Disadvantages of TCP Reno

The Fast-Recovery mechanism introduced by TCP Reno handles multiple packet losses within a single window poorly. This can lead to situations where TCP Reno must wait for a retransmission timer to expire before continuing and is discussed further in the next section.

2.3 End-to-End Improvement Methods

End-to-end improvements attempt to reduce TCP's susceptibility to packet losses without making changes to any of the intermediate nodes (i.e. routers, and base stations). All upgrades are performed at the sender and the receiver. In many cases this is desirable because it may be impossible or impractical to modify the intermediate nodes. This is especially true when base stations are owned by a service provider, and the mobile and fixed hosts are owned by the user. These approaches do not sacrifice end-to-end reliability and require no state maintenance at intermediate nodes. A further advantage is that they provide the same performance improvements regardless of whether the sender is the mobile host or the fixed host.

2.3.1 TCP New-Reno

TCP New-Reno is an experimental version of TCP Reno proposed by Hoe[8] which makes a slight modification to the Fast-Recovery algorithm of TCP Reno. Unlike the Reno Fast-Recovery phase which ends when the resent packet is acknowledged, the New-Reno Fast-Recovery phase ends when all of the data is acknowledged which was outstanding at the time the loss was detected. In the case of a single packet loss, Reno and New-Reno will perform identically. However, if two or more packets are lost in a single window of data, Reno will invoke Fast-Recovery multiple times while New-Reno will handle the multiple losses with a single Fast-Recovery phase.

Advantages of TCP New-Reno

Avoiding multiple executions of Fast-Recovery in rapid succession allows New-Reno to eliminate two detrimental effects. First, it prevents the connection from shrinking the congestion window too drastically (i.e. by factors of four or more). This can provide substantial improvements for high delay-bandwidth paths requiring large window sizes. Second, and more importantly, by not shrinking the congestion window as drastically, it can eliminate the stalls which often happen to TCP Reno after two packets are dropped in a single window.

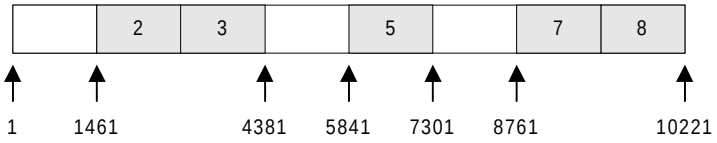
Disadvantages of TCP New-Reno

While TCP New-Reno is capable of handling multiple packet losses in a single window, it is limited to detecting and resending at most one lost packet per round-trip-time. This deficiency becomes more pronounced as the delay-bandwidth becomes greater. More importantly, there are situations where stalls can still occur if packets are lost in successive windows. Also, like all of the previous versions of TCP discussed above, New-Reno still infers that all lost packets are caused by congestion and it may therefore unnecessarily cut the congestion window size when errors occur.

2.3.2 Selective Acknowledgments (SACKS)

Selective acknowledgments are an idea which has been around for at least a decade, but have seen a great deal of renewed interest in the past year or two. The main idea with any of the proposed SACK schemes is to provide the sender with more detailed information about the state of the receiver than is possible with cumulative ACKs. In 1996, Mathis, Mahdavi, Floyd, and Romanow proposed the MMFR SACK standard for use in the Internet[9]. This proposal allows a TCP receiver to send SACK information as a set of options within the TCP header which is complimentary to the existing TCP ACK. The new option fields indicate the starting and ending sequence of non-contiguous sets of data existing at the receiver. Depending on the other TCP options being used by the connection, a maximum of either two or three blocks of data may be reported by a single ACK. The receiver shown in Figure 3 below has just received packet number eight and the information sent in the SACK will be: first block = 8761 to 10221, second block = 5841 to 7301, and third block = 1461 to 4381.

Figure 3 - Receiver's Window (Multiple Losses)



Receivers include SACK information in the TCP header only when duplicate ACKs are sent in response to the arrival of an out-of-order packet. A new scoreboard matrix is introduced at the sender to keep track of this SACK data, and a new pipe mechanism is used to track the number of packets currently in transit (i.e. in the "network data pipe"). SACK performs Fast-Retransmit just like New-Reno. It enters the Fast-Retransmit phase when a loss is detected, and it exits when all of the data has been acknowledged which was outstanding when the Fast-Retransmit phase began.

Advantages of SACK

SACK improves upon TCP New-Reno when multiple packets are lost in a single window. The additional information provided by SACK allows the sender to retransmit multiple lost packets within a single round-trip-time. New-Reno has to wait for the ACK of the first retransmitted packet to determine which (if any) other packets have been lost [7].

Disadvantages of SACK

SACK still makes no attempt to distinguish between losses due to congestion and errors on the wireless link (it just decreases the effects losses have on performance). Also, it seems that with the currently proposed implementation of SACK, there are still situations where stalls could occur if packets are lost in very specific patterns.

2.3.3 Explicit Loss Notification (ELN)

Explicit Loss Notification is an end-to-end approach which allows the TCP sender to distinguish between wireless and congestive losses. Currently, ELN is not considered a viable protocol because there is no completely reliable method to inform the sender which losses are due to congestion and which are due to errors on the wireless link. Experimental versions of ELN operate on the assumption that the receiver (or an intermediate node) is able to inform the sender of the reason for packet losses. When a packet is lost due to congestion, the receiver sends back its typical duplicate ACKs to invoke Fast-Recovery at the sender. But, when a packet must be resent due to errors, the duplicate ACK is sent with a new option field which denotes an error related loss. This causes the sender to retransmit the packet without taking congestion avoidance measures. Also, if the ELN option is set, the sender may retransmit the lost packet immediately rather than waiting for two more duplicate ACKs.

Results for an ideal ELN scheme are given in [3]. The Mobile Computing Group at Texas A&M is considering one approach which may be realistically used to implement a form of ELN. Packets lost due to congestion will never make it to the receiver. However, packets which have errors introduced on the wireless link may make it to the receiver and be discarded due to an improper checksum. With the proposed approach, the TCP receiver would examine packets which pass the IP checksum but fail the TCP checksum. Since the IP checksum was valid, the receiver knows that the source and destination addresses and port numbers are correct. Other TCP header fields would be examined to ensure they are plausible. Then a duplicate ACK with the ELN option would be returned to the sender to force a retransmission of the packet with the error.

Advantages of ELN

A TCP sender equipped with the ELN upgrade is capable of distinguishing between congestive and wireless losses. If the loss notification scheme is accurate, congestion prevention measures will only be taken when appropriate.

Disadvantages of ELN

There is no known scheme that distinguishes perfectly between congestive and wireless losses which does not require modification of intermediate nodes. The method proposed above falls short of perfect differentiation because it will consider packets which do not reach the receiver's TCP layer as being lost due to congestion.

2.4 Link-Layer Improvement Methods

Reliable wireless link-layer protocols comprise the second class of TCP improvement schemes for heterogeneous networks. Each of the link-layer improvement methods attempts to make the wireless link appear like a slower, reliable (i.e. low error rate) link to the TCP sender. Reliability can be accomplished using forward error recovery (error corrective coding), automatic repeat requests (ARQ) for lost frames, or a combination of both. Besides the method used to provide

reliability, link-layer protocols also differ in how they deliver frames to the layers above them. Some attempt an in-order delivery of frames while others allow out-of-order delivery.

Advantages of Link-Layer Protocols

Typically, link-layer protocols require little or no state maintenance at the base station (depending on whether the link-layer provides in-order delivery). This allows link-layer solutions to function efficiently during handoffs between base stations. Also, link-layer protocols do not interfere with the end-to-end semantics of TCP, so a base station or link failure will not result in lost data. Link-layer protocols are also independent of the upper layers, so the gains they achieve can be used for other transport layer protocols besides TCP. Finally, reliable link-layer scheme can be optimized for each type of wireless connection.

Disadvantages of Link-Layer Protocols

Reliable link-layer protocols perform local retransmissions between the base station and the mobile host. It has been shown in [3] and [4] that these retransmissions at the link-layer can interfere with TCP's retransmission mechanisms when error rates become significant. This interference is due to competition between the retransmission timers at the two different layers, and inadvertent invocation of Fast-Retransmit by link-layer schemes which allow out-of-order frame delivery. It could also be argued that providing reliable, sequenced delivery of frames moves the functionality of several layers down to the link-layer.

2.5 Split Connection Improvement Methods

Split connection approaches make up a third class of TCP enhancement schemes for heterogeneous networks. Split connection protocols distinguish errors on the wireless link from congestive losses by dividing the end-to-end connection into separate wireless and wired connections. An unmodified version of TCP may be used between the fixed host and the wireless host. A version of TCP customized for wireless connections, or a new transport protocol specifically designed for wireless communications is then used between the base station and mobile host. The base station is the focal point for these protocols. It must take connection requests from either the fixed host or mobile host and establish the second connection on behalf of the requester. While the two hosts are communicating, the base station is responsible for transferring data between the wireless and wired connections. It must also move this responsibility to a new base station in a manner that is transparent to both ends when a handoff occurs. The most well known of the split connection approaches is indirect TCP (I-TCP) developed at Rutgers[10] [12]. It uses TCP for both connections.

Advantages of Split Connections

Because split connection protocols completely separate wireless and congestive losses, they are able to obtain very good throughput.

Disadvantages of Split Connections

Split connection approaches require hard state maintenance at the base station. This means that handoffs will require a good deal of information exchange between the old and new base station serving the connection. Split connections also violate the end-to-end reliability of TCP because two connections are involved. Failures at the base station or along the transmission path can result in information losses because the sender receives an ACK before the data truly makes it to the ultimate destination. Finally, maintaining two separate connections creates a good deal of overhead at the base station because each piece of data must traverse the entire protocol stack twice. This could become a problem if many connections are serviced by a base station.

2.6 TCP-Aware Link-Layer Improvement Methods

The snoop protocol developed by Berkeley is an example of a TCP-Aware link-layer protocol [3]. It is a hybrid solution which uses concepts from the link-layer schemes and the split connection approaches. Unlike the other solutions discussed, the basic snoop protocol is specifically designed for data transfers from the fixed host to the mobile host. To improve the performance of data transfers in the other direction a NACK (negative acknowledgment) may be used between the mobile host and the base station. The basic snoop protocol is also unique from the others because it only requires modifications at the base station.

A snoop base station caches each packet received on the wired link and then forwards it to the mobile host. The snoop agent then watches the ACKs being returned from the mobile host to the sender. The agent maintains round-trip-time estimates for the wireless link and performs local retransmission if an ACK is not received before the timer expires. A much finer timer is used at the base station than at the TCP sender so that multiple local retransmissions are possible before the sender times out. Also, a single duplicate ACK for a packet cached at the base station results in a local retransmission. The base station is able to interpret a single duplicate ACK (rather than three) as a packet loss because the wireless link delivers packets in order.

The key to snoop is that unlike other link-layer protocols, it maintains state which it is able to use to determine whether a packet was lost due to congestion (before reaching the base station), or received at the base station and subsequently lost due to errors on the wireless link. This information is used to shield the sender from duplicate ACKs due to wireless losses, and to pass through duplicate ACKs caused by congestive losses. The state maintained by snoop also allows it to recognize TCP retransmissions so that it will not compete with them.

Advantages of the Snoop Protocol

The fact that snoop only requires modifications to the base station can be an advantage if it is desirable to change the base station without modifying any of the hosts within the Internet. More importantly, throughput improvements provided by snoop are better than any of the other experimental schemes. This is because the base station is able to accurately distinguish between congestive and wireless losses so congestion avoidance mechanisms at the sender are invoked

only when appropriate. Finally, the state maintained at the base station is soft in the sense that it can be lost without affecting the reliability of TCP.

Disadvantages of the Snoop Protocol

Although the state maintained at the base station by the snoop agent is soft, it can require a significant amount of memory, and it does complicate handoffs. Perhaps snoop's greatest disadvantage is that it requires the ACKs to follow the same path as the data in order to shield the sender from losses. This is not a problem for network topologies containing a single wireless path which every packet must traverse. It does become a problem when multiple wireless paths are possible, or with asymmetric links where the sender uses a high bandwidth, high delay path (such as a satellite link) to send the data and the receiver uses a low bandwidth terrestrial path to return the ACKs. Finally, snoop has no method of informing the sender when the base station experiences a period of high errors and this could lead to unnecessary timer expirations which invoke congestion avoidance procedures.

2.7 SCPS-TP Improvement Methods

The Space Communications Protocol Standards - Transport Protocol (SCPS-TP) is a set of extensions to TCP developed for use in satellite networks by The MITRE Corporation, Gemini Industries, NASA, and the Department of Defense [11]. Although some of the concepts introduced with this set of standards are satellite specific, there are number that can be used to improve the performance of TCP on any wireless link that experiences errors. SCPS-TP introduces a default assumption for losses which is set differently depending on the specific network environment. Unlike current versions of TCP which always use the default assumption that losses are due to congestion, SCPS-TP may use the default assumption that losses are due to errors. This makes sense in networks known to have high error rates and little or no congestion. The default assumption can also be temporarily changed during periods of high error rates so that congestion avoidance measures are not unnecessarily invoked.

SCPS-TP uses a selective negative acknowledgment (SNACK) option. The goal of this option is to provide the sender with a more complete picture of which packets have been lost. The SNACK option serves the same purpose as the SACK option discussed previously, but its implementation is completely different. Since the bandwidth available to ACKs on a satellite is often very limited, the SNACK option conveys lost packets in a way that uses fewer bytes and requires fewer ACKs to generate a retransmission.

SCPS-TP also uses a different method of congestion avoidance which is often referred to as TCP Vegas. The other standard TCP implementations discussed above find an optimal operating point by linearly increasing their bandwidth until a loss occurs. Since any losses severely degrade throughput on high delay-bandwidth links, TCP Vegas attempts to avoid this pattern of ramping up throughput until losses occur. It does this "by increasing its congestion window more slowly than standard TCP and by measuring the achieved throughput gain after each increase to detect the available capacity without incurring loss." [11]

Advantages of SCPS-TP

SCPS-TP achieves a very significant performance improvement over standard TCP when errors are present in a high delay-bandwidth wireless environment. SCPS-TP combines the benefits of a number of different techniques. First, it is able to distinguish between congestion and wireless losses. Then, it combines this ability with the SNACK option so that it efficiently retransmits the lost packets. SCPS-TP is also an ideal candidate for the wireless protocol used in a split connection.

Disadvantages of SCPS-TP

The changes made as part of SCPS-TP are quite significant and so most fixed hosts will probably not be equipped to take advantage of these upgrades. The authors indicate in [11] that the TCP Vegas congestion control mechanisms are very sensitive to round-trip-times and require a significant amount of tuning.

3.0 The Partial Acknowledgment Protocol

The partial acknowledgment (ACKP) protocol is a new protocol that was proposed in [6] by the Mobile Computing Group at Texas A&M and implemented and tested as part of this project. Previous research efforts at Texas A&M focused on the proposal and simulation of an Explicit Bad State Notification (EBSN) protocol [5]. The EBSN protocol was designed to send a notification from the base station to the receiver when the base station was experiencing high error rates (deep fades). This notification then caused the receiver to revise its retransmission timer to give the base station more time to make the delivery and lower the likelihood of an unnecessary time-out/Slow-Start at the sender. Results from the EBSN study indicated that it effectively prevented unnecessary time-outs and congestion avoidance measures during high wireless error periods which resulted in performance improvements over plain TCP of 50 percent on wireless LANs and 100 percent on wide area wireless networks.

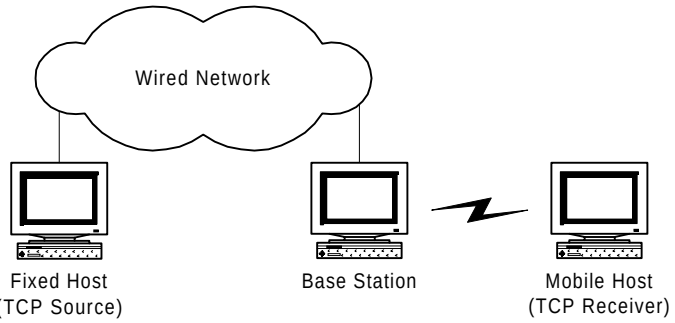
The ACKP protocol has been proposed as a means of combining the effectiveness of the snoop protocol at performing TCP-Aware link-layer retransmissions with the capabilities of an EBSN-like protocol which prevents unnecessary time-outs during periods of high error rates on the wireless link. This combination will maintain the end-to-end semantics of TCP and differentiate between congestion and wireless losses over a wide spectrum of error conditions.

The central concept behind the ACKP protocol is to further distinguish between congestive and wireless losses by splitting the acknowledgment mechanism into two parts rather than splitting the connection in two as with I-TCP and other split connection protocols. The two types of ACKs used by the protocol are: partial acknowledgment (Ack_p) and complete acknowledgment (Ack_c). A partial acknowledgment with the sequence number N informs the sender that packets up to N-1 have been received by the base station. An Ack_c is the standard TCP ACK sent by the receiver which informs the sender that packets up to N-1 have been received. This new protocol requires upgrades to the base station, and to the sender, but no modifications are required at the receiver.

3.1 Expected Network Topology

The ACKP protocol operates between a fixed host and a base station. The basic network topology that ACKP is designed to work with is shown below. The protocol is flexible and could be expanded to work with a wireless link in the middle of the connection, wireless links at both ends of the connection, or a number of other network topologies.

Figure 4 - General Heterogeneous Network Topology



3.2 ACKP During Connection Establishment

The ACKP protocol is designed to be compatible with older versions of TCP, but only TCP versions containing the ACKP upgrades can take advantage of the performance improvements offered by the protocol. During the connection establishment phase, the sender indicates whether it is equipped to handle the ACKP protocol. Newer versions of TCP containing the ACKP upgrade will set a TCP header option field to indicate they can take advantage of partial acknowledgments. Older TCP versions which are unaware of ACKP will omit this option field. If the base station receives the connection request with the option included, it knows that the sender is equipped to handle partial acknowledgments. If the option field is omitted, the base station will not send any partial acknowledgments.

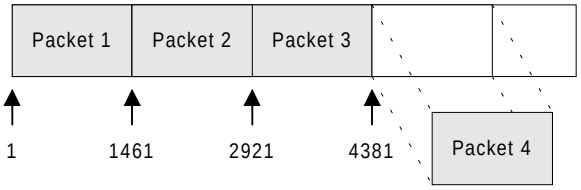
3.3 Base Station Operation

The ACKP protocol is designed to work at a base station that is already equipped with the snoop protocol since it relies on state information maintained by the snoop agent. When the ACKP protocol is executed at the base station, the snoop buffer is scanned to determine the packet with the highest contiguous sequence number. A new variable called *lastackp* is then compared to the sequence number of the packet. If it is less than the sequence number of the packet, an Ack_p would provide the sender with new information. If *lastackp* is equal to the sequence number, there is no need to send an Ack_p because the sender already has the latest information from the base station. The partial acknowledgment protocol is not run immediately after receiving a packet. Instead, a delayed ACK timer is used so that under normal circumstances (i.e. in the

absence of errors) an Ack_c will be received before the timer expires. This will drastically reduce the number of Ack_p packets sent which in turn allows more efficient use of the wired network.

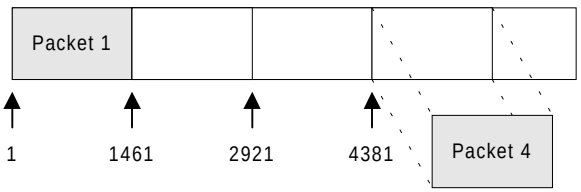
When data is received at the base station, one of several scenarios can occur. In the first scenario, an in-order packet arrives and snoop caches the packet and starts a local retransmission timer. This retransmission timer also functions as the delayed ACK timer for the ACKP protocol. If the timer expires before an Ack_c is obtained from the receiver, snoop will retransmit the packet. The partial acknowledgment protocol will then scan through the snoop buffer and determine that *lastackp* is less than this packet's sequence number, so an Ack_p will be returned to the sender informing it that the packet was received by the base station, but errors on the wireless have prevented normal delivery. The *lastackp* variable will also be updated. In figure 5 below, the Ack_p sent by the base station would be for packet number four.

Figure 5 - ACKP Scenario 1



The second scenario involves an out-of-order packet arriving at the base station. The packet may be out-of-order for two reasons: it took a faster path between the sender and the base station, or some packets ahead of it were lost due to congestion. The snoop agent will again buffer the packet and start a local retransmission timer. If the timer expires before an Ack_c is obtained from the receiver, snoop will retransmit the packet. The partial acknowledgment protocol will then scan through the snoop buffer. If the other packets have arrived at the base station, this packet will have the highest contiguous sequence number and the same steps will be followed as in the first scenario. If the packets have been lost due to congestion, the packet with the highest contiguous sequence number will be equal to *lastackp* and lower than this packet's sequence number, so no Ack_p will be sent. This will cause the sender to take congestion avoidance measures just as it normally would. In the figure below, no Ack_p will be sent in response to the arrival of packet four because packets two and three have been lost due to congestion.

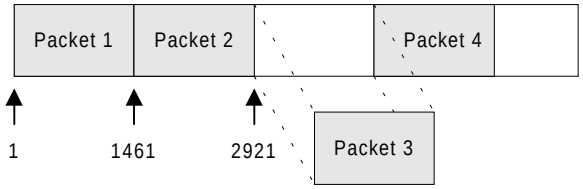
Figure 6 - ACKP Scenario 2



The third scenario involves the receipt of a packet which fills a hole in the packet buffer at the base station. This situation will normally occur when a packet has been lost due to congestion and then retransmitted. The same initial steps are followed as above. If the retransmission timer

expires, the snoop buffer is scanned, and a packet with a sequence number above this one will be the packet with the highest contiguous sequence number. An Ack_p will be sent for this packet with the highest sequence number and *lastackp* will be updated. In figure 7 below, an Ack_p will be sent for packet four if the partial acknowledgment protocol is invoked in response to the arrival of packet three.

Figure 7 - ACKP Scenario 3



3.4 Operation at The Sender

A TCP sender equipped with the ACKP upgrade is able to distinguish an Ack_p from an Ack_c by whether or not the packet has the partial acknowledgment option set. ACKs with the option set indicate to the sender that all packets up to but not including the sequence number acknowledged have been received at the base station, and the base station is having trouble delivering the packets over the wireless link. This partial acknowledgment causes the sender to reset its retransmission timer so that an unnecessary time-out is avoided while the base station attempts to deliver the packet. Other possible variations on this would involve using a constant or exponential backoff rather than merely resetting the timer when an Ack_p is received.

If the option field is not set, the ACK is a complete acknowledgment sent by the receiver. The sender responds by performing the complete set of TCP ACK processing which includes updating round-trip-timers, adjusting windows, freeing memory held by the acknowledged packets, and sending new data. None of these actions are performed when a partial acknowledgment is received. One variation on the handling of complete acknowledgments would be to ignore round-trip timing measurements for packets which also had an Ack_p associated with them. This is analogous to Karn's algorithm which says that the RTT measurement for a packet which has been retransmitted should be ignored.

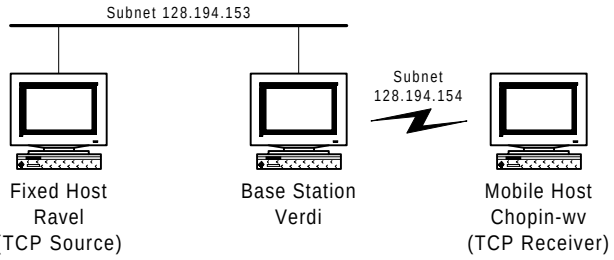
4.0 Implementation Details

The experimental testbed used for this project consisted of three Pentium based desktop computers with each machine running the FreeBSD 2.1 operating system. The first computer, Ravel, was connected only to a 10 Mbps ethernet LAN and was used as a fixed (wired) host. The second PC, Verdi, was connected to both the ethernet LAN and a 2 Mbps WaveLAN wireless LAN operating in the 915 MHz band. The wireless LAN used a Direct Sequence Spread Spectrum (DSSS) modulation technique with a CSMA/CA media access protocol and an

addressing scheme identical to ethernet. Verdi operated as the base station which relayed packets between the wired and wireless links. The third computer, Chopin, was connected only to the WaveLAN wireless LAN and was used as a mobile (wireless) host. The name used to refer to a machine on its wireless interface always has the suffix "-wv" added. Thus Chopin is referred to as Chopin-wv and the wireless connection for Verdi is referred to as Verdi-wv.

Unless otherwise noted, all of the experiments in this project have focused on 2 million byte bulk data transfers from the fixed host to the mobile host. Data flow in this direction is typical since the majority of the time mobile hosts act as clients accessing data bases, files and WEB pages residing on fixed hosts. The initial test set-up is illustrated in figure 8 below.

Figure 8 - Initial Heterogeneous Testbed



One of the key goals of this project was to perform experimentation using a real testbed rather than simulating the effects of various TCP enhancements on bulk data transfers. In past efforts it has been observed that simulation results are often more likely to be questioned due to uncertainties regarding the accuracy of the simulation environment and the assumptions made. With a real implementation, there is much less room for dispute since all of the processing delays, propagation delays, and protocol idiosyncrasies are taken into account.

The Mobile Computing Group at Texas A&M looked at several possible operating systems before deciding to use FreeBSD. The main requirement was to obtain an operating system which was freely distributed and came with the source code so that experimental modifications could be made. Early on, LINUX was chosen because of its wide user base and availability of drivers. Soon afterwards, the group discovered that the experimental work being performed at Berkeley and most other places was being done on various BSD derivatives (i.e. FreeBSD, BSDI, NetBSD). Therefore, a switch was made to FreeBSD so that we could take better advantage of these existing efforts and so that our results would be more directly comparable.

4.1 Addition of Snoop Functionality to FreeBSD

Once the testbed shown above was operating correctly with the Reno version of TCP, which comes standard with FreeBSD 2.1, the next step taken was to add the snoop protocol. As discussed in the introduction, this gave us an exercise which allowed us to become familiar with the inner workings of the networking code while also providing us with a good baseline for

further development and comparison. Since the ACKP protocol was designed to operate on top of the snoop agent, it was essential to first have a working version of the snoop protocol.

The source code for the version of snoop designed to run on BSDI was obtained from Berkeley. The snoop source code modifications were spread across five main files. Three of the files contained existing TCP procedures which had been modified to add the snoop functionality, and two files were completely new. The three existing files were the hardest to examine because there were many differences between the versions obtained from Berkeley and those which come standard with FreeBSD. Some of the variations were due to differences between the FreeBSD and BSDI operating systems. Others were related to the snoop protocol, and a third group of variations were related to other networking upgrades Berkeley was testing. After sorting out exactly which lines of code needed to be added to FreeBSD, problems such as memory buffer (mbuf) allocation for the snoop cache, and other kernel configuration issues were solved. In all, the required modifications were modest. It was the process of determining exactly what each piece of code did and whether to add it that was difficult. In many cases a whole trail of source code files had to be examined to determine the meaning of a line of code. Appendix C contains documentation which shows each modification in detail.

4.2 Partial Acknowledgment Implementation

Once the FreeBSD version of the snoop protocol was working, an initial version of the partial acknowledgment protocol was implemented. The changes made to the FreeBSD kernel for the ACKP protocol are described in detail in Appendix E. The upgrade involved code additions in three key areas. The first and hardest portion of the upgrade was the implementation of a module at the base station which could create an appropriate Ack_p packet. This procedure is unusual in that it is constructing a TCP packet to pass from the base station to the sender, but the base station is not really part of the connection so it must create a packet that looks like it is coming from the receiver. Luckily, the information required to mimic the receiver is contained in the header of the incoming data packets.

The second portion of the upgrade involved determining when to send an Ack_p. Although the procedure outlined in section three was written, the initial implementation installed at the base station was much simpler. The goal of the initial implementation was to determine if the ACKP protocol would provide performance enhancements such as better throughput and more efficient use of the wired and wireless links during periods of high wireless error rates. The best way to implement the protocol in order to obtain these desired results was to send an Ack_p for every packet received. This initial approach is valid for three reasons. First, since we know that our sender is equipped to handle Ack_p packets, the sender and base station do not have to negotiate this option during the connection establishment phase. The option negotiation feature can be added later for backwards compatibility. Second, within the controlled environment of the testbed no congestive losses are present, so sending an Ack_p for every packet will not adversely affect the connection's response to congestion. Third, the number of Ack_p packets sent between the base station and the fixed host should not affect the parameters we want to measure. These packets may create additional collisions on the wired link, but since the wired link is several times

faster than the wireless link, the additional collisions on it should not affect throughput or any of our other performance metrics. The full implementation can be added once it is determined whether or not the ACKP protocol will offer significant performance improvements.

The third portion of the upgrade involved adding coding at the sender so that it could recognize and react to an Ack_p packet. The TCP option handling code now recognizes a new ACKP option. When this option is present in the packet, the option handling code sets a flag. The portion of the TCP code which handles incoming packets sees that this flag is set, updates the retransmission timer so that a time-out is less likely, and then drops the packet without processing it any further. Since an Ack_p is being sent for every packet in this initial implementation, the retransmission timer can not be incremented exponentially. Instead, it is reset to a constant value which is roughly twice the initial time-out value each time an Ack_p packet is received. This ensures that the new protocol gives the base station roughly twice as long to recover during periods of burst errors.

4.3 Bit Error Model Implementation

The bit error model used for the experiments was obtained from UC Berkeley. It is designed to use either a Poisson-distributed bit-error model or a Markov model which transitions between high and low error rates. The Poisson model is capable of damaging a single byte or creating a burst of errors which damage several packets in a row. One copy of the model runs at the mobile host to damage data packets, and another runs at the base station to damage ACKs. Parameters such as the mean error rate and the burst size determine how the model behaves.

When the error model determines that an error should be injected, it modifies either the IP header checksum or the TCP checksum depending on where the error occurs within the packet. This forces either the TCP or IP layer to drop the packet. Then it calculates the interval in bytes between the spot of the current error and the place the next error is to occur. To do this, a table of 50,000 exponentially distributed integers is maintained. A number is randomly chosen from the table and scaled by the mean error rate. This scaled value is then used as the interval between errors. The table size combined with the random arrival of packets ensures that a repeating pattern of errors is highly unlikely.

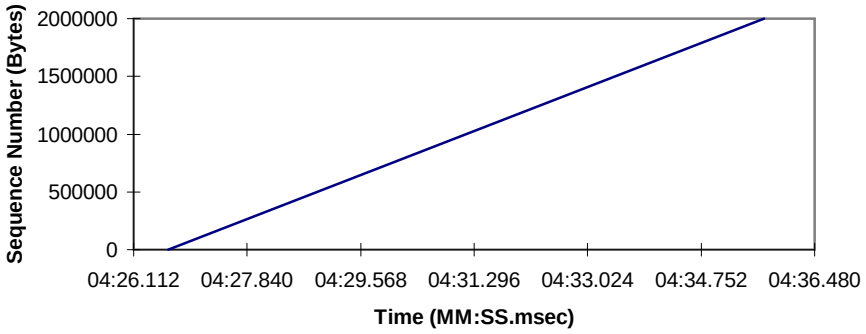
While this type of model is not necessarily an exact representation of the pattern of errors experienced on a noisy wireless link, it is effective because the Poisson distribution has a variance equal to its mean. The result is that errors are not evenly distributed. Sometimes they occur in rapid succession and other times they are spaced quite far apart. As we will see in the results section, closely spaced errors cause extreme performance degradation in existing versions of TCP. The Poisson-distributed error model is therefore effective because it exercises the area where TCP is most vulnerable.

4.4 Measurement Methods

Two methods were used to obtain performance metrics. The first method was rather straightforward and involved using a packet filter called BPF (Berkeley Packet Filter) which comes standard with the FreeBSD distribution. The packet filter is a pseudo device within the kernel that functions as a promiscuous receiver at the link-layer (listens to all frames transferred on the link). An application program called tcpdump which is also included in the standard FreeBSD distribution is used to filter the packets captured by BPF. It allows the user to provide filter specifications such as the source, destination, and protocol of packets which should be accepted. These packets can be stored to a file and then later analyzed to look for duplicate ACKs, retransmitted packets, long periods of inactivity, and other performance information. A portion of a tcpdump file has been included for reference in Appendix A.

One metric which often provides insight is a plot of the sequence numbers versus time. The plot for an ideal connection which has every packet delivered in-order with no retransmissions is shown in Figure 9 below. The horizontal axis is the time in terms of hours, seconds, and milliseconds. The vertical axis is the sequence number in bytes. The ideal plot is a straight line and the slope of the line is the throughput for the connection. Diagrams such as these were made by taking the raw tcpdump data and then parsing and plotting it in Microsoft Excel.

Figure 9 - Ideal Plot of Sequence Number vs. Time



The second method used to measure performance was more involved. While tcpdump provides data related to packet headers, it is capturing this data below the transport layer, so it is incapable of providing information about the state variables related to the TCP connection. To completely understand the performance of a connection, information such as retransmission time-outs, Fast-Recovery initiation, and other parameters contained in the TCP control block at the sender must be measured.

An initial attempt was made to write code from scratch which would measure the desired parameters. This code was able to successfully capture the information, but the methods available

to move the information from within the kernel to a file for logging were messy at best. Instead, modifications were made to existing debugging mechanisms. TCP contains a trace function which records information into a debugging buffer provided that debugging has been enabled for the given connection. Each buffer entry contains the packet header and a copy of the values contained within the TCP control block at the time the entry was logged. An application program called trpt which has symbolic links to the buffer can then be used to print the contents. Changes were made in the kernel to increase the size of the debugging buffer, to modify where the trace code was called, and to add a new type of debugging entry for Ack_p packets. The kernel configuration files also had to be changed to include the trace code and its related files. Finally, the trpt program was customized to print the desired information in a format that was useful.

The advantage of using TCP's trace coding in combination with the trpt program is that the data capture and data display functionality are completely separated. Because the trace function captures both the TCP header and the TCP control block, virtually any parameter of interest is available. The trpt program can then be customized to print only the desired parameters. This means that after the initial kernel upgrades are made, the only changes being made are to trpt which is an application program. Therefore, the kernel does not have to be rebuilt each time the user wants to display different pieces of information. This provides a very flexible means of displaying the data captured.

4.5 Test Sender Program

The test sender program is a simple application program which uses the socket API to open a connection with a remote machine and make bulk data transfers. It has a text based menu which allows the user to select the port number and name of the destination machine. When the program is first started, the default destination machine is Chopin-wv and the default port number is 9 which is the well know port number for the discard server on a UNIX machine. The discard server appears like a regular receiver to the TCP sender, but it throws away each packet that it receives. A destination other than the default may be chosen by the user. Once this destination is chosen, it becomes the new default. The test sender program also allows the user to enable or disable debugging on the socket which is used for the data transfer. Enabling debugging causes the kernel to record incoming packets and the state of the TCP control block in the trace buffer. These packets can be examined using trpt at a later time as discussed in the previous section. The test sender program also times the transfer to the nearest millisecond. The source code for the test sender program is included in Appendix B.

4.6 Wireless Configuration Program

The wireless configuration program is another application program which uses the socket API to check and set the error model parameters and TCP enhancement features. This program is very useful because it allows the test environment to be changed without requiring the kernel to be modified and rebooted which saves time and ensures that conditions remain similar during testing. The *setsockopt* and *getsockopt* socket system calls allow an application program to set features

within the kernel. Normally this involves features such as the maximum segment size, and buffer sizes which relate to a particular socket. The FreeBSD kernel running on Verdi and Chopin has been upgraded to include additional global socket options related to the error model and TCP enhancements. These options determine whether snoop, ACKP, and the error model are enabled or disabled. They also allow error model parameters such as the mean error rate and the burst size to be changed. The configuration program contains the *setsockopt* calls required to set these new kernel features, and the *getsockopt* calls required to check the current kernel settings. The source code for the wireless configuration program is included in Appendix B.

5.0 Experimental Results

In this section, the experimental results are presented and explained. The first sub-section gives some of the general results which were obtained early on in the project. These are shown to demonstrate how unmodified TCP Reno performs. They illustrate and confirm many of the problems associated with using TCP in heterogeneous networks which were presented earlier in this report. The rest of the results look at how the snoop protocol and the ACKP protocol affect the performance of TCP. Section 5.2 compares results in a wireless LAN environment, and section 5.3 compares results in a low bandwidth wireless environment. The experiments performed in sections 5.1 and 5.2 were designed to test the performance of the ACKP protocol in two situations: during periods of single-packet wireless losses and during periods where packets are lost in bursts. The intent of the experiments was to show that ACKP would not degrade the performance of snoop which already performs well for single-packet losses and that ACKP would improve the performance of snoop during bursts of packet losses.

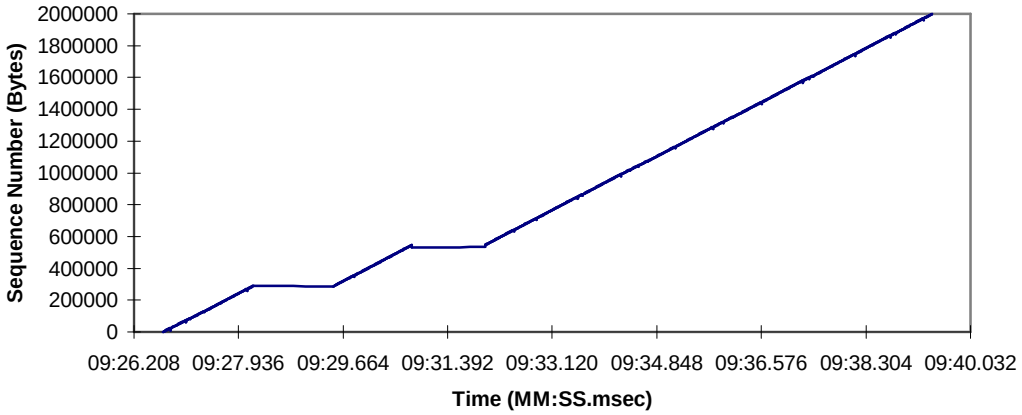
5.1 General Results

This section discusses various plots which show how plain TCP (Reno) reacts to losses due solely to errors, losses due solely to congestion, and losses due to a combination of both. The first plot shown in Figure 10 on the next page was made from tcpdump information captured during a 2 million byte data transfer from Ravel to Chopin using the testbed shown previously in Figure 8. The typical round-trip-time for a packet in this testbed was between 6 and 8 milliseconds, and the ideal throughput for a connection was just over 200 KB/s. During the transfer plotted below, the error model was used to inject errors on the wireless link at an average rate of once per 64 KB (i.e. a bit error rate of 1.9×10^{-6}). These errors resulted in a transfer time of 12.71 seconds with an average throughput of 153.67 KB/s. This throughput value, which is only 25 percent below the ideal rate, is actually unusually good for TCP Reno as we will see in the next section.

Several interesting concepts can be observed in the plot in Figure 10. First, the flat portions of the plot indicate that the transfer has stalled. It is also evident from looking at a number of plots such as this that these stalls account for almost the entire difference in performance between a connection with errors, and a connection that is error-free. A closer review of the data collected during the transfer reveals the reason for the stalls. When packets are lost in quick succession, TCP Reno shrinks its congestion window in half for each loss. This results in a congestion

window which is too small to obtain the three duplicate ACKs required to initiate fast recovery. Therefore, the connection waits in this state of temporary deadlock until the sender's retransmission timer expires causing the lost packet(s) to be resent. Snoop and many of the other TCP enhancements prevent the sender from shrinking its congestion window in response to errors so they are able to maintain a congestion window which is large enough to avoid stalls such as these.

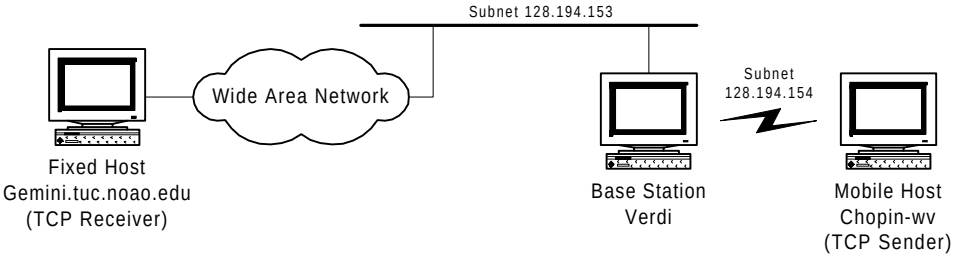
Figure 10 - TCP Reno Transfer on a LAN with Wireless Errors



The second concept which this plot illustrates is the effectiveness of Fast-Recovery. Each of the 30 small "blips" in the plot represents a packet which was lost and retransmitted without a stall occurring. When the congestion window is large enough to detect the loss and initiate Fast-Recovery, lost packets have very little effect on throughput since the main penalty is the time required to detect and retransmit the packet. Without Fast-Recovery, each of the "blips" would have created a stall.

The next two plots involved sending data over a WAN using the topology shown in Figure 11 below. For both plots, a connection was established between Chopin (the mobile sender) and the discard server on the machine gemini.tuc.noao.edu in Tucson, Arizona. A 2 million byte data transfer was then performed from Chopin to Gemini and recorded using tcpdump.

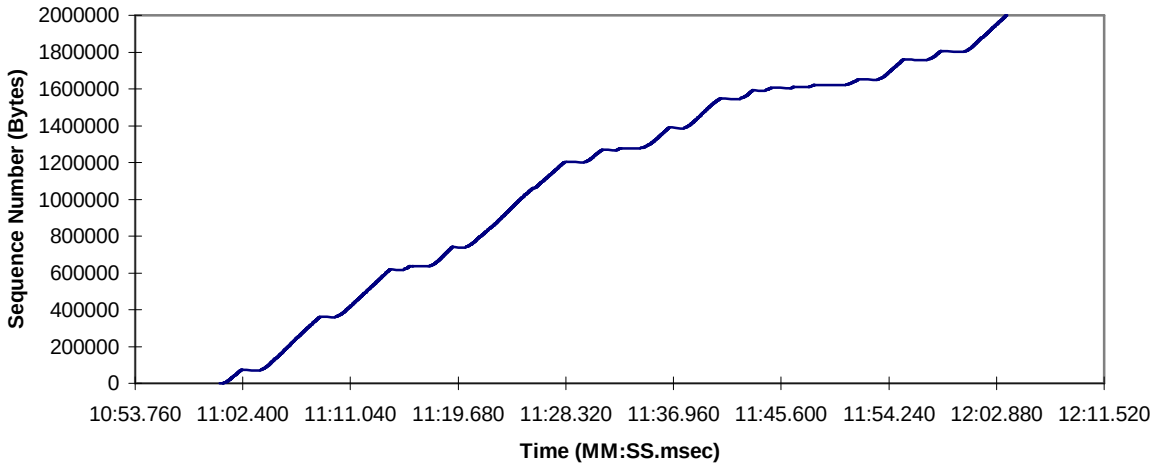
Figure 11 - Wide Area Testbed



Even with an ideal connection which experienced no losses, one would expect lower throughput on a WAN because the available bandwidth is lower. Also, the packet size for a WAN transfer is

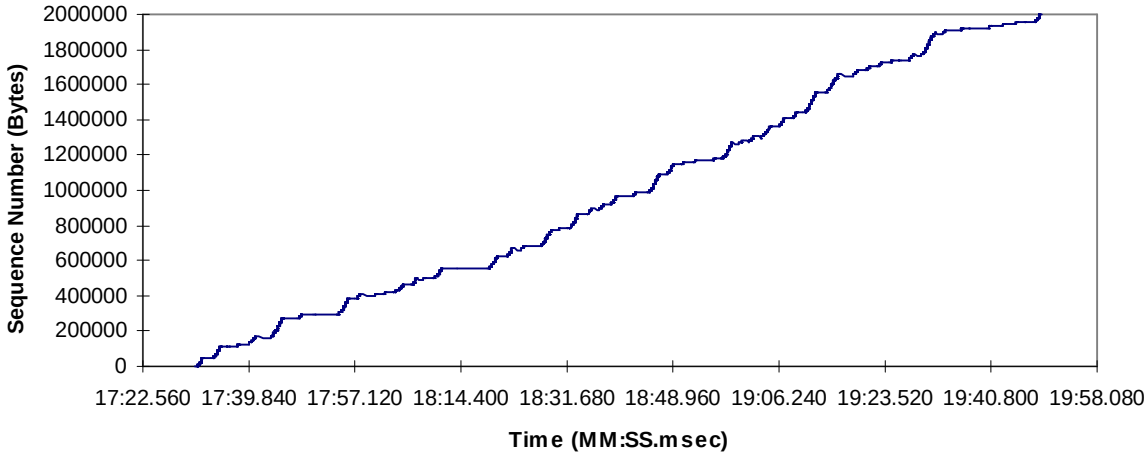
512 bytes and the packet size for a LAN transfer is 1460 bytes. This smaller packet size adds overhead and lowers the ideal throughput. The typical round-trip-time for a packet was 67 milliseconds, and the ideal throughput for a connection operating within this WAN environment was almost 61 KB/s. The plot shown in Figure 12 below contained no errors on the wireless link, but 17 packets were dropped due to congestion within the WAN. These losses (coupled with the competition for bandwidth among the connections at the congested gateway) resulted in a transfer time of almost 63 seconds with an average throughput of 31.0 KB/s.

Figure 12 - TCP Reno Transfer on a WAN with No Wireless Errors



For the plot shown in Figure 13 the same test set-up was used, but errors were injected on the wireless link at a mean rate of once per 64 KB. The number of packets lost due to congestion plus errors on the wireless link was 54 or roughly 1.38 percent of the total sent. These losses resulted in a transfer time of almost 137 seconds with an average throughput of 14.3 KB/s.

Figure 13 - TCP Reno Transfer on a WAN with Wireless Errors



The results from these transfers across a WAN demonstrate several things. First, a TCP connection across a WAN experiences performance degradation mainly due to stalls caused by two or more packets being lost in close succession. This is no different from a LAN, except that the stalls can be more pronounced on a WAN because the RTT variance is greater and the retransmission timers are consequently set much higher. Second, while a LAN can achieve optimal throughput even with a congestion window as small as two packets (just over 2 KB), a WAN such as the one tested often needs a congestion window size of at least 8 KB to perform optimally. Third, even though transfers without any wireless losses were far from optimal, the injection of errors reduced throughput an additional 50 percent by unnecessarily invoking congestion avoidance measures.

5.2 Wireless LAN Performance

The main purpose of this section is to show how the ACKP protocol affects the performance of TCP over a connection composed of a wired LAN and a wireless LAN. All of the experiments conducted in this section use the testbed shown in Figure 8. Subnet 128.194.153 is the 10 Mbps ethernet connection, and subnet 128.194.154 is the 2 Mbps WaveLAN wireless LAN. This test environment ensures that no congestion occurs, and all losses may therefore be attributed to errors on the wireless link.

Results were obtained for several different scenarios. Each test scenario involved sending 2 million bytes of data from the fixed host to the mobile host and a total of twenty test runs were performed for each scenario. The values given in the tables below reflect the average of the runs. Four parameters were measure for each test case: transfer time, wired goodput, wireless goodput, and timer expirations. The transfer time was used to compute the average throughput for each run. Goodput is defined as the ratio of the actual transfer size verses the total number of bytes transmitted over the path[3]. A transfer in which there are no losses has a goodput of 100 percent and a transfer with losses has a goodput below 100 percent. The goodput values given in the tables below were determined by dividing the ideal number of data packets required for the transfer by the actual number of data packets recorded for the transfer. The timer expirations are measured because the purpose of the ACKP protocol is to eliminate or at least decrease the timer expirations in an effort to improve performance.

5.2.1 Bit Errors

The results for the bit error tests are shown in Table 1 on the next page. The first scenario, called “Ideal Transfer”, used plain TCP Reno with no errors injected on the wireless link. This test case was used to determine the ideal performance of the LAN testbed. The results for this case could then be used as a basis for comparison among the other scenarios. As expected, no packets were lost, so the number of timer expirations was zero and the goodput both on the wireless link and the wired links was 100 percent. The throughput of 199.92 KB/s (1.6 Mbps) falls below the nominal wireless bandwidth of 2 Mbps because of collisions and overhead, but it is in line with the WaveLAN performance cited in other studies and the manufacture’s specifications.

Table 1 - LAN Performance with Bit-Errors

	Ideal Transfer	TCP Reno	Snoop	Snoop + ACKP
Avg. Transfer Time	9.770 sec.	20.878 sec.	10.513 sec.	10.576 sec.
Trans. Time Std. Dev.	0.034 sec.	4.695 sec.	0.280 sec.	0.520 sec.
Avg. Throughput	199.92 KB/s	98.36 KB/s	185.91 KB/s	185.07 KB/s
Throughput Std. Dev.	0.682 KB/s	23.117 KB/s	4.890 KB/s	8.372 KB/s
Pct. Ideal Throughput	------	49.20 %	93.00 %	92.58 %
Avg. Wired Goodput	100 %	97.4 %	99.4 %	99.5 %
Avg. Wireless Goodput	100 %	97.4 %	97.3 %	97.2 %
Avg. Timer Expirations	0.00	7.50	0.00	0.00

The remaining test cases in this sub-section involved injecting single bit-errors at a mean rate of once every 64 KB. The second test case used standard TCP Reno. As expected, the wireless errors caused TCP Reno to invoke congestion avoidance measures which resulted in a drastic reduction in throughput. Because each lost packet must be retransmitted at the sender, the goodput for the wired and wireless links is identical. Also, a number of stalls occurred during each transfer as indicated by the timer expirations in Table 1.

The third test case used the same bit error rate, but the snoop agent at the base station was turned on. The performance of snoop with errors is below the ideal transfer for several reasons. First, anytime errors occur, performance must suffer to a certain degree since some of the packets must be sent more than once between the base station and the mobile host. This is reflected in the wireless goodput which is 97.3 percent. It also takes a finite amount of time to detect these errors. The receiver's window may fill while waiting for the lost packet and cause the transfer to stall for a very short time (on the order of ten milliseconds) while the lost packet(s) are resent and the window is shifted right. This last problem can be cured or reduced by increasing the size of the receiver's window.

The third test case demonstrates that in a LAN environment the snoop protocol is optimal for single bit-errors in the sense that the sender is completely shielded from wireless losses. Shielding the sender from losses prevented any congestive measures from being taken (no timer expirations or duplicate ACKs were observed) which resulted in excellent performance in terms of throughput and goodput on the wired link. These results were not surprising and confirmed the claims made by the developers of the snoop protocol.

The final test case involved using both the snoop protocol and the new ACKP protocol. Since the previous test case showed that no timer expirations occurred with the snoop protocol, the only result which was important to show was that the additional processing performed by the ACKP protocol did not degrade performance in situations where snoop was already behaving optimally. As the results in Table 1 show, the performance with ACKP is almost identical to the performance obtained using only snoop. The slight reduction in throughput can most likely be attributed to variation in the error patterns which resulted in slightly higher transfer times for two of the ACKP test runs.

Although it is not shown in the table above, snoop and ACKP were tested at mean error rates of once every 32 KB and once every 16 KB as well. These results also showed that the snoop protocol was optimal at shielding the sender from wireless losses and that the ACKP protocol did not degrade performance even when the error rates became more severe.

5.2.2 Burst Errors

The test cases in this section look at how TCP Reno, the snoop protocol, and the ACKP protocol perform when bursts of errors damage six consecutive packets. The mean distance between the onset of two error bursts is 64 KB which is the same as for the single bit-errors. The ideal case is the same as in Table 1 and is therefore not shown in Table 2 below. The first test case which uses unmodified TCP Reno is included just for comparison.

The second and third test cases show the performance of snoop by itself, and of snoop with the ACKP protocol. These test cases do show that the ACKP protocol was able to decrease the number of sender time-outs and retransmissions during periods of high error rates. However, the number of time-outs even without ACKP was still relatively small. As one might expect, the throughput and wireless goodput dropped significantly due to the greater number of total errors. Unfortunately, the performance with the ACKP protocol was not noticeably better than the performance obtained using just the snoop protocol. This can be attributed to several factors. First, since there were few time-outs when snoop alone was used, there is little room for improvement once the ACKP protocol is added. These rare time-outs do result in congestion avoidance measures at the sender, but they do not occur frequently enough to shrink the window to the point that a stall can occur. Second, both the wired and wireless LANs are high bandwidth, low delay links. In this type of environment there is no significant penalty for re-transmitting data which has already successfully reached the base station or the receiver. Also, the reduction of the congestion window due to an occasional time-out does not affect the throughput rate of the connection.

Table 2 - LAN Performance with Burst Errors

	TCP Reno	Snoop	Snoop + ACKP
Avg. Transfer Time	300.455 sec.	19.031 sec.	19.054 sec.
Trans. Time Std. Dev.	90.581 sec.	3.378 sec.	3.411 sec.
Avg. Throughput	7.19 KB/s	105.37 KB/s	105.66 KB/s
Throughput Std. Dev.	2.584 KB/s	16.548 KB/s	18.978 KB/s
Pct. Ideal Throughput	3.60 %	52.71 %	52.85 %
Avg. Wired Goodput	85.9 %	99.6 %	99.7 %
Avg. Wireless Goodput	85.9 %	87.5 %	88.2 %
Avg. Timer Expirations	56.60	0.65	0.30

5.3 Performance for a Low Bandwidth Wireless Network

After observing the failure of ACKP to improve the performance of snoop in a test environment composed of wired and wireless LANs, the question that remained was if the ACKP protocol would offer significant performance improvements in other environments. The round-trip-time for a packet in a wireless LAN remains relatively small regardless of whether the packet reaches the receiver immediately or must be retransmitted from the base station several times due to errors on the wireless link. On a low bandwidth wireless network, retransmissions between the base station and receiver can take a significant amount of time. Therefore, if a packet or a burst of packets is lost in this type of environment, the sender's retransmission timer is more likely to expire unnecessarily. Wide area wireless networks (i.e. those that cover an entire city rather than the floor of a building like the WaveLAN) and wireless LANs under heavy load are examples of low bandwidth wireless environments.

The tests conducted in this section use the same testbed as the previous section except that a delay has been added at the base station in each portion of the code where a packet is forwarded on the wireless link (see Appendix E). These transmission delays make the wireless link appear to have a peak bandwidth of 14.6 KB/s with a propagation delay that is still below 10 milliseconds. The ideal case shown in Table 3 below demonstrates that the testbed behaves as expected when no errors are present on the wireless link. Also, as expected, the introduction of bit errors at the mean rate of once per 64 KB causes the throughput of TCP Reno to be cut in half with goodput percentages that are about the same as for the wireless LAN test case.

The snoop and the snoop plus ACKP test cases also have results which are similar to those in the last section. As Table 3 shows, the sender experiences some time-outs with snoop even when only a single packet loss occurs on the low bandwidth wireless link. Again however, the number of time-outs is very low and performance is therefore not significantly affected by them. Still, the fact that single packet losses cause sender time-outs for the snoop test cases is promising since it means an even higher number of time-outs should occur for burst errors.

Table 3 - Low Bandwidth Wireless Network with Bit Errors

	Ideal Transfer	TCP Reno	Snoop	Snoop + ACKP
Avg. Transfer Time	138.100 sec.	259.830 sec.	143.515 sec.	143.558 sec.
Trans. Time Std. Dev.	0.925 sec.	8.146 sec.	1.739 sec.	1.363 sec.
Avg. Throughput	14.14 KB/s	7.52 KB/s	13.61 KB/s	13.61 KB/s
Throughput Std. Dev.	0.093 KB/s	0.230 KB/s	0.163 KB/s	0.127 KB/s
Pct. Ideal Throughput	------	53.20 %	96.24 %	96.20 %
Avg. Wired Goodput	100 %	97.6 %	99.8 %	99.8 %
Avg. Wireless Goodput	100 %	97.6 %	97.5 %	97.6 %
Avg. Timer Expirations	0.00	5.40	0.35	0.05

The last set of test cases looks at how TCP Reno, the snoop protocol, and the ACKP protocol perform when bursts of errors damage six consecutive packets on a low bandwidth wireless network. The mean distance between the onset of two error bursts is 64 KB just like in the

wireless LAN test cases. The ideal case is the same as in Table 3 and is therefore not shown in Table 4 below. The first test case which uses unmodified TCP Reno is included just for comparison.

Table 4 - Low Bandwidth Wireless Network with Burst Errors

	TCP Reno	Snoop	Snoop + ACKP
Avg. Transfer Time	720.520 sec.	181.970 sec.	182.451 sec.
Trans. Time Std. Dev.	125.229 sec.	15.235 sec.	17.914 sec.
Avg. Throughput	2.79 KB/s	10.80 KB/s	10.79 KB/s
Throughput Std. Dev.	0.500 KB/s	0.828 KB/s	0.952 KB/s
Pct. Ideal Throughput	19.73 %	76.35 %	76.31 %
Avg. Wired Goodput	84.9 %	97.9 %	99.5 %
Avg. Wireless Goodput	84.9 %	86.8 %	87.8 %
Avg. Timer Expirations	57.55	4.30	0.70

The last two sets of test cases show that this time the snoop protocol experienced a significant number of time-outs on many of the test runs while the number of time-outs with the ACKP protocol remained low. However, the difference in performance between plain snoop and snoop plus ACKP was still not significant. The throughput values for the snoop protocol and the ACKP protocol were almost identical. The biggest difference between the two cases was in wired goodput. Although the goodput for snoop plus ACKP was better, the overall network traffic on the wired link would probably be about the same because of the Ack_p packets being sent between the base station and sender.

The ACKP protocol definitely achieves its objective of reducing the number of unnecessary time-outs at the sender, but this does not result in improved performance for burst errors in either of the environments tested. To see why, one must first look at what caused the degradation in performance during bursts of errors. The causes are really the same factors which prevent snoop from achieving ideal performance, but they are accentuated when packets are lost in bursts. Re-transmitting a whole set of damaged packets takes time. It takes time for the base station to detect lost packets and the detection time becomes greater when many packets are lost because less feedback (or no feedback) is provided by the receiver. Also, unless the receiver's window size is fairly large (32 KB or more) the sender will generally stall while the burst of lost packets is delivered. The combination of these factors is what results in the unnecessary time-out at the receiver. Eliminating the unnecessary time-outs does not improve performance for the cases tested because it can not eliminate these factors which are the primary reasons for performance degradation.

The other reason the ACKP protocol failed to achieve a performance improvement is because both the wireless LAN and the low bandwidth wireless environment had low delay wired links in combination with low delay wireless links. The time-outs which the ACKP protocol prevents will only have a significant detrimental effect on throughput when the delay-bandwidth product is quite large. Satellite connections and perhaps some combination of high delay wired and wireless WANs would fit this description. Unlike LAN connections which only require a window size of

several kilobytes to obtain high transfer rates, these types of connections require a large window on the order of 100 KB to obtain reasonably high throughput. They are therefore extremely sensitive to any unnecessary congestion avoidance measures which shrink the window. With round-trip-times of several hundred milliseconds, Slow-Start can take seconds, and the Congestion-Avoidance phase may prevent the window from returning to its full size for several minutes.

6.0 Conclusions

This project has provided me with an excellent opportunity to study the way in which reliable transport layers such as TCP operate. I have gained a deeper understanding of the problems that heterogeneous networks cause for TCP which is unable to distinguish between congestive and wireless losses and I have had the chance to study a number of the different methods proposed by others to improve TCP's end-to-end performance. Reviewing these proposed improvements has given insight into where the potential gains exist and the drawbacks associated with each approach. The most important aspect of the problem which was learned during my studying and experimentation was that it is generally not the congestion avoidance mechanisms themselves that degrade performance in heterogeneous networks. Instead, it is the secondary effect they have of reducing the congestion window size to the point that Fast-Recovery can no longer operate and the transfer stalls when a packet is lost.

One of the most significant results of this project has been the successful development of a test environment for the Mobile Computing Group here at Texas A&M. Wireless configuration and testing programs have been implemented, an error model has been integrated into FreeBSD, and a method for measuring and displaying results has been developed. This provides a foundation which will allow future work to begin right from the protocol implementation phase.

Finally, this project successfully implemented and tested a completely new TCP enhancement, the ACKP protocol. The ACKP protocol accomplished its goal of reducing the number of time-outs, but it did not significantly improve performance for the environments in which it was tested. It is believed that the ACKP protocol will offer significant gains in throughput for high delay-bandwidth connections such as satellite links, but this remains an area that has yet to be explored. Even if the ACKP protocol does not result in improved performance in other test environments, it has still been a significant accomplishment because it has provided us with a framework which may be followed in developing and testing other protocols designed to enhance TCP's performance.

7.0 Future Work

At this point there are several possible paths which may be pursued in future work. The first is to continue experimenting with the ACKP protocol to see how it performs in other network topologies. As discussed in the results, it is likely that the ACKP protocol can still yield significant performance improvements in terms of throughput and goodput on high bandwidth,

long delay paths such as satellite links. The current test set-up provides high bandwidth over the wireless link, but a method would need to be developed to introduce a long delay since the current delay over the WaveLAN is on the order of several milliseconds. A high bandwidth, high delay wireless link could be simulated by introducing a kind of shift register at the base station which the packets must pass through before being sent on the wireless link. The drawback to this approach is that it is starting to get away from the original objective of experimenting with protocols using a real system rather than simulations.

End-to-end solutions are another potential area for future work. This would most likely involve using preliminary SACK implementations as the base case. New improvements such as our proposed ELN mechanism discussed in section 2.3.3 would then be added to the basic SACK implementation. This combination would combine SACK's efficiency at responding to losses with ELN's ability to distinguish between congestion and wireless losses. Studies would also have to be performed to determine what percentage of packets with errors actually make it to the TCP layer at the receiver on actual wireless networks in use today since the percentage must be significant in order for our proposed ELN scheme to work.

A third area which may be explored in future work is the development of a protocol that operates at the base station but requires less state per connection than the snoop protocol. The idea would be to cache sequence numbers rather than packets at the base station. Duplicate acknowledgments could be shielded from the sender just like with the snoop protocol. However, packets would have to be resent from the sender rather than the base station. Therefore, when a packet loss is detected at the base station (either due to duplicate ACKs or local time-outs), it would transmit a type of explicit loss notification to the sender so that the lost packet is resent without invoking congestion avoidance measures.

A fourth and final area for future work is to evaluate the performance of multiple connections when congestive and wireless losses occur. All of the work done in this project and most of the other current research efforts have focused on the performance of a single connection. The type of study proposed would look at how losses on one connection affect the performance of other connections. It would also test the relative performance of existing and new solutions in terms of aggregate throughput for the set of connections at a base station.

8.0 Acknowledgments

I would first like to thank my wife for understanding my desire to return to graduate school and for her support throughout my Masters program. I would next like to thank the members of my committee for their time and support. Dr. Vaidya has been a wonderful committee chair who has always been very accessible and has provided me with guidance and insight on many issues during my project work. The close working relationship among the students within the Mobile Computing Group at Texas A&M has also been a great help. In many cases we have been able to leverage the work of others in the group, and our almost daily discussions have been very insightful. I am especially grateful for Miten Mehta who installed FreeBSD, the WaveLAN drivers, and other related software on our testbed machines. I would like to thank Hari

Balakrishnan at UC Berkeley for making the snoop source code available and answering questions I had about their implementation. The research of the Mobile Computing Group at Texas A&M is funded in part by Texas Advanced Technology Grant 009741-052-C (also known as 999903-052).

9.0 References

[1] D. C. Cox, "Wireless Personal Communications: What is It?," *IEEE Personal Communications Magazine*, Vol. 2, Issue 2, April 1995, pp. 20-35.

[2] V. Jackobson and M. J. Karels, "Congestion Avoidance and Control", *Proceedings of SIGCOMM '88*, August 1988.

[3] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", *Proceedings of ACM SIGCOMM '96*, August 1996.

[4] A. DeSimone, M. C. Chuah, and O. C. Yue, "Throughput Performance of Transport-Layer Protocols over Wireless LANs", *Proceeding of Globecom '93*, December 1993.

[5] B. S. Bakshi, P. Krishna, N. H. Vaidya, D. K. Pradhan, "Improving Performance of TCP over Wireless Networks", *Texas A&M University Technical Report TR-96-014*, May 1996, (To be presented at ICDCS '97).

[6] S. Biaz, M. Mehta, S. West, N. Vaidya, "TCP Over Wireless Networks Using Multiple Acknowledgments", *Texas A&M University Technical Report TR-97-001*, January 1997.

[7] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", http://ftp.ee.lbl.gov/floyd/sacks.html, May 1996.

[8] J. Hoe, "Improving the Start-Up Behavior of a Congestion Control Scheme for TCP", *SIGCOMM '96 Symposium on Communications Architectures and Protocols*, August 1996.

[9] M. Mathis, J. Jahdavi, S. Floyd, A. Romanov, "TCP Selective Acknowledgment Options", *RFC 2018*, October 1996.

[10] A. V. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", *Technical Report DCS-TR-314*, Dept. of Computer Science, Rutgers University, October 1994.

[11] R. C. Durst, G. J. Miller, and E. J. Travis, "TCP Extensions for Space Communications", *Proceedings of MOBICOM '96*, pp. 15-26, November 1996.

[12] A. V. Bakre and B. R. Badrinath, "Implementation and Performance Evaluation of Indirect TCP", *IEEE Transactions on Computers*, Vol. 46, Number 3, March 1997, pp. 260-278.