IMPROVING TCP PERFORMANCE BY ESTIMATING ERRORS IN A LONG DELAY, HIGH ERROR RATE ENVIRONMENT

THESIS

Stephanie E. Carroll, SMSgt, USAF

AFIT/GCS/ENG/04-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCS/ENG/04-04

IMPROVING TCP PERFORMANCE BY ESTIMATING ERRORS IN A LONG DELAY, HIGH ERROR RATE ENVIRONMENT

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Systems

Stephanie E. Carroll, BS

SMSgt, USAF

July 2004

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/GCS/ENG/04-04

IMPROVING TCP PERFORMANCE BY ESTIMATING ERRORS IN A LONG DELAY, HIGH ERROR RATE ENVIRONMENT

Stephanie E. Carroll, BS

SMSgt, USAF

Approved:

//SIGNED//	22 July 2004
Dr. Rusty O. Baldwin (Chairman)	Date
//SIGNED//	22 July 2004
Dr. Richard A. Raines (Member)	Date
//SIGNED//	22 July 2004
Dr. Henry B. Potoczny (Member)	Date

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr Rusty Baldwin, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated. His encouragement directly aided in completion of this project.

Stephanie E. Carroll

Table of Contents

Page

List of Figures

Page

List of Tables

Page

Abstract

Interest in finding methods of improving TCP performance over satellite and wireless networks is high. This has been an active area of research within the networking community. This research develops an algorithm, CETEN-R for TCP to determine if a particular packet is lost due to congestion or corruption and react accordingly. An analysis of the performance of CETEN-R under a variety of conditions is studied and then compared to TCP Reno and TCP New Reno. When delay is high and the error rate is high CETEN-R showed a 77.5% increase in goodput over TCP New Reno and a 33.8% increase in goodput over TCP Reno. When delay is low and the error rate is high, CETEN-R showed a 146% increase in goodput over TCP New Reno and a 77% increase in goodput over TCP Reno. At low error rates, CETEN-R provides no advantage over TCP Reno or TCP New Reno.

IMPROVING TCP PERFORMANCE BY ESTIMATING ERRORS IN A LONG DELAY, HIGH ERROR RATE ENVIRONMENT

I. Introduction

Background

The Transmission Control Protocol (TCP) is the de facto standard transport protocol for Internet transmissions requiring reliable delivery, such as file transfer, web browsing, and e-mail. TCP was designed for and works quite well in a traditional wired network. However, an active research area in the TCP field has been improving TCP performance over satellite networks. Satellite and other wireless networks have unique characteristics which impact the performance of TCP. In general, satellite communications and other wireless networks offer lower link speeds than traditional fixed networks. They also suffer from higher loss rates due to corruption and long latency. The long latency problem is particularly pronounced for networks using geostationary satellites.

Improving TCP performance is of particular interest to the Department of Defense (DoD) and Air Force (AF) communities. As warfare becomes more information based and net-centric, and as it becomes more crucial for deployed forces to be able to "reachback" from long distances to Continental United States (CONUS) for logistics support and critical up-to-date information about the battlespace, it is vital that methods

be developed to improve the performance of TCP over satellite and other wireless networks.

Research Goals

The primary purpose of this research is to improve the performance of TCP over satellite communications networks. TCP treats all lost packets as congestion losses. In theory, if TCP can accurately determine the actual cause of a loss, either corruption or congestion, performance in a high error rate environment can be improved by just retransmitting the corrupted packets and avoiding going into the standard slow start mode TCP uses to control congestion.

An algorithm is developed to permit IP to notify TCP clients of the probability that a packet was lost due to corruption. The TCP client uses this information to infer whether a particular packet loss was due to corruption or congestion. If TCP determines the packet was lost due to errors, it retransmits a single packet.

To evaluate algorithm performance, a satellite network is simulated using the Optimum Network Performance (OPNET) simulator. OPNET's standard TCP/IP models are modified to accommodate the algorithm and experiments are run using multiple error rates, latency times, number of TCP flows and versions of TCP. The main result of this study is a comparison under various conditions of the impact of the algorithm on TCP performance.

Document Overview

This document is organized as follows. This chapter provides an overview of the problem; TCP over a long latency network with a high error rate, such as a satellite communications network. Chapter 2 is a literature review of prior research into TCP performance over satellite networks. Chapter 3 describes the methodology used including parameters and workload factors selected and experimental design. Chapter 4 discusses the results and analysis of those results. Conclusions of this research are presented in Chapter 5.

II. Literature Review

Chapter Overview

The purpose of this chapter is to review the characteristics of satellite channels that impact TCP performance and the TCP mechanisms that can mitigate those effects. First, specific satellite channel characteristics, in particular delay and latency, and their impact on TCP performance are examined. Then, TCP congestion control mechanisms are reviewed. Next, an overview of the characteristics of the commonly deployed versions of TCP is provided. Methods of detecting corruption, including Explicit Congestion Notification and Forward Error Correction are examined. Transport layer approaches to detecting corruption are also reviewed.

Satellite Channel Characteristics

Satellite networks have several characteristics that degrade the performance of TCP/IP. First, the propagation delay inherent in satellite communications networks results in users suffering long latencies. For geosynchronous satellites at an altitude of 36,000 kilometers, round trip propagation delay ranges from approximately 240 milliseconds if the ground station is directly below the satellite to 280 milliseconds if the ground station is at the edge of the view area for that satellite. This accounts for one ground station to satellite to ground station hop. In the TCP/IP protocol, messages require acknowledgement. This means the message and its corresponding reply will be delayed by at least 560 milliseconds [AGS99]. However, this is not the only component of delay. Other delay factors include queuing delay, processing time (both at ground

stations and onboard the satellite itself), transmission time, and the propagation delay of any other links in the network path.

Other satellite characteristics that degrade the performance of TCP include the long feedback loop, large delay times bandwidth product, and transmission errors. The long feedback loop caused by the propagation delay of approximately 250 milliseconds over geostationary satellites can result in a significant delay for a TCP node to determine if a packet was received successfully. This results in poor performance of interactive applications such as telnet or http and also adversely affects some TCP congestion control algorithms [AGS99].

Delay times bandwidth product is the amount of data that is in flight (i.e., data that has been transmitted, but not yet acknowledged). The delay in this case is the round trip time, or approximately 500 milliseconds and the bandwidth is the capacity of the satellite link. Since the delay component in geostationary satellite networks is large, the delay times bandwidth product will also be large and a large number of packets must be in flight to use the channel efficiently [AGS99].

Many satellite links have a higher bit error rate than typical terrestrial links. TCP interprets all packet loss as an indication of network congestion and reduces its window size to reduce congestion in the network. In a relatively high error environment, this can results in the sliding window being reduced even though the packets were dropped due to errors and not congestion in the network [AGS99].

TCP Congestion Control Mechanisms

Overview.

The higher bit error rate typical of satellite links poses a particular obstacle to good TCP/IP performance. TCP provides reliable data to applications by guaranteeing that corrupted or missing packets will be retransmitted. But, TCP uses packet loss as an indicator of the level of congestion in the network. In a traditional terrestrial network packet loss is mainly due to buffer overflows caused by congestion. TCP responds to packet loss by decreasing the size of the congestion window. The congestion window is essentially an estimate of the available bandwidth on the path to the receiver. The size of the window is increase or decreased based on the current estimate of congestion in the network [KAP02].

The basic congestion control algorithm works in the following manner. Every round trip time that an acknowledgment (ACK) is received, TCP increases the congestion window by one maximum-sized segment. If the sender determines a packet was lost, it assumes the loss occurred due to network congestion so it halves the size of the congestion window. Transmission resumes increasing the size of the congestion window by one each round trip time that an ACK is received. However, over geostationary satellite links packet loss is more likely due to bit errors instead of congestion. The standard congestion algorithm results in the available bandwidth being underestimated; the TCP algorithm, as implemented, takes a long time to recover bandwidth capacity [PaS97]. The end result is poor performance of the network.

Slow Start.

TCP sessions begin with the slow start phase. As detailed in [Jac88, Ste97], the slow start algorithm initializes the congestion window (*cwnd*) to 1 segment (1**smss*), where *smss* is the maximum segment size in bytes. TCP transmits one segment and then waits until it receives and ACK for that segment; at this point the *cwnd* is increased by one segment. During slow start, *cwnd* increases approximately exponentially as follows: TCP starts by sending one packet; when this packet is successfully acknowledged, *cwnd* is incremented from one to two. When those two packets are acknowledged, *cwnd* is increased from two to four. This process continues until *cwnd* is equal to or exceeds the slow start threshold (*ssthresh*), which is initially set equal to the size of the receiver's advertised window, unless loss is detected.

Congestion Avoidance.

Congestion on a network can occur for two reasons: 1) packets arrive on a large pipe, but get sent out on a smaller pipe; 2) multiple TCP connections arrive at a router whose output capacity is smaller than the sum of the connections arriving at that router. Congestion avoidance [Jac88, Ste97] is the method used to deal with these problems and is entered when the value of *cwnd* is greater than or equal to *ssthresh*. The purpose of congestion avoidance is to slowly probe the network for additional bandwidth; during this phase the size of *cwnd* increases linearly instead of exponentially as during slow start. Congestion avoidance increases *cwnd* by 1/*cwnd* whenever an ACK is received. During congestion avoidance, *cwnd* increases by a maximum of one segment size each round trip time regardless of how many ACKs are received.

According to [AGS99], slow start and congestion avoidance result in poor bandwidth utilization of satellite networks. For example, on a gigabit per second geostationary satellite link with a 500 millisecond round trip time, it takes 29 round trip times (14.5 seconds) to complete the slow start phase [PaS97]. Additionally, the entire data transfer is often complete before slow start is finished, especially with bursty traffic such as a web transfer [PaS97].

Fast Retransmit.

TCP normally uses timeouts to detect dropped segments. If TCP does not receive an acknowledgement for a packet within an expected time, the packet is retransmitted. This expected time, or retransmission timeout (RTO) is based upon observed round trip time. When the RTO expires, TCP retransmits the lost segment and halves the size of *cwnd* and reenters slow start [AGS99].

A TCP ACK acknowledges the highest in-order segment received; it also implicitly acknowledges all segments which are less than that segment number. Because TCP is required to generate an immediate acknowledgement when an out of order segment is received, this leads to duplicate acknowledgements. As detailed in [Jac90, Ste97], fast retransmit uses duplicate ACKs to detect lost segments. In particular, the fast retransmit algorithm retransmits a segment if it receives 3 duplicate ACKs, even if the RTO has not expired [Ste97].

Fast Recovery.

In most TCP implementations, fast retransmit is implemented together with fast recovery. The fast recovery algorithm [Ste97] performs congestion avoidance after a fast

retransmit; slow start is not performed. The fast recovery algorithm adjusts the congestion window in the following manner as described in [Ste97]. After the third duplicate ACK, set *ssthresh* equal to *cwnd*/2, but not less the two segments and then retransmit the missing segment. Set *cwnd* equal to *ssthresh* plus number of duplicate ACKs (typically three) times *smss*. This accounts for the segments being cached at the TCP receiver. Whenever another duplicate ACK is received, *cwnd* is incremented by *smss*. This inflation of the congestion window accounts for the segment removed from the network. TCP continues to transmit packets if permitted by *cwnd*. When an ACK for the retransmitted packet is received, *cwnd* is set equal to *ssthresh* and congestion avoidance is entered.

If TCP detects congestion is due to duplicate ACKs, fast retransmit and fast recovery can be used, since TCP can infer congestion is not severe since traffic is still flowing over the network. On the other hand, when a retransmit occurs due to a timeout, TCP cannot infer anything about the status of the network and must resort to slow start or risk congestion collapse [AGS99].

Fast retransmit and fast recovery can result in multiple fast retransmits per window. As [Flo94] showed, if window size is large and multiple nonconsecutive packet losses occur within a window time, multiple fast retransmits can occur resulting in *cwnd* being reduced multiple times for a single loss event and a reduction in performance.

Performance of fast retransmit and fast recovery algorithms are improved by using the selective acknowledgement (SACK) algorithm, which lets TCP receivers inform TCP senders exactly which segments have been received [AGS99].

TCP Variants

Background.

This section describes the evolution of TCP implementations. Early implementations of TCP used a model where the sender would transmit packets on the network up to the receiver's advertised window size. As packets were positively acknowledged, the window would slide to the right. Any data lost during transport could not be retransmitted until the retransmit timer expired. Although these implementations were adequate if the two TCP hosts were on the same network, if there were routers or slower links in between the two hosts, excessive network congestion and network collapse were observed due to buffer overflows [Jac88, Ste97]. [Jac88] showed how this led to a series of congestion collapses on the Internet in 1986.

Tahoe.

TCP Tahoe was the first implementation of TCP to add several algorithms designed to control congestion while maintaining good user throughput. Tahoe added the slow-start, congestion avoidance, and fast retransmit algorithms described in the previous section. In addition, it modified the round-trip time estimator used to set retransmission timeout values [Jac88, Ste94].

Reno.

TCP Reno built upon the enhancements contained in TCP Tahoe by modifying the fast retransmit mechanism to include fast recovery as described in [Jac90, FaF96].

In TCP Reno, the fast recovery algorithm is optimized for the case when a single packet is dropped within a window of data; that is a maximum of one dropped packet per

round-trip time is retransmitted. The performance of TCP Reno is significantly better than that of TCP Tahoe in this case. However, Reno can suffer from performance problems when multiple packets are dropped from a window of data or in the presence of burst errors.

New Reno.

TCP New Reno is based upon TCP Reno, but modifies the fast recovery mechanism slightly [FlH99]. This change involves how TCP Reno behaves when a partial acknowledgement (ACK) is received. In Reno, when a partial ACK, which acknowledges some but not all of the outstanding data, is received during fast recovery, TCP Reno reacts by taking TCP out of fast recovery and back into slow start. In New Reno, partial ACKS do not cause TCP to come out of fast recovery and reinitiate slow start. Instead, when a partial ACK is received during fast retransmit, New Reno treats this as an indication that the packet immediately following the acknowledged packet is lost and needs to be retransmitted. New Reno remains in fast recovery until all segments that were outstanding when fast recovery was entered are acknowledged.

*Selective Acknowledgement (SACK)***.**

TCP SACK is the version of TCP that uses all the standard congestion control mechanisms used in TCP Reno plus the SACK option [MMF96]. When TCP is in fast recovery, SACK maintains a variable called *pipe*, which contains the estimated number of packets or bytes (depending upon the implementation) that are outstanding in the path. Only when the size of the *pipe* is less than the congestion window will a SACK sender transmit new data or retransmit old data. Senders also maintain a data structure called

scoreboard which maintains information about which packets have already been acknowledged. During fast recovery, when a sender is permitted to transmit, it sends the next packet from the list of packets it 'infers' are missing at the receiving end. When this list is exhausted, the sender transmits a new packet if the congestion window is large enough. Fast recovery is terminated when the sender receivers a recovery ACK acknowledging all the data that was outstanding when fast recovery was initiated.

Detecting Losses Due to Corruption

It would be beneficial if congestion and corruption could be differentiated. Segments lost due to buffer overflow are due to congestion, and segments lost due to damaged bits are due to corruption. This is a difficult problem for TCP since TCP treats all lost segments as loss due to congestion [ADG00]. TCP should handle these two situations differently. If congestion in the network is occurring, then adjusting the congestion window is appropriate. On the other hand, if losses in the network are due to corruption, there is no need for TCP to adjust its congestion window; instead it should simply retransmit the bad segment [ADG00].

This problem is particularly prevalent in satellite and other wireless networks where bit error rates are typically higher than in terrestrial networks and loss due to corruption is more common.

Forward Error Correction.

A partial solution to the problem of loss due to corrupted data is to employ forward error correction (FEC). FEC is a coding technique used to improve the bit-error rate performance of a link. FEC is quite commonly used on satellite and other wireless

links. However, it is not always possible to correct bit errors in a satellite network. The goal is to make the link as error-free as possible in order to prevent TCP from incorrectly determining the network is congested [AGS99].

FEC does not come without some costs. It requires additional hardware for encoding/decoding, uses additional bandwidth, and can add delay and timing jitter due to the encoding/decoding processing time. FEC will not solve all problems associated with lossy satellite and wireless links. Problems with noise due to rain attenuation, jamming, or interference cannot be solved by FEC [AGS99].

Explicit Congestion Notification.

Explicit congestion notification (ECN) [Flo94, ADG00, RFB01] is used by routers to notify TCP of imminent congestion without dropping segments. There are two major types of ECN: forward explicit congestion notification (FECN) and backward explicit congestion notification (BECN). In FECN, a router specially marks a packet that congestion is imminent and forwards the packet to the receiver. The receiver echoes the congestion information back to the sender in the ACK message. BECN transmits information about congestion directly to the originator.

Both ECN schemes require the deployment of active queue management schemes such as Random Early Drop [FLJ93, BCC98] in network routers. The routers signal congestion to the sender in the form congestion signs, i.e., segment drops or ECN messages, instead of discarding large amounts of TCP segments. In FECN schemes, TCP transmits segments with an "ECN-Capable Transport" bit set in the IP header of the packet. If the active queue management algorithm would otherwise discard the packet, it

instead sets the "Congestion Experienced" bit in the IP header. The receiver sends the information back to the TCP sender in the ACK message by using a bit in the TCP header. The sender reacts by adjusting the size of the congestion window, just as it would have if the segment had been dropped.

Since satellite networks tend to have higher error rates than terrestrial networks, being able to determine if a packet was lost due to congestion or corruption may result in better TCP performance over satellite networks. This is not a solution to the problem of poor performance in a higher error rate system; rather, ECN can be one part of a means to achieving the goal of better performance.

One advantage of ECN is avoiding unnecessary packet drops for short or delay-sensitive TCP connections [Flo94]. Another advantage of ECN is it avoids some unnecessary retransmission timeouts. A possible drawback of ECN is that a non-compliant TCP connection could falsely advertise itself as ECN-capable, and a TCP ACK packet carrying an ECN-Echo message could itself be dropped in the network.

Experimental evaluations of ECN include [SaA00, K98]. ECN TCP gets moderately better throughput than non-ECN TCP; ECN TCP flows are fair with respect to non-ECN TCP flows; and ECN TCP is robust with two-way traffic (i.e. congestion in both directions) and with multiple congested gateways. Experiments with many short web transfers show that most of the short connections have similar transfer times with or without ECN. However, a small percentage of the short connections have very long transfer times for the non-ECN experiments as compared to the ECN experiments. ECN performance is summarized in Figure 1 through Figure 3.

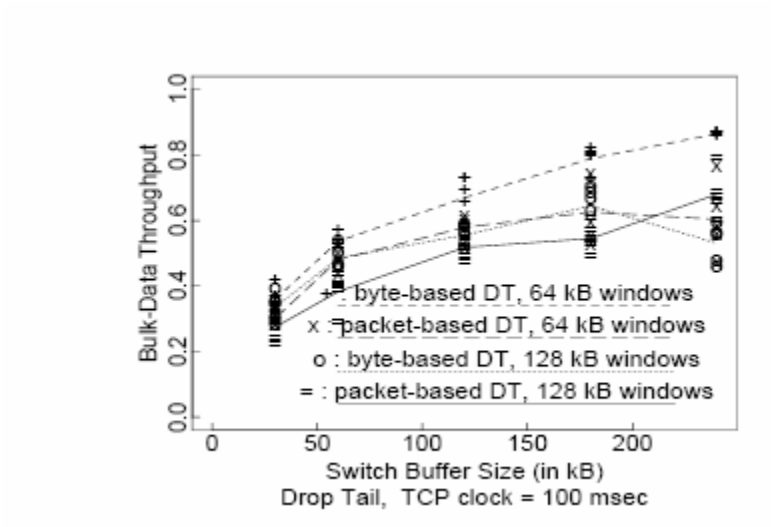

Figure 1. Drop Tail Throughput, 100 ms TCP Clock, Four Connections [Flo94]

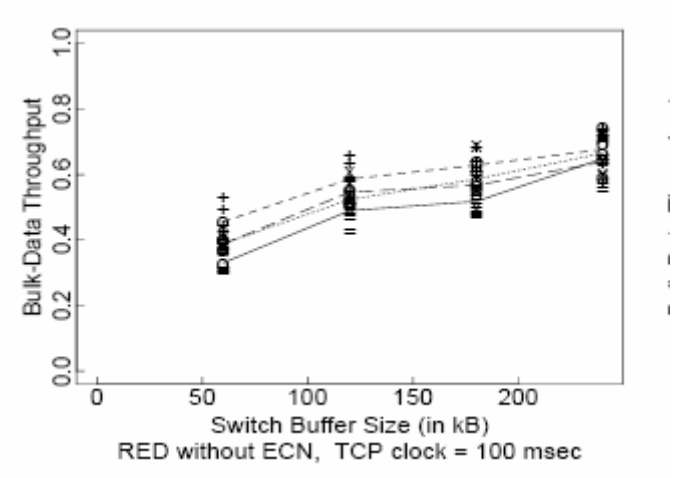

Figure 2. RED without ECN Throughput, 100 ms TCP Clock, Four Connections [Flo94]

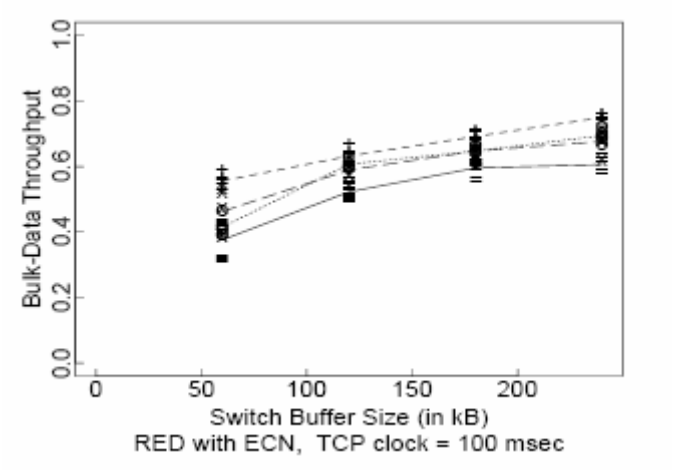

Figure 3. RED with ECN Throughput, 100 ms TCP clock, Four Connections [Flo94]

Transport Layer Approaches

The two approaches discussed thus far to improve TCP performance in a high error rate environment were lower layer approaches. There are also transport layer approaches. There are basically two ways to accomplish error notification. First, TCP can be explicitly notified that errors are occurring and second, TCP can infer that errors are occurring [PaS97]. The first approach is similar to explicit congestion notification used with the IP protocol.

Generally speaking, a challenge with any explicit notification scheme is TCP/IP usually does not know errors are occurring since those packets are discarded by lower layers before they are passed to TCP/IP [PaS97]. The idea is to have TCP retransmit any packets lost due to errors without reducing the size of the congestion window, which

reduces throughput, while still maintaining network stability by reducing the size of the congestion window when congestion actually occurs on the network [KAP02].

Explicit Transport Error Notification.

Explicit Transport Error Notification (ETEN) informs TCP of lost packets due to errors. ETEN is similar to ECN, the difference being ETEN is notifying TCP of errors, while ECN is notifying TCP of congestion in the network [APS03]. There are several types of ETEN: Oracle ETEN, forward ETEN, backward ETEN, forward cumulative ETEN and backward cumulative ETEN [KAP02].

Oracle ETEN is an ideal, although unrealizable ETEN mechanism. Oracle ETEN is a theoretical construct useful in determining cases where ETEN would improve performance and cases where ETEN would provide no improvement to performance. Oracle ETEN makes two assumptions: sufficient information about corrupted packets is available to the device which detects the error and the TCP source can be immediately notified of the packet's corruption. Oracle ETEN provides an upper bound on the performance gains possible using any implemented ETEN scheme.

Forward ETEN operates in a manner similar to FECN. Forward ETEN notifies the TCP receiver about corrupted packets, and returns the corruption information to the sender via TCP Acknowledgement (ACK) packets. Backward ETEN works similar to BECN. In backward ETEN, the router or host that detects a packet with errors directly notifies the source. Cumulative ETEN estimates the error rate in one of several possible ways and lets the sender know by either forward or backward signaling. Cumulative

ETEN information can also be sent along with data and acknowledgement packets [KAP02].

Forward and backward ETEN can be used when the source and destination IP addresses, the source and destination TCP ports, and the TCP sequence number can be determined correctly from the packet. However, due to corruption it is often impossible to determine one or more of these items. In that event, the node detecting errors can only calculate cumulate errors for each link [KAS03].

With cumulative ETEN (CETEN), information about errors on the link can be conveyed to the end hosts in several different ways. An absolute error rate, in terms of bits, bytes, or packets can be observed. The error rate can be categorized as one of a small number of steps, e.g., low, medium, or high. An indication can be sent that the error rate exceeds some threshold. Error rates can be classified relative to some previous value, i.e., the rate has increased or decreased from the previous value. Error rates can be estimated based upon likelihood a packet has not been corrupted [KAS03].

CETEN-specific software modifications have been made for a particular implementation of CETEN [KAP02]. These include addition of fields and access methods to carry corruption and congestion survival estimates; addition of variables and methods to track packet corruption statistics and modify packet headers; and modifications to initialize the CETEN packet header fields and to decide if a packet was lost due to corruption.

Explicit Transport Error Notification Performance Conclusions.

The initial study of ETEN led to some interesting conclusions. The per-packet ETEN mechanisms (forward and backward) showed substantial gain in goodput when the network was not experiencing congestion. The result held over a wide range of link capacities and delays and was observed with both TCP Reno and TCP SACK. An approximate 7-fold improvement was noted in some cases when the error rate was in the range of 10^{-5} to 10^{-7} [KAP02]. Figure 4 displays a summary of ETEN performance over a high delay, high bandwidth network. Figure 5 displays a summary of ETEN performance over a low delay, high bandwidth network.

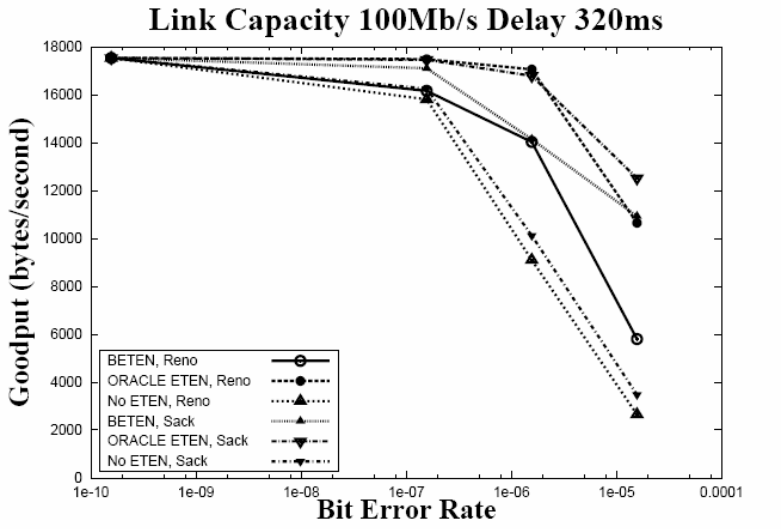

Figure 4. TCP with ETEN Performance, High Delay, High Bandwidth Network [KAP02]

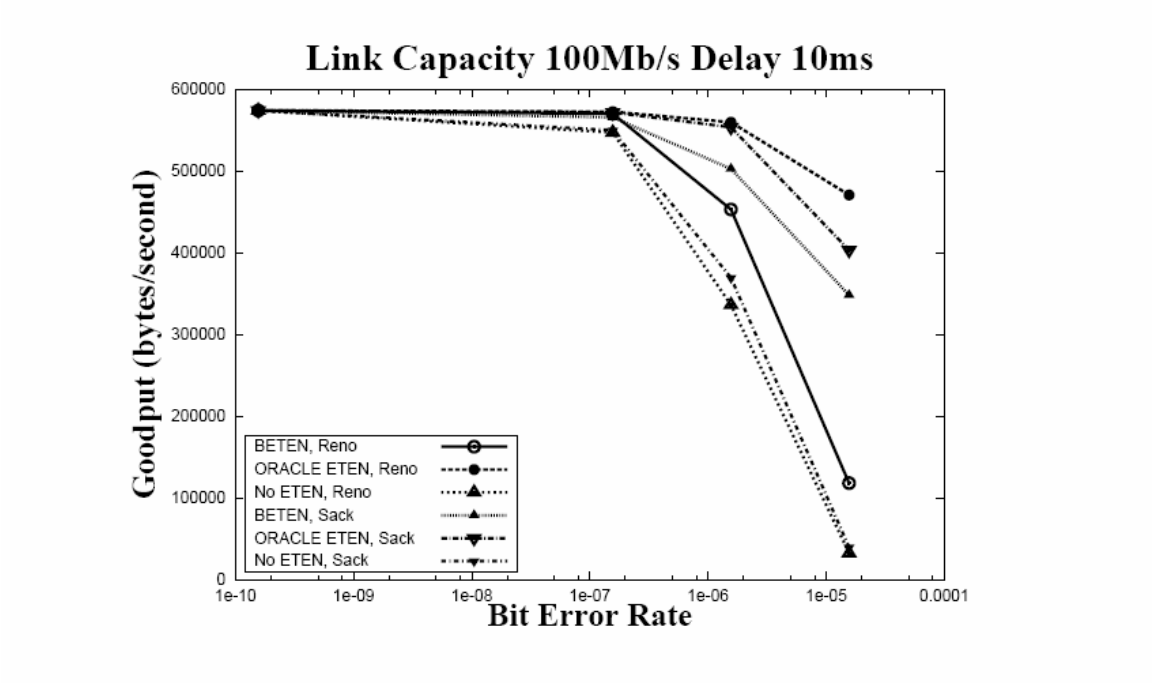

Figure 5. TCP with ETEN Performance, Low Delay, High Bandwidth Network [KAP02]

Gains observed with forward and backward ETEN are not significant when the network is congested. ETEN, by design, defers to TCP congestion avoidance in this case and this result is expected. ETEN is mostly likely to be beneficial in high-error, uncongested networks [KAP02].

Gains observed with CETEN showed moderate improvement over TCP Reno except at high error rates. As with the per packet ETEN mechanisms, CETEN provides greater performance improvements when there is less congestion [KAP02]. CETEN appears to be a promising approach in some situations. The greatest challenge to developing an effective CETEN scheme is the inability of TCP endpoints to estimate within a few packets of accuracy the total loss and to do so in a timely manner [KAP02].

Figure 6 summarizes CETEN performance with TCP Reno and User Datagram Protocol (UDP) crossflows.

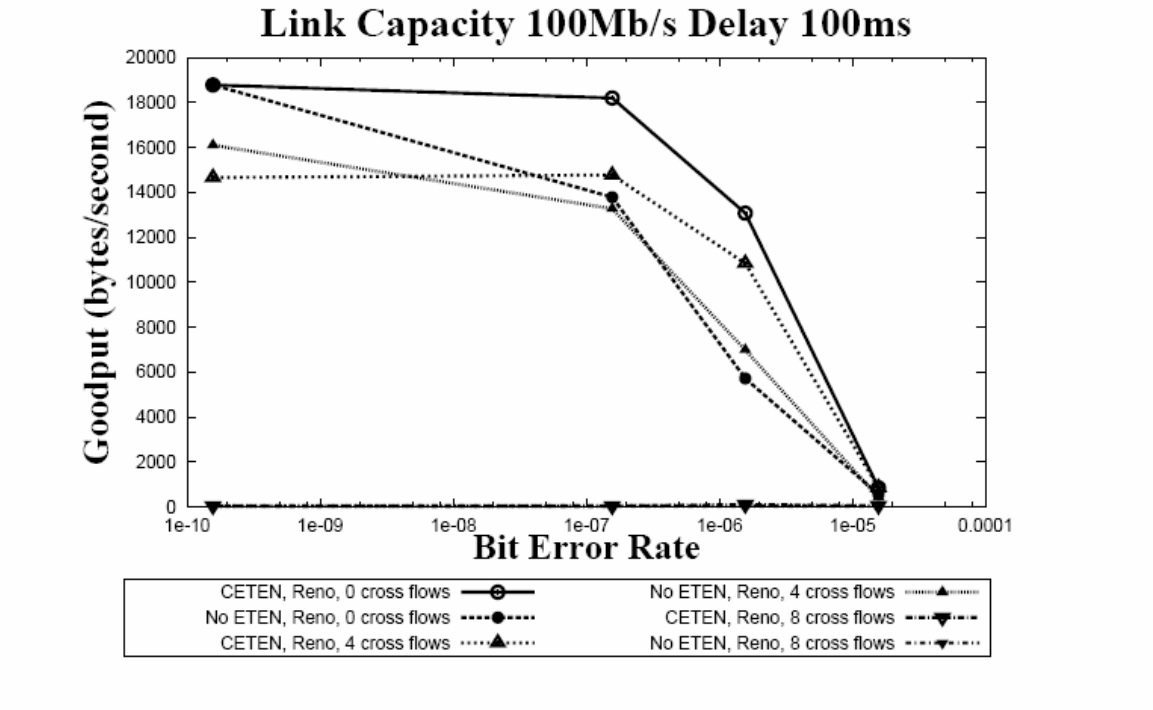

Figure 6. CETEN Performance with TCP Reno and UDP Crossflows [KAP02]

Summary

In this chapter, an overview of the operation of TCP in a satellite communications environment of high error rates and high latency was provided. TCP congestion control mechanisms, slow start, congestion avoidance, fast retransmit and fast recovery were discussed. The versions of TCP in used on operational networks were explained. Finally, Explicit Congestion Notification and Explicit Transport Error Notification mechanisms for detecting congestion and corruption were discussed.

III. CETEN-R Algorithm Description

Overview

TCP retransmits lost packets under two conditions: (1) when it receives a set number of duplicate acknowledgements, typically three, and (2) upon timeout. TCP treats all lost packets as lost due to congestion in the network. The goal of CETEN-R is to determine if a packet was lost due to corruption and retransmit just the dropped packet.

CETEN-R Algorithm

Instead of treating all lost packets as lost due to congestion, CETEN-R attempts to determine when a packet was lost due to the packet being corrupted. CETEN-R is only used when a packet is being retransmitted due to duplicate acknowledgements being received. When a duplicate acknowledgement is received, TCP determines whether it is a true duplicate. If the packet is truly a duplicate, then TCP checks to see if CETEN-R is enabled and if the packet was lost due to corruption on the network. If CETEN-R determines the packet was lost due to corruption, it retransmits just the first unacknowledged packet and then continues transmitting data normally, bypassing the normal congestion control mechanisms.

In OPNET, in the *tcp_conn_v3* process model, *tcp_ack_check* function, once TCP determines a packet is a true duplicate, the CETEN-R mechanism determines if CETEN-R is turned on. If CETEN-R is enabled, then a new function *tcp_ceten_packet_loss* is called to determine if the packet was lost due to congestion or corruption. If this function determines the loss was due to corruption, then the congestion control mechanisms will

be bypassed and *tcp_eten_retransmit* is called. This function sends the first unacknowledged segment and after the segment is sent restores the values of the next segment to send and the congestion window to the state they were in before the function was called, thereby bypassing the congestion control mechanisms. See Appendix A for the OPNET implementation.

To support this algorithm, the *tcp_seg_sup* header file and C file are modified to add fields to the TCP header to track the corruption status of the link and functions to set the values of these fields. This information is passed to and from the Internet Protocol (IP) layer by modifying existing Interface Control Information (ICI) formats to account for this information. When a packet arrives, the *ip_rte_central_cpu* process model uses information about the state of the link and modifies the CETEN-R information in the ICI, which is passed back up to TCP where it is used to determine whether a packet was lost due to corruption or congestion. In OPNET, the state of the link is determined by using built in functions to determine if a packet has errors. See Appendix A for the OPNET implementation.

Summary

This chapter describes the CETEN-R algorithm. This algorithm uses information gathered from IP about the state of the network to determine whether a packet was lost due to corruption or congestion.

IV. Methodology

Chapter Overview

The purpose of this chapter is to identify the goals of this thesis and to describe the system and component under tests, selected metrics, system services, system and workload parameters, and selected factors. Finally, the experimental design and evaluation techniques are described.

Problem Definition

Goals and Hypothesis.

The primary goal of this study is to evaluate the improvement in goodput, i.e. user throughput less loss due to congestion and corruption, achieved by implementing a CETEN scheme, hereafter known as CETEN-R over TCP/IP networks on satellite and other large delay*bandwidth product networks. The secondary goal of this study is to determine the feasibility of implementing the CETEN-R scheme within the existing TCP/IP framework.

It is expected that goodput will improve with CETEN-R enabled. Further it is expected that the improvement will be greater at higher error rates, since TCP itself performs well at low error rates.

Approach.

A bulk data flow is sent across a TCP network to determine the goodput of the system with CETEN-R enabled compared to off the shelf versions of TCP. A bulk data flow will provide the most stress to the system, since over satellite networks an http transaction is usually complete while the system is still in the slow start phase.

System Boundaries

The System Under Test (SUT) in this study is a satellite communications network. A block diagram of a notional system is provided in figure 1. The system includes the following:

i) TCP endpoint(s)

ii) 1 IP Hubs

ii) 2 IP routers

iii) 2 Satellite Modems

iv) 2 Satellite Terminals

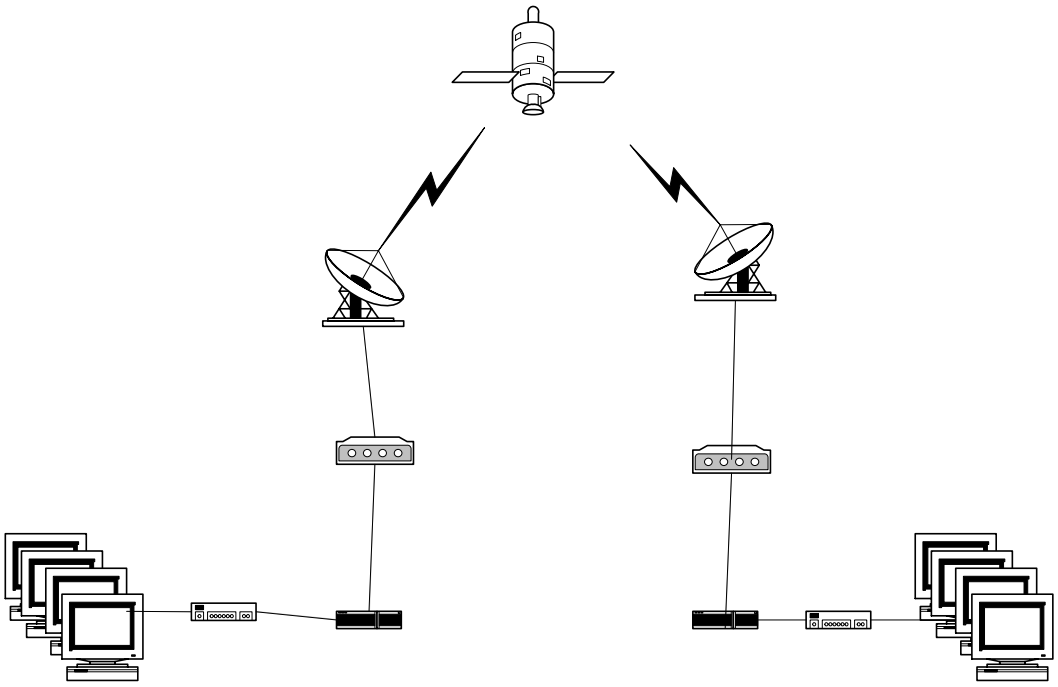

Figure 7. Block Diagram of System Under Test

In the OPNET simulation, a simple model is constructed which takes into account the satellite portion of the network. The satellite network is simulated by appropriately configuring the router and link characteristics. The TCP endpoint at one end is replaced by an FTP server. In the simulation model, TCP endpoint(s) transmit bulk data flows over the network to the distant end; the FTP server is the endpoint device which processes the file transfers. Figure 8 shows the network model.

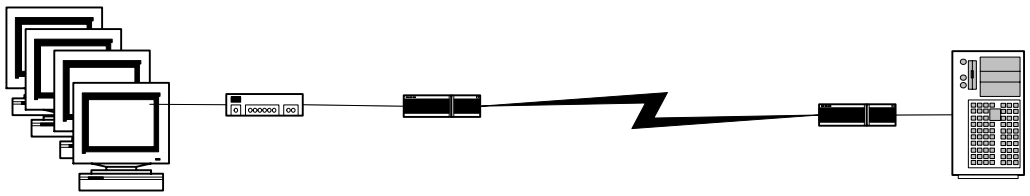

Figure 8. Block Diagram of the OPNET Network Model

The Component Under Test (CUT) is the CETEN-R algorithm itself. This algorithm extends the work done by [KAP02] and is an extension to TCP.

System Services

The network provides packet data transfer from a TCP source node to a TCP destination node over a simulated satellite link. There are three possible outcomes of this service. First, a packet can successfully reach its destination without error. Second, a packet can be dropped, i.e., lost due to congestion. Finally, a packet can reach the destination, but be corrupted due to errors on the transmission link.

IP Hub IP Router

TCP Endpoint(s)

Metrics

The primary metric for this research is goodput; the measure of the amount of user data in bits per second successfully transferred across the network. This metric most fully supports the main goal of the study; to measure the performance enhancement obtained by the CETEN-R algorithm.

The OPNET simulator statistic, throughput (bits/second), is computed by dividing the number of bits successfully by a receiver to date by the current simulation time. This statistic automatically takes into account packets which are dropped or contain errors and is actually equal to goodput.

Parameters

Parameters are the characteristics of the system and the workload that affect system performance [Jai91].

System Parameters.

The system parameters of interest are:

i) Link capacity - the amount of data (10 Mb/s) that can be transmitted on a link in a given period of time and provides a limit on throughput.

ii) Error rate – the rate of data corruption on a link affects network performance.

iii) TCP variant – network performance is affected by the version of TCP used.

iv) CETEN-R - whether CETEN-R is enabled or not affects network performance

v) Delay – for a single hop geostationary satellite network, this figure is a minimum of 250 milliseconds and affects network performance.

Workload.

The workload parameters are:

i) Number of clients – varying the number of clients can have an impact on the performance of the CETEN-R algorithm and on the network; one client and four clients are used.

ii) Type of application – short bursty traffic, such as http; long file transfer. In this research a long file transfer is used to provide the most stress to the system.

Factors

The factors selected for this research are shown in Table 1:

Table 1. Experimental Factors

FACTOR	LEVELS
Error Rate	Low (10^{-7}), Medium (10^{-6}), High (10^{-5})
TCP Variant	Reno, New Reno
CETEN-R	On, Off
Delay	50 ms, 320 ms
# Clients	1, 4

Error rate was selected since the purpose of the CETEN-R algorithm is to improve the performance of TCP in a high error rate environment such as a satellite network. Since TCP performs well in a low error rate environment, it is expected that that performance will improve with CETEN-R enabled as the error rate increases. Error rates of 10^{-7}, 10^{-6} and 10^{-5} are studied. During pilot testing it was discovered that for error rates higher than 10^{-5}, TCP collapsed and even the addition of CETEN-R did not prevent collapse. For error rates lower than 10^{-7}, pilot testing revealed TCP worked quite well on its own and addition of the CETEN-R mechanism provided no benefit.

TCP variants selected were TCP Reno and TCP New Reno. TCP Reno was selected since it is still a commonly used version of TCP. TCP New Reno was selected as it performs better than TCP Reno and is becoming more commonly deployed over the Internet. It is expected that performance will improve with CETEN-R enabled.

The number of clients is selected to evaluate how the algorithm behaves in the presence of multiple clients. One client is selected as a baseline for performance. Four clients are selected to evaluate the performance of CETEN-R in a multiple flow environment. It is expected that total goodput will be higher with more clients; however, per client goodput may be lower as the system becomes more congested.

Evaluation Technique

The evaluation technique is simulation of the system. The simulation is developed in OPNET. Direct measurement was impossible; it was not feasible or cost effective to obtain satellite time. A simulation was deemed more valuable than an analytical model as simulation allows measurement and analysis of how a proposed change to an existing protocol could affect the network.

Workload

Two different workloads are offered to the system. The small workload consists of one client transmitting a TCP flow. The TCP flow will be configured to simulate continuous traffic, such as a large file transfer. The high workload consists of four clients. All flows are configured to simulate large file transfers. As discussed earlier, large files are used instead of short bursty traffic, since over satellite links transmission of

short, bursty traffic usually completes before the slow start phase is completed [AGS99]. During pilot testing, it was discovered a single 500 Mb file would provide the maximum load possible to the system, allowing packets to be transmitted for the duration of the simulation, without causing unacceptable performance constraints from the OPNET simulator.

Experimental Design

There are two groups of experiments performed for this research. First, a full factorial experimental design with 48 experiments with 5 replications is chosen for this research. Each TCP variant, delay, error rate and number of clients combination is tested with CETEN-R on and off. A second group of 12 experiments with 5 replications is performed where each TCP variant, delay, and error rate combination is tested for 4 clients with one CETEN-R enabled client and three ordinary TCP clients. Test cases 1-24 are shown in Table 2; test cases 25-60 are shown in Table 3.

Summary

To determine the performance of a TCP satellite network with CETEN-R enabled, a system is simulated using OPNET 10.0. The factors selected are error rate, number of clients, client type, and CETEN-R enabled or disabled. A full factorial, 5 replication set of tests is run to determine the effect of the various factor levels.

The key metric of goodput is examined over a range of error rates, number of clients and types of clients to evaluate the performance of the satellite network with

CETEN-R enabled and disabled. The results are analyzed to determine the impact CETEN-R has on network performance.

Table 2. Test Cases 1-24.

Test #	# Clients	TCP Flavor	Error Rate	Delay (ms)	CETEN-R
1	1	Reno	10^{-7}	50	Off
2	1	Reno	10^{-6}	50	Off
3	1	Reno	10^{-5}	50	Off
4	1	Reno	10^{-7}	50	On
5	1	Reno	10^{-6}	50	On
6	1	Reno	10^{-5}	50	On
7	1	Reno	10^{-7}	320	Off
8	1	Reno	10^{-6}	320	Off
9	1	Reno	10^{-5}	320	Off
10	1	Reno	10^{-7}	320	On
11	1	Reno	10^{-6}	320	On
12	1	Reno	10^{-5}	320	On
13	1	New Reno	10^{-7}	50	Off
14	1	New Reno	10^{-6}	50	Off
15	1	New Reno	10^{-5}	50	Off
16	1	New Reno	10^{-7}	50	On
17	1	New Reno	10^{-6}	50	On
18	1	New Reno	10^{-5}	50	On
19	1	New Reno	10^{-7}	320	Off
20	1	New Reno	10^{-6}	320	Off
21	1	New Reno	10^{-5}	320	Off
22	1	New Reno	10^{-7}	320	On
23	1	New Reno	10^{-6}	320	On
24	1	New Reno	10^{-5}	320	On

Table 3. Test Cases 25-60

Test #	# Clients	TCP Flavor	Error Rate	Delay (ms)	CETEN-R
25	4	Reno	10^{-7}	50	Off
26	4	Reno	10^{-6}	50	Off
27	4	Reno	10^{-5}	50	Off
28	4	Reno	10^{-7}	50	On
29	4	Reno	10^{-6}	50	On
30	4	Reno	10^{-5}	50	On
31	4	Reno	10^{-7}	320	Off
32	4	Reno	10^{-6}	320	Off
33	4	Reno	10^{-5}	320	Off
34	4	Reno	10^{-7}	320	On
35	4	Reno	10^{-6}	320	On
36	4	Reno	10^{-5}	320	On
37	4	New Reno	10^{-7}	50	Off
38	4	New Reno	10^{-6}	50	Off
39	4	New Reno	10^{-5}	50	Off
40	4	New Reno	10^{-7}	50	On
41	4	New Reno	10^{-6}	50	On
42	4	New Reno	10^{-5}	50	On
43	4	New Reno	10^{-7}	320	Off
44	4	New Reno	10^{-6}	320	Off
45	4	New Reno	10^{-5}	320	Off
46	4	New Reno	10^{-7}	320	On
47	4	New Reno	10^{-6}	320	On
48	4	New Reno	10^{-5}	320	On
49	4	Reno	10^{-7}	50	Mixed
50	4	Reno	10^{-6}	50	Mixed
51	4	Reno	10^{-5}	50	Mixed
52	4	Reno	10^{-7}	320	Mixed
53	4	Reno	10^{-6}	320	Mixed
54	4	Reno	10^{-5}	320	Mixed
55	4	New Reno	10^{-7}	50	Mixed
56	4	New Reno	10^{-6}	50	Mixed
57	4	New Reno	10^{-5}	50	Mixed
58	4	New Reno	10^{-7}	320	Mixed
59	4	New Reno	10^{-6}	320	Mixed
60	4	New Reno	10^{-5}	320	Mixed

V. Analysis and Results

Chapter Overview

Data was analyzed in two ways: (1) as one client and four client experiments analyzed separately and (2) as an overall system. Additionally, holding error rate, delay and CETEN-R constant, one client and four client experiments were compared to each other to assess the impact of adding cross flows on CETEN-R algorithm performance.

The one client experiments are paired and for each error rate, delay and TCP version the mean throughput for the CETEN-R enabled experiments is compared to the mean throughput for TCP Reno and TCP New Reno. Then an analysis of variance (ANOVA) is performed on the one client experiments. Next, the four client experiments are paired and for each error rate, delay and TCP version the mean throughput for the CETEN-R experiments is compared to the mean throughput for TCP Reno and TCP New Reno. Then an ANOVA is performed on this group of experiments. An ANOVA is performed on all the experiments to determine the impact of the various factors on TCP performance. Finally, a group of four client experiments where some clients are using CETEN-R and others are not is analyzed to determine if CETEN-R is overly aggressive.

Analysis is only conducted for error rates of 10^{-5}, 10^{-6} and 10^{-7}. As noted in Chapter III, during pilot testing for error rates of 10^{-5} and lower, TCP experienced congestion collapse and addition of the CETEN-R algorithm did not change this result.

Model Verification and Validation

A group of pilot studies of a simple model with one client transmitting a large file to an FTP server using standard TCP Reno were conducted at error rates ranging from 10^{-3} through 10^{-10} and delay of 320 ms were conducted. The results were compared to the results obtained by [KAP02] for similar experiments to determine that the modifications to the OPNET process models did not impact the performance of standard TCP mechanisms. The maximum segment size was set to 536 bytes and receiver window size was set to 20 segments, the same as the values used by [KAP02]. The results obtained were consistent with the results obtained by that study.

The pilot studies confirmed that TCP suffers from congestion collapse at very high error rates; it would have been counterproductive to perform experimental evaluation of error rates higher than 10^{-5}. The purpose of the experiments was to determine the impact of the CETEN-R algorithm on TCP goodput; improvements in goodput were expected at high error rates and the study confirmed this result.

Side by Side Comparison of One Client Experiments

Throughput is compared for each variant of TCP (Reno and New Reno) at two delay measurements (50 ms and 320 ms). This is the one of the two most important comparisons because when the results are paired, significant differences in mean throughput are observed in several cases. TCP Reno results are contained in Table 4 and displayed graphically in Figure 3. TCP New Reno results are in Table 5 and graphically displayed in Figure 4.

Table 4. One Client TCP Reno Mean Throughput Results

Delay ms	Error Rate	CETEN-R	Mean bps	Std Dev	Std Err Mean	90% Confidence Interval (bps)
50	10^{-7}	Off	808558	6171	2760	(802674, 814441)
		On	767317	5035	2252	(762517, 772118)
	10^{-6}	Off	384695	8264	3696	(376816, 392574)
		On	398321	2681	1199	(395765, 400878)
	10^{-5}	Off	45059	1156	517	(43957, 46162)
		On	79848	877	392	(79012, 80684)
320	10^{-7}	Off	138178	1985	888	(136285, 140071)
		On	132471	2585	1156	(130006, 134936)
	10^{-6}	Off	72374	2642	1181	(69856, 74893)
		On	71118	845	378	(70313, 71923)
	10^{-5}	Off	15896	330	147	(15545, 16174)
		On	21274	428	191	(20866, 21682)

For TCP Reno, at 320 ms delay and error rate of 10^{-5}, CETEN-R mean goodput is 21274 bps compared to 15896 bps, a 33.8% performance increase. At the same delay and an error rate of 10^{-6}, the goodput confidence intervals overlap and the mean of the CETEN-R throughput falls within the confidence interval of the TCP Reno throughput. In this instance, the algorithms are not statistically different. For the 320 ms, 10^{-7} case, Reno TCP goodput is 4% higher than CETEN-R.

For the 50 ms experiments, CETEN-R performed better except at a low error rate, where TCP Reno performed better than CETEN-R. For an error rate of 10^{-5}, CETEN-R mean goodput is 79848 bps compared to 45059 bps, a 77% increase. For an error rate of 10^{-6}, CETEN-R mean goodput is 398321 bps compared to 384695 bps, a modest increase of 3.5%. For an error rate of 10^{-7}, TCP Reno mean goodput is 808558 bps compared to 767317 bps for CETEN-R. In this case the goodput for CETEN-R is 5% less than the goodput for TCP Reno.

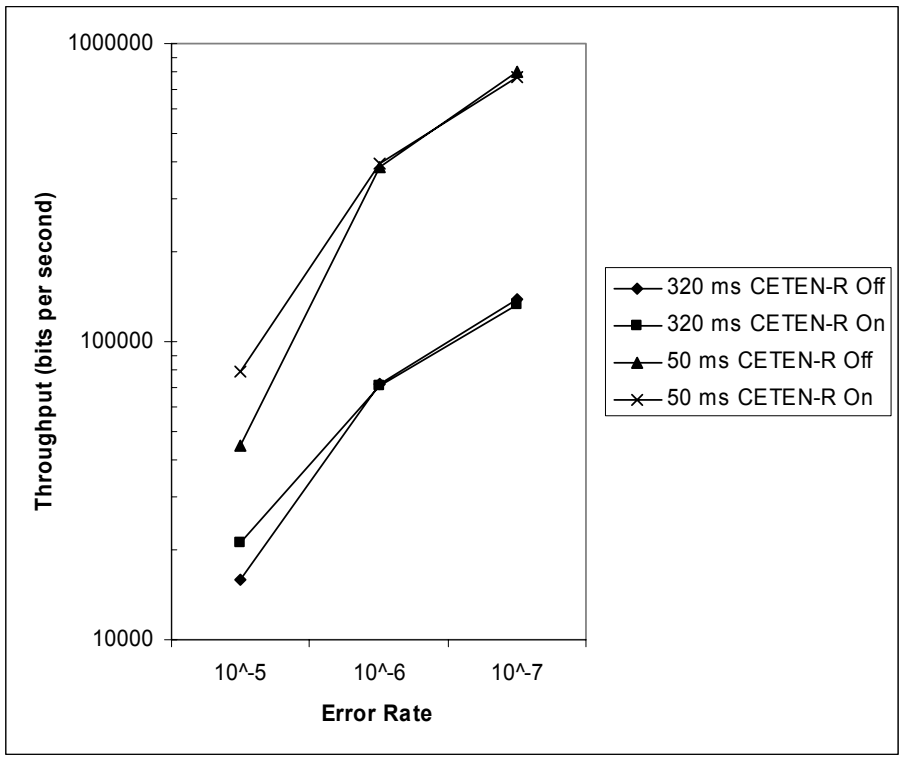

Figure 9. TCP Reno One Client Throughput

CETEN-R's positive impact on TCP New Reno was more significant than TCP Reno. For the 320 ms delay experiments, at an error rate of 10^{-5}, CETEN-R goodput is 28465 bps compared to 16036 bps, a 77.5% increase. At an error rate of 10^{-6}, CETEN-R goodput is 85709 bps compared to 72093 bps, an increase of 18.9%. For an error rate of 10^{-7}, a *t*-test was performed. The results were paired and sample mean was calculated as 2277.68 bits/second, sample variance was calculated as 9162866.46, and sample standard deviation was 3072.02 (cf. Table 6). The 0.95-quartile *t*-variate with 4 degrees of freedom is 2.132. The 90% confidence interval for the mean is calculated as

$$2277.68 +/- 2.132(1353.72) = (-607.45, 5164.81)$$

Since the confidence interval includes zero, for the 320ms, 10^{-7} case, CETEN-R and TCP New Reno do not perform differently.

Table 5. One Client TCP New Reno Mean Throughput

Delay (ms)	Error Rate	CETEN-R	Mean (bps)	Std Dev	Std Err Mean	90% Confidence Interval (bps)
50	10^{-7}	Off	777994	2245	1004	(775854, 780135)
		On	795913	6753	3020	(789475, 802352)
	10^{-6}	Off	387657	4528	2025	(383339, 391974)
		On	486847	3617	1618	(483398, 490295)
	10^{-5}	Off	45304	1048	469	(44305, 46303)
		On	111455	2908	1300	(108683, 114227)
320	10^{-7}	Off	134248	1772	792	(132559, 135937)
		On	136526	2257	1009	(134374, 138678)
	10^{-6}	Off	72093	1925	861	(70258, 73928)
		On	85709	1698	760	(84090, 87329)
	10^{-5}	Off	16036	743	332	(15328, 16744)
		On	28465	728	326	(27771, 29159)

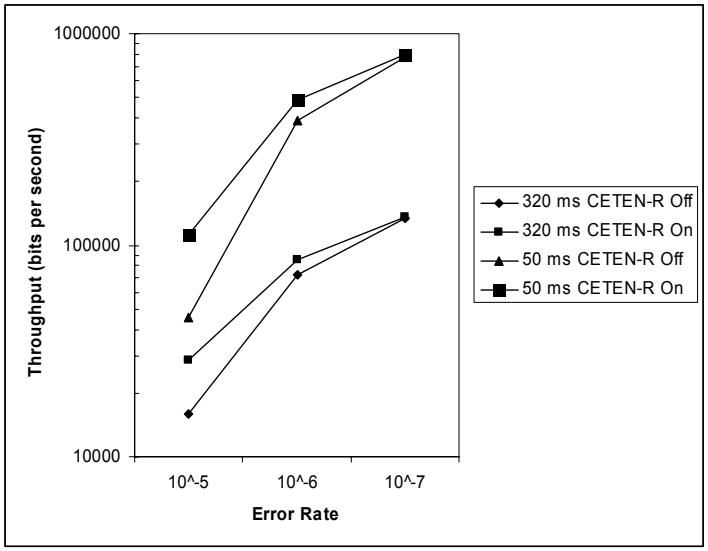

Figure 10. TCP New Reno One Client Throughput

For the 50 ms delay experiments, CETEN-R performs better than TCP Reno at all error rates. At an error rate of 10^{-5}, CETEN-R goodput is 111455 bps compared to 45304 bps, a 146% increase. At an error rate of 10^{-6}, CETEN-R goodput is 486847 bps compared to 387657, an increase of 25.6%. At an error rate of 10^{-7}, CETEN-R goodput is 795913 bps compared to 777994 bps, a 2.3% increase.

Table 6. *t*-Test Calculation

Replication	On	Off	Difference
1	137847.00	132562.30	5284.70
2	135952.40	132253.60	3698.80
3	136665.10	134581.60	2083.50
4	139072.80	136032.10	3040.70
5	133091.10	135810.40	-2719.30
Sum			11388.40
Mean			2277.68
Sample Variance			9162866.46
Standard Deviation			3027.02

ANOVA for One Client Experiments

Table 7 shows the Analysis of Variance (ANOVA) for the one client experiments. As the table shows, the main factors account for 79.24% of variation, with delay and error rate accounting for virtually all (79.082%) of that variation. First-order interactions accounted for another 20.596% of variation, with the interaction between delay and error rate accounting for 20.267% of that total. Main effects and first-order interactions account for 99.836% of total variation, with second-order interactions, third-order interactions and experimental error accounting for the remaining 0.164% of total

variation. Since the probability associated with the F-ratio is <0.0001 for all effects and interactions, the model is considered to be a better fit for the data statistically than the response mean alone.

As expected, when all of the one client experiments are analyzed as a whole, delay and error rate account for the greatest amount of variation in mean goodput. TCP goodput is primarily a function of round trip time and loss rate [PFT98]. Round trip time includes the propagation delay of all links in the network path, processing time at all nodes, transmission time and queuing delay. As round trip time increases, goodput will naturally decrease. In the network model, propagation delay and processing time for the ground stations and satellite itself are accounted for in the delay value selected. Loss rate includes losses from both congestion and corruption. At higher error rates, total losses are higher, resulting in lower goodput. The interaction between delay and error rate also accounted for a substantial portion of total variation. This is expected; for example, high delay but low error rate has better goodput than low delay high error rate. The combination of high delay and high error rates results in poor performance, while the combination of low delay and low error rates results in the best performance.

Table 8 shows the main effects and first-order interaction between delay and error. Since these account for over 99.5% of total variation, all other interactions are statistically insignificant. For all significant effects and interactions, the probability that the absolute value of the *t*-ratio is greater than the computed *t*-value is less than 0.0001. This indicates the effect or interaction is not zero. The values in Table 8 are the expected amounts in bits per second each factor/level causes goodput to vary from the mean. For

example, let's examine the case of TCP New Reno, CETEN-R Off, error rate 10^{-7} and delay of 320 ms. Using the values in Table 8, we can calculate an estimate of goodput as follows:

$$25055 + 5966 - 173526 - 9050 + 210846 - 152519 = 132272$$

This calculated estimate is very close to the observed mean goodput of 134248 bps; only 1.5% of the observed goodput is unaccounted for, which is acceptable amount of error for the purposes of estimating the goodput.

When CETEN-R is considered along with all other factors and interactions, it appears to not have a significant impact on TCP performance. This is true, when the experimental design as a whole is considered, since delay and error rate have the greatest impact on TCP performance. However, when you compare experiments side by side, controlling for all factors except CETEN-R, it is apparent that in some cases CETEN-R can provide moderate to significant improvements in throughput.

Side by Side Comparison of Four Client Experiments

As with the one client experiments, average per client throughput is compared for each variant of TCP (Reno and New Reno) at two delay measurements (50 ms and 320 ms). As with the one client side by side comparison, this comparison is important because when the results are paired, significant differences in mean goodput are observed in several cases. These results only compare the experiments where all the clients had CETEN-R enabled or all the clients were using a standard version of TCP. TCP Reno results are contained in Table 9 and displayed graphically in Figure 5. TCP New Reno results are in Table 10 and displayed graphically in Figure 6.

Table 7. ANOVA for One Client Experiments

Component	Sum of Squares	Percentage of Variation	Degrees of Freedom	F Ratio	Prob > F
y	16481261625731		120		
ybar	7533339392812		1		
y - ybar	8947922232919	100.000	119		
Main Effects	7090312191271	79.240	5		
TCP Flavor (F)	4270497987	0.048	1	380.542	<0.0001
Delay (D)	3613339349705	40.382	1	321983.300	<0.0001
Error Rate (E)	3462873228420	38.700	2	154287.661	<0.0001
CETEN-R (A)	9829115159	0.110	1	875.869	<0.0001
First-Order Interactions	1842871971197	20.596	9		
FD	2065421299	0.023	1	184.049	<0.0001
FE	3688542224	0.041	2	164.342	<0.0001
FA	8837107186	0.099	1	787.471	<0.0001
DE	1813470880115	20.267	2	80798.851	<0.0001
DA	5580255068	0.062	1	497.254	<0.0001
EA	9229765305	0.103	2	411.230	<0.0001
Second-Order Interactions	12984015724	0.145	7		
FDE	2034630841	0.023	2	90.653	<0.0001
FDA	4453810033	0.050	1	396.877	<0.0001
FEA	1205669494	0.013	2	53.718	<0.0001
DEA	5289905355	0.059	2	235.691	<0.0001
Third-Order Interactions	676729980	0.008	2		
FDEA	676729980	0.008	2	30.152	<0.0001
Errors	1077324747	0.012	96		

Table 8. One Client Main Effects and Significant Interactions

Term	Estimate	Std Error	t Ratio	Prob>|t|
TCP Flavor[New Reno]	5966	305.81	19.51	<0.0001
TCP Flavor[Reno]	-5966	305.81	-19.51	<0.0001
Delay[320 ms]	-173526	305.81	-567.44	<0.0001
Delay[50 ms]	173526	305.81	567.44	<0.0001
CETEN-R[Off]	-9050	305.81	-29.60	<0.0001
CETEN-R[On]	9050	305.81	29.60	<0.0001
Error Rate[10^{-5}]	-205142	432.48	-474.34	<0.0001
Error Rate[10^{-6}]	-5703	432.48	-13.19	<0.0001
Error Rate[10^{-7}]	210846	432.48	487.53	<0.0001
Delay[320 ms]*Error Rate[10^{-5}]	148522	432.48	343.42	<0.0001
Delay[320 ms]*Error Rate[10^{-6}]	3998	432.48	9.24	<0.0001
Delay[320 ms]*Error Rate[10^{-7}]	-152519	432.48	-352.67	<0.0001
Delay[50 ms]*Error Rate[10^{-5}]	-148522	432.48	-343.42	<0.0001
Delay[50 ms]*Error Rate[10^{-6}]	-3998	432.48	-9.24	<0.0001
Delay[50 ms]*Error Rate[10^{-7}]	152519	432.48	352.67	<0.0001

For TCP Reno, at 320 ms delay and an error rate of 10^{-5}, CETEN-R mean goodput is 21344 bps compared to 15695 bps, a 36% increase. At the same delay and an error rate of 10^{-6}, TCP Reno mean goodput is 73392 bps compared to 71470 bps for CETEN-R; in this case TCP Reno mean goodput is 2.7% higher than CETEN-R. In the 320 ms, 10^{-7} case, TCP Reno mean goodput is 138410 bps compared to 132052 bps for CETEN-R; in this case TCP Reno goodput is 4.8% higher than CETEN-R.

For the 50 ms delay experiments, CETEN-R performed better except at an error rate of 10^{-7}. At an error rate of 10^{-5}, CETEN-R mean goodput is 78580 bps compared to 44034 bps, a 78% increase. At an error rate of 10^{-6}, CETEN-R mean goodput is 335767 bps compared to 328435 bps, an increase of 2.2%. At an error rate of 10^{-7}, the confidence intervals overlap and both means are within the other confidence interval;

therefore, in this instance CETEN-R and TCP Reno are not different. These results are comparable to the results seen for the one client group of experiments.

Table 9. Four Client TCP Reno Mean Throughput

Delay ms	Error Rate	CETEN-R	Mean bps	Std Dev	Std Err Mean	90% Confidence Interval (bps)
50	10^{-7}	Off	359536	554	248	(359007, 360064)
		On	359507	541	242	(358990, 360023)
	10^{-6}	Off	328435	1281	573	(327213, 329656)
		On	335767	918	411	(334892, 336643)
	10^{-5}	Off	44034	895	400	(43181, 44888)
		On	78580	588	263	(78019, 79140)
320	10^{-7}	Off	138410	729	326	(137505, 139315)
		On	132052	856	383	(130990, 133115)
	10^{-6}	Off	73392	865	387	(72567, 74216)
		On	71470	601	269	(70897, 72043)
	10^{-5}	Off	15695	436	195	(15279, 16111)
		On	21344	91	41	(21257, 21429)

As with the one client experiments, CETEN-R's impact on performance was more significant for TCP New Reno than TCP Reno. For the 320 ms delay experiments, at an error rate of 10^{-5}, CETEN-R mean goodput is 28226 bps compared to 16186 bps, a 74% increase. At an error rate of 10^{-6}, CETEN-R mean goodput is 86291 bps compared to 71890 bps, an increase of 20%. At an error rate of 10^{-7}, the goodput confidence intervals overlap and the CETEN-R goodput mean falls within the confidence interval of TCP New Reno mean goodput; there is no statistical difference in this case.

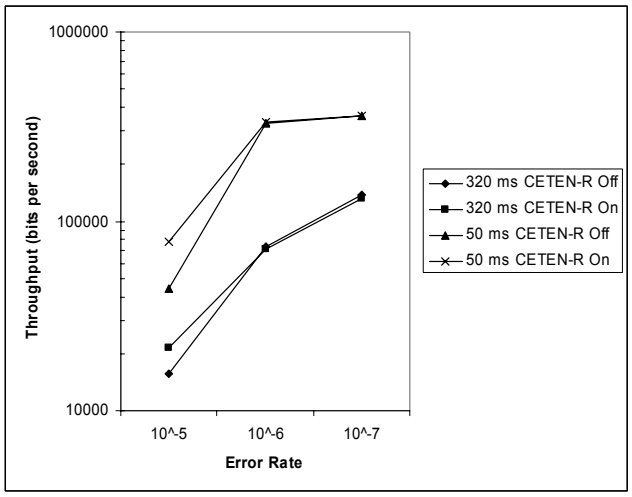

Figure 11. TCP Reno Four Client Throughput

For 50 ms delay experiments, CETEN-R again outperformed TCP New Reno at all error rates except 10^{-7}, in which case the goodput was not statistically different. At an error rate of 10^{-5}, CETEN-R mean goodput was 108477 bps compared to 45352 bps, a 139% increase. At an error rate of 10^{-6}, CETEN-R mean goodput was 348976 bps compared to 328936 bps, an increase of 6.1%. At an error rate of 10^{-7}, the goodput confidence intervals overlap and the both goodput means fall within the other confidence interval; there is no statistical difference in this case. Again, these results are comparable to the results observed for the one client experiments.

Table 10. Four Client TCP New Reno Mean Throughput

Delay ms	Error Rate	CETEN-R	Mean bps	Std Dev	Std Err Mean	90% Confidence Interval (bps)
50	10^{-7}	Off	359520	556	249	(358990, 360050)
		On	359619	642	287	(359006, 360231)
	10^{-6}	Off	328936	1052	471	(327933, 329939)
		On	348976	700	313	(348308, 349643)
	10^{-5}	Off	45352	479	214	(44896, 45809)
		On	108477	1883	842	(106682, 110272)
320	10^{-7}	Off	134677	1028	460	(133697, 135656)
		On	135753	1547	692	(134278, 137228)
	10^{-6}	Off	71890	937	419	(70997, 72783)
		On	86291	1025	458	(85314, 87268)
	10^{-5}	Off	16186	296	133	(15904, 16469)
		On	28226	374	167	(27870, 28582)

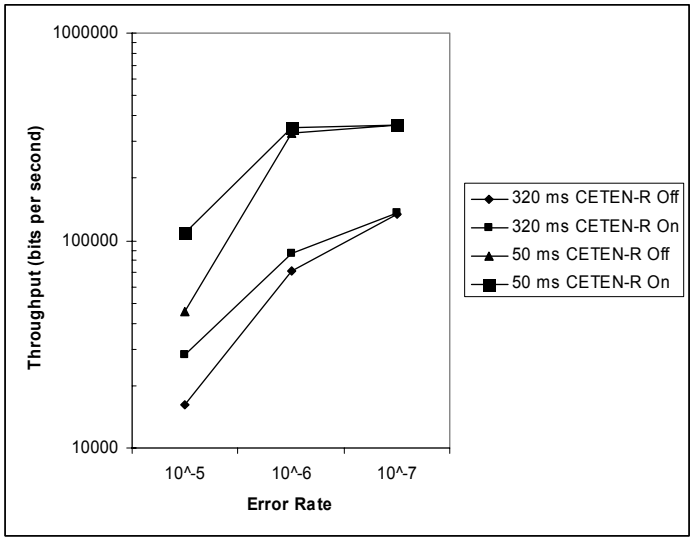

Figure 12. TCP New Reno Four Client Throughput

ANOVA for Four Client Experiments.

Table 11 shows the ANOVA for the four client experiments where all clients either had CETEN-R enabled or all clients were using Stock TCP. The results are similar to the results for the one client experiments. The main factors account for 87.504% of variation, which is even more than the one client experiments, with delay and error rate accounting for virtually all (87.242%) of that variation. First-order interactions accounted for another 12.357% of variation, with the interaction between delay and error rate accounting for 11.936% of that total. Main effects and first-order interactions account for 99.861% of total variation, with second-order interactions, third-order interactions and experimental error accounting for the remaining 0.139% of total variation. Since the probability associated with the F-ratio is <0.0001 for all effects and interactions, the model is considered to be a better fit for the data statistically than the response mean alone.

As previously discussed, delay and error rate account for most of the variation in goodput, since goodput is primarily a function of round trip time and loss rate. In the four client experiments, the additional clients result in a reduction in mean goodput per client as delay and error rate decrease, causing the interaction between delay and error rate to have a less significant effect on mean goodput than for the one client experiments, although the interaction still accounts for 11.963% of total variation.

Table 12 shows the main effects and first-order interaction between delay and error. Since these account for over 99.4 of total variation, all other interactions are statistically insignificant. For all significant effects and interactions, the probability that

the absolute value of the *t*-ratio is greater than the computed *t*-value is less than 0.0001. This indicates the effect or interaction is not zero. As discussed in the ANOVA for the one client experiments, the values in Table 12 are the expected amounts in bits per second each factor/level causes goodput to vary from the mean.

Comparing One Client and Four Client Experiments

To determine the impact of adding clients to a link, the mean goodput of a typical client in the four client experiments is compared to the mean goodput for a client in the one client experiments. Results are compared when all other factors are held constant. Table 13 and Figures 7 and 8 summarize the results for TCP New Reno. Table 14 and Figures 9 and 10 summarize the results for TCP Reno.

For TCP New Reno, for the 320 ms experiments, average goodput per client in the four client experiments is statistically the same as average client throughput in the one client experiments at error rates of 10^{-5}, 10^{-6} and 10^{-7}. The corresponding goodput confidence intervals overlap, and the goodput means are contained in the opposite confidence interval indicating there is no statistical difference.

For the 50 ms experiments, however, at error rates of 10^{-6} and 10^{-7}, per client average throughput for the four client experiments is less than average throughput for a single client. An examination of the raw data obtained from the simulations reveals that for an error rate of 10^{-7}, in the one client experiments average segment delay was 56 ms compared to 125 ms for the four client experiments; four client average segment delay is 123% higher. This additional delay helps explain the reduced per client goodput seen in the four client experiments. Since round trip time is one of the major factors influencing

Table 11. Four Client ANOVA

Component	Sum of Squares	Percentage of Variation	Degrees of Freedom	F Ratio	Prob > F
y	5438334657038		120		
ybar	3303607604992		1		
y - ybar	2134727052047	100.000	119		
Main Effects	1867966277030	87.504	5		
TCP Flavor (F)	898773333	0.042	1	1166.060	<0.0001
Delay (D)	946388369944	44.333	1	1227835.243	<0.0001
Error Rate (E)	915991751361	42.909	2	594199.480	<0.0001
CETEN-R (A)	4687382392	0.220	1	6081.365	<0.0001
First-Order Interactions	263784067649	12.357	9		
FD	123623034	0.006	1	160.387	<0.0001
FE	488532236	0.023	2	316.909	<0.0001
FA	1066895064	0.050	1	1384.179	<0.0001
DE	255372887082	11.963	2	165659.174	<0.0001
DA	2092736322	0.098	1	2715.096	<0.0001
EA	4639393911	0.217	2	3009.553	<0.0001
Second-Order Interactions	2579920735	0.121	7		
FDE	231703431	0.011	2	150.305	<0.0001
FDA	26437914	0.001	1	34.300	<0.0001
FEA	259878947	0.012	2	168.582	<0.0001
DEA	2061900443	0.097	2	1337.545	<0.0001
Third-Order Interactions	322791947	0.015	2		
FDEA	322791947	0.015	2	209.394	<0.0001
Errors	73994686	0.003	96		

Table 12. Four Client Main Effects and Significant Interactions

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	165922	80.14	2070.28	<0.0001
TCP Flavor[New Reno]	2737	80.14	34.15	<0.0001
TCP Flavor[Reno]	-2737	80.14	-34.15	<0.0001
Delay[320 ms]	-88806	80.14	-1108.08	<0.0001
Delay[50 ms]	88806	80.14	1108.08	<0.0001
CETEN-R[Off]	-6250	80.14	-77.98	<0.0001
CETEN-R[On]	6250	80.14	77.98	<0.0001
Error Rate[10^{-5}]	-121185	113.34	-1069.20	<0.0001
Error Rate[10^{-6}]	39723	113.34	350.47	<0.0001
Error Rate[10^{-7}]	81462	113.34	718.73	<0.0001
Delay[320 ms]*Error Rate[10^{-5}]	64432	113.34	568.48	<0.0001
Delay[320 ms]*Error Rate[10^{-6}]	-41077	113.34	-362.42	<0.0001
Delay[320 ms]*Error Rate[10^{-7}]	-23355	113.34	-206.06	<0.0001
Delay[50 ms]*Error Rate[10^{-5}]	-64432	113.34	-568.48	<0.0001
Delay[50 ms]*Error Rate[10^{-6}]	41077	113.34	362.42	<0.0001
Delay[50 ms]*Error Rate[10^{-7}]	23355	113.34	206.06	<0.0001

goodput, this increase in segment delay corresponds to an increase in round trip time in the four client experiments, causing a reduction in goodput. Total goodput for the four client low delay, low error rate experiments, however, was higher than for the one client experiments and at 1.4 Mb/s was very close to the router's datagram forwarding rate of 1.544 Mb/s.

For TCP Reno, the comparisons reveal some ambiguous results. For the 320 ms delay and an error rate of 10^{-5} experiments where CETEN-R is enabled, each client in the four client experiments had a mean goodput of 28226 bps compared to a mean goodput of 21274 bps for the one client experiments, a 32.7% increase. The reasons for this anomaly are not apparent and further investigation is needed to determine the cause. For

Table 13. One Client/Four Client New Reno Comparison

Num Clients	Delay ms	Error Rate	CETEN-R	Mean bps	Std Dev	Std Err Mean	90% Confidence Interval (bps)
1	50	10^{-7}	Off	777994	2245	1004	(775854, 780135)
4	50	10^{-7}	Off	359520	556	249	(358990, 360050)
1	50	10^{-7}	On	795913	6753	3020	(789475, 802352)
4	50	10^{-7}	On	359619	642	287	(359006, 360231)
1	50	10^{-6}	Off	387657	4528	2025	(383339, 391974)
4	50	10^{-6}	Off	328936	1052	471	(327933, 329939)
1	50	10^{-6}	On	486847	3617	1618	(483398, 490295)
4	50	10^{-6}	On	348976	700	313	(348308, 349643)
1	50	10^{-5}	Off	45304	1048	469	(44305, 46303)
4	50	10^{-5}	Off	45352	479	214	(44896, 45809)
1	50	10^{-5}	On	111455	2908	1300	(108683, 114227)
4	50	10^{-5}	On	108477	1883	842	(106682, 110272)
1	320	10^{-7}	Off	134248	1772	792	(132559, 135937)
4	320	10^{-7}	Off	134677	1028	460	(133697, 135656)
1	320	10^{-7}	On	136526	2257	1009	(134374, 138678)
4	320	10^{-7}	On	135753	1547	692	(134278, 137228)
1	320	10^{-6}	Off	72093	1925	861	(70258, 73928)
4	320	10^{-6}	Off	71890	937	419	(70997, 72783)
1	320	10^{-6}	On	85709	1698	760	(84090, 87329)
4	320	10^{-6}	On	86291	1025	458	(85314, 87268)
1	320	10^{-5}	Off	16036	743	332	(15328, 16744)
4	320	10^{-5}	Off	16186	296	133	(15904, 16469)
1	320	10^{-5}	On	28465	728	326	(27771, 29159)
4	320	10^{-5}	On	28226	374	167	(27870, 28582)

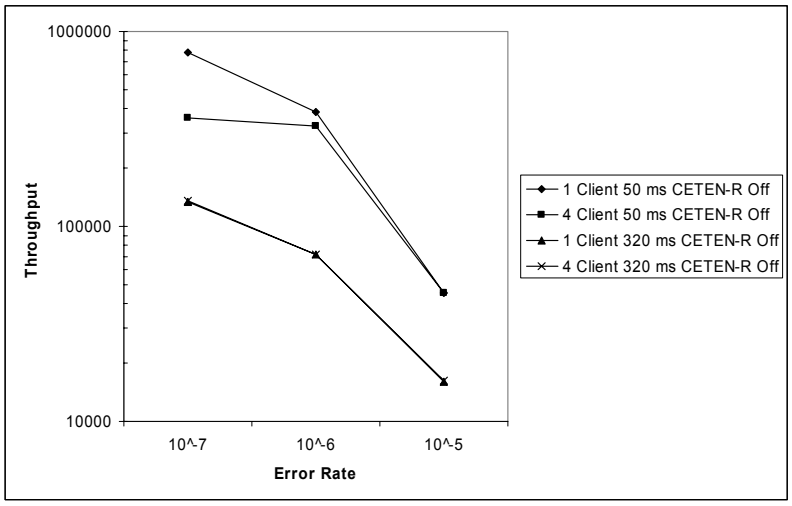

Figure 13. TCP New Reno One Client/Four Client CETEN-Off Throughput Comparison

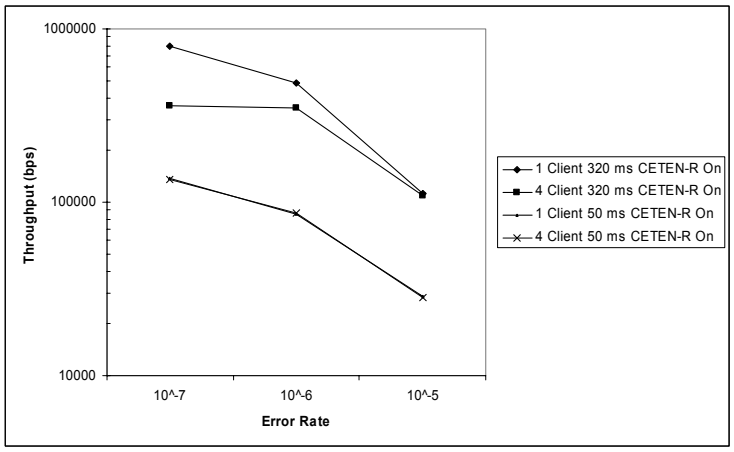

Figure 14. TCP New Reno One Client/Four Client CETEN-On Throughput Comparison

error rates of 10^{-6} and 10^{-7} and a delay of 320 ms, experiments reveal per client mean goodput is the statistically the same for both the four client and one client experiments. The corresponding goodput confidence intervals overlap, and the goodput means are contained in the opposite confidence interval indicating there is no statistical difference.

For the 50 ms delay, error rate of 10^{-5} experiments where CETEN-R is enabled, for the one client experiments, mean goodput is 79848 bps compared to 78580 bps for the four client experiments, which is 1.6% greater. No apparent cause could be discovered for this difference. For error rates of 10^{-6} and 10^{-7} and a delay of 50 ms, average throughput for the four client experiments is less than average throughput for a single client experiment. Total goodput is higher and was very close to the router datagram forwarding rate of 1.544 Mb/s. This result is the same as for TCP New Reno.

ANOVA for both One and Four Client Experiments

Next, an ANOVA was done over the entire one client and four client experiments where all clients either had CETEN-R enabled or all clients were using a standard version of TCP. The results are summarized in Table 15. When experimental design is considered as a whole, main effects account for 73.306% of total variation, with delay accounting for 35.866% and error rate accounting for another 33.545%. An additional 3.733% of variation is accounted for by the number of clients. This accounts for 73.144% of total variation. The remaining two main effects account for only 0.162% of total variation. First order interactions account for another 22.007% of total variation, with the bulk of that variation accounted for by three interactions: number of clients*delay (3.741%), number of clients*error rate (4.491%) and delay*error rate (13.491%). Second order interactions account for 4.616% of total variation, with number of clients*delay*error rate accounting for most of that variation (4.480%). All higher order iterations and errors explain only 0.074% of total variation and are considered insignificant. Since the probability associated with the F-ratio is <0.0001 for all

significant effects and interactions, the model is considered to be a better fit for the data statistically than the response mean alone.

Table 14. One Client/Four Client TCP Reno Comparison

Num Clients	Delay ms	Error Rate	CETEN-R	Mean bps	Std Dev	Std Err Mean	90% Confidence Interval (bps)
1	50	10^{-7}	Off	808558	6171	2760	(802674, 814441)
4	50	10^{-7}	Off	359536	554	248	(359007, 360064)
1	50	10^{-7}	On	767317	5035	2252	(762517, 772118)
4	50	10^{-7}	On	359507	541	242	(358990, 360023)
1	50	10^{-6}	Off	384695	8264	3696	(376816, 392574)
4	50	10^{-6}	Off	328435	1281	573	(327213, 329656)
1	50	10^{-6}	On	398321	2681	1199	(395765, 400878)
4	50	10^{-6}	On	335767	918	411	(334892, 336643)
1	50	10^{-5}	Off	45059	1156	517	(43957, 46162)
4	50	10^{-5}	Off	44034	895	400	(43181, 44888)
1	50	10^{-5}	On	79848	877	392	(79012, 80684)
4	50	10^{-5}	On	78580	588	263	(78019, 79140)
1	320	10^{-7}	Off	138178	1985	888	(136285, 140071)
4	320	10^{-7}	Off	138410	729	326	(137505, 139315)
1	320	10^{-7}	On	132471	2585	1156	(130006, 134936)
4	320	10^{-7}	On	132052	856	383	(130990, 133115)
1	320	10^{-6}	Off	72374	2642	1181	(69856, 74893)
4	320	10^{-6}	Off	73392	865	387	(72567, 74216)
1	320	10^{-6}	On	71118	845	378	(70313, 71923)
4	320	10^{-6}	On	71470	601	269	(70897, 72043)
1	320	10^{-5}	Off	15896	330	147	(15545, 16174)
4	320	10^{-5}	Off	16186	296	133	(15904, 16469)
1	320	10^{-5}	On	21274	428	191	(20866, 21682)
4	320	10^{-5}	On	28226	374	167	(27870, 28582)

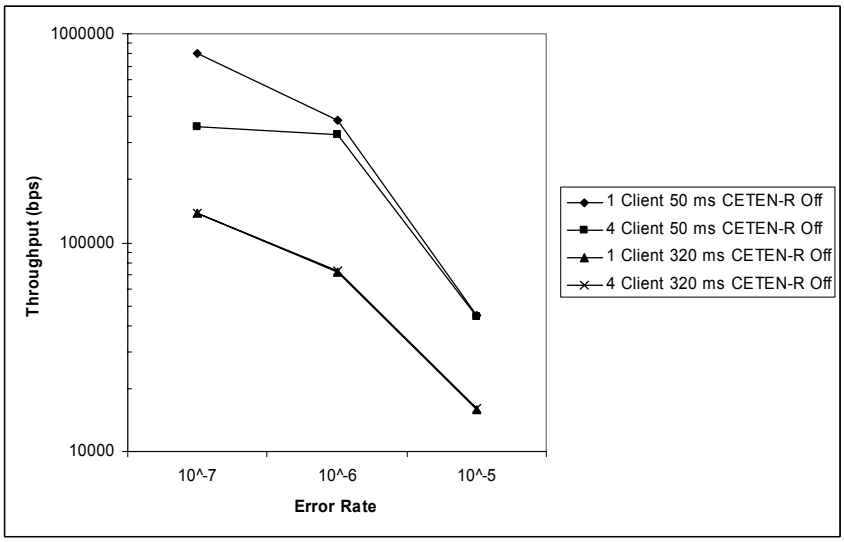

Figure 15. TCP Reno One Client/Four Client CETEN-R Off Throughput Comparison

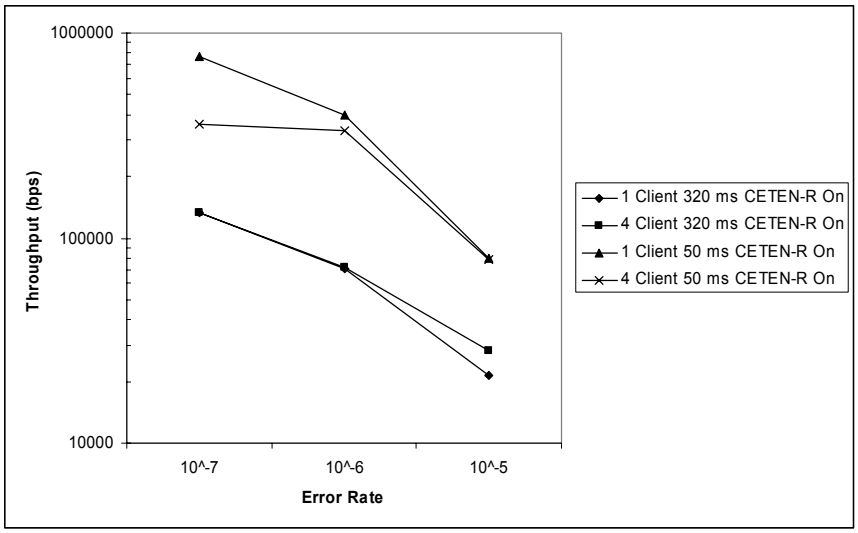

Figure 16. TCP Reno One Client/Four Client CETEN-R Enabled Throughput Comparison

As shown in the one client and four client ANOVAs, error rate and delay account for most of the variation in mean goodput because goodput is largely a function of round

trip time and loss rate [PFT98]. The number of clients impacts goodput because as the number of clients increases, the router queues can lose the ability to process all the packets as they arrive, leading to an increase in congestion, causing greater loss due to congestion, reducing goodput [Jac88, Ste94]. The first order interactions between the number of clients and delay and the number of clients and error rate and the second order interaction between number of clients, delay and error are significant. Looking back at Tables 13 and 14 provides clues as to why these interactions are significant. When delay is high, the mean goodput for comparable one client and four client experiments was statistically the same. When delay is low, as the number of clients increases, per client goodput decreases if error rates are moderate or low. This suggests that there is a significant interaction between the number of clients, delay and error rate and is expected.

Table 16 shows the significant main effects and interactions. Since these account for over 99.3% of total variation all other interactions and errors are not considered here. A *t*-test was performed on each significant effect or interaction. In all cases, the probability that the absolute value of the *t*-ratio is greater than the computed *t*-value is less than 0.0001 indicating the effect or interaction is not zero. As previously discussed in the ANOVA for the one client experiments, the values in Table 16 are the expected amounts in bits per second each factor/level causes goodput to vary from the mean.

Table 15. Overall ANOVA

Component	Sum of Squares	Percentage of Variation	DOF	F Ratio	Prob > F
y	21919596282770		240		
ybar	10407180476575		1		
y - ybar	11512415806195	100.000	239		
Main Effects	8439266670600	73.306	6		
# Clients (N)	429767000000	3.733	1	71670.094	<0.0001
TCP Flavor (F)	4543770600	0.039	1	757.743	<0.0001
Delay (D)	4129090000000	35.866	1	688587.788	<0.0001
Error Rate (E)	3861820000000	33.545	2	322008.243	<0.0001
CETEN-R (A)	14045900000	0.122	1	2342.375	<0.0001
First-Order Interactions	2533561696928	22.007	14		
NF	625500721	0.005	1	104.312	<0.0001
ND	430642000000	3.741	1	71816.055	<0.0001
NE	517049000000	4.491	2	43112.875	<0.0001
NA	470552456	0.004	1	78.472	<0.0001
FD	1599827665	0.014	1	266.796	<0.0001
FE	2862461857	0.025	2	238.679	<0.0001
FA	8022549290	0.070	1	1337.882	<0.0001
DE	1553120000000	13.491	2	129503.080	<0.0001
DA	7253804939	0.063	1	1209.682	<0.0001
EA	11916000000	0.104	2	993.588	<0.0001
Second-Order Interactions	531398783900	4.616	16		
NDE	515725000000	4.480	2	43002.478	<0.0001
All others	15673783900	0.136	14		
Third-Order Interactions	6148557912	0.053	9		
Fourth-Order Interactions	1255005198	0.011	2	74.939	<0.0001
Errors	1151319433	0.010	192		

Table 16. All Experiments Significant Effects and Interactions

Term	Estimate	StdErr	t Ratio	Prob>|t|
# Clients[1]	42317	158	267.71	<0.0001
# Clients[4]	-42317	158	-267.71	<0.0001
Delay[320 ms]	-131166	158	-829.81	<0.0001
Delay[50 ms]	131166	158	829.81	<0.0001
Error Rate[10^{-5}]	-163164	224	-729.91	<0.0001
Error Rate[10^{-6}]	17010	224	76.09	<0.0001
Error Rate[10^{-7}]	146154	224	653.81	<0.0001
# Clients[1]*Delay[320 ms]	-42360	158	-267.99	<0.0001
# Clients[1]*Delay[50 ms]	42360	158	267.99	<0.0001
# Clients[4]*Delay[320 ms]	42360	158	267.99	<0.0001
# Clients[4]*Delay[50 ms]	-42360	158	-267.99	<0.0001
# Clients[1]*Error Rate[10^{-5}]	-41979	224	-187.79	<0.0001
# Clients[1]*Error Rate[10^{-6}]	-22713	224	-101.61	<0.0001
# Clients[1]*Error Rate[10^{-7}]	64692	224	289.40	<0.0001
# Clients[4]*Error Rate[10^{-5}]	41979	224	187.79	<0.0001
# Clients[4]*Error Rate[10^{-6}]	22713	224	101.61	<0.0001
# Clients[4]*Error Rate[10^{-7}]	-64692	224	-289.40	<0.0001
Delay[320 ms]*Error Rate[10^{-5}]	106477	224	476.32	<0.0001
Delay[320 ms]*Error Rate[10^{-6}]	-18540	224	-82.94	<0.0001
Delay[320 ms]*Error Rate[10^{-7}]	-87937	224	-393.38	<0.0001
Delay[50 ms]*Error Rate[10^{-5}]	-106477	224	-476.32	<0.0001
Delay[50 ms]*Error Rate[10^{-6}]	18540	224	82.94	<0.0001
Delay[50 ms]*Error Rate[10^{-7}]	87937	224	393.38	<0.0001
# Clients[1]*Delay[320 ms]*Error Rate[10^{-5}]	42045	224	188.09	<0.0001
# Clients[1]*Delay[320 ms]*Error Rate[10^{-6}]	22538	224	100.82	<0.0001
# Clients[1]*Delay[320 ms]*Error Rate[10^{-7}]	-64582	224	-288.91	<0.0001
# Clients[1]*Delay[50 ms]*Error Rate[10^{-5}]	-42045	224	-188.09	<0.0001
# Clients[1]*Delay[50 ms]*Error Rate[10^{-6}]	-22538	224	-100.82	<0.0001
# Clients[1]*Delay[50 ms]*Error Rate[10^{-7}]	64582	224	288.91	<0.0001
# Clients[4]*Delay[320 ms]*Error Rate[10^{-5}]	-42045	224	-188.09	<0.0001
# Clients[4]*Delay[320 ms]*Error Rate[10^{-6}]	-22538	224	-100.82	<0.0001
# Clients[4]*Delay[320 ms]*Error Rate[10^{-7}]	64582	224	288.91	<0.0001
# Clients[4]*Delay[50 ms]*Error Rate[10^{-5}]	42045	224	188.09	<0.0001
# Clients[4]*Delay[50 ms]*Error Rate[10^{-6}]	22538	224	100.82	<0.0001
# Clients[4]*Delay[50 ms]*Error Rate[10^{-7}]	-64582	224	-288.91	<0.0001

Analysis of Four Client Mixed CETEN-R Experiments

A group of experiments was performed to determine the impact having some TCP flows with CETEN-R enabled sharing a single TCP link with other flows which were using a standard TCP version. Table 17 and Figure 17 compare the goodput for TCP Reno. Table 18 and Figure 18 compare the goodput for TCP New Reno. The data in Tables 17 and 18 are examined to determine CETEN-R enabled TCP flows are too "aggressive" when sharing a single link with standard TCP flows. If the CETEN-R flows are unfair to the normal TCP flows, they will, on average, use more than their fair share of the bandwidth or more bandwidth than an average CETEN-R flow receives when all flows are CETEN-R enabled.

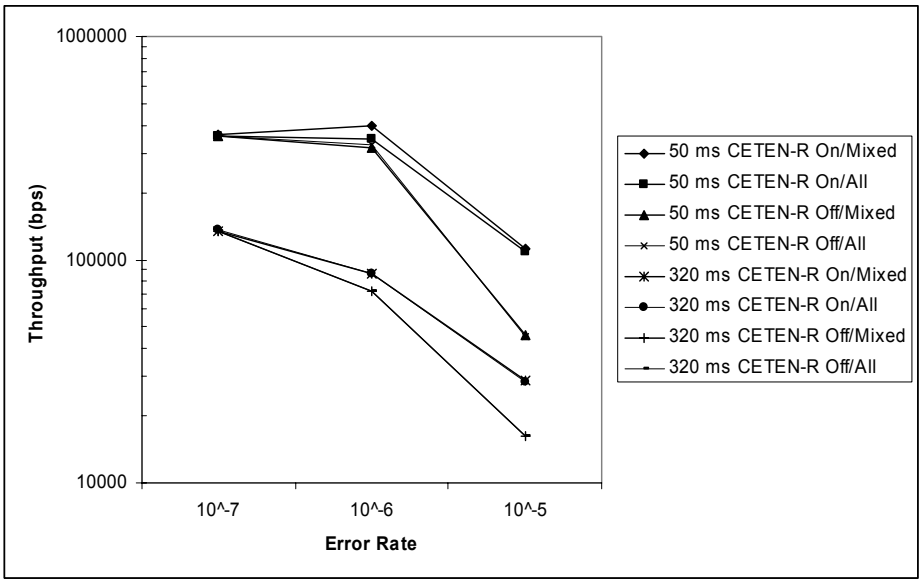

Figure 17. TCP New Reno Throughput Comparisons

Table 17. Mean Throughput Comparison, TCP New Reno, Four Client Mixed and Four Client Homogenous Experiments

Delay ms	Error Rate	CETEN-R	Mean bps	Std Dev	Std Err Mean	90% Confidence Interval (bps)
50	7-Oct	Off/All	359520	556	249	(358990, 360050)
50		Off/Mixed	358394	1416	633	(357043, 359744)
50		On/All	359619	642	287	(359006, 360231)
50		On/Mixed	362872	4004	1791	(359055, 366689)
50	6-Oct	Off/All	328936	1052	471	(327933, 329939)
50		Off/Mixed	316227	2199	983	(314131, 318324)
50		On/All	348976	700	313	(348308, 349643)
50		On/Mixed	399656	5114	2287	(394780, 404532)
50	5-Oct	Off/All	45352	479	214	(44896, 45809)
50		Off/Mixed	45789	568	254	(45248, 46330)
50		On/All	108477	1883	842	(106682, 110272)
50		On/Mixed	112272	2662	1191	(109734, 114811)
320	7-Oct	Off/All	134677	1028	460	(133697, 135656)
320		Off/Mixed	134399	818	366	(133619, 135178)
320		On/All	135753	1547	692	(134278, 137228)
320		On/Mixed	134249	2112	945	(132235, 136263)
320	6-Oct	Off/All	71890	937	419	(70997, 72783)
320		Off/Mixed	72057	859	384	(71238, 72877)
320		On/All	86291	1025	458	(85314, 87268)
320		On/Mixed	87122	2123	949	(85098, 89146)
320	5-Oct	Off/All	16186	296	133	(15904, 16469)
320		Off/Mixed	16148	207	93	(15951, 16346)
320		On/All	28226	374	167	(27870, 28582)
320		On/Mixed	28698	657	294	(28072, 29323)

Table 18. Mean Throughput Comparison, TCP Reno, Four Client Mixed and Four Client Homogeneous Experiments

Delay ms	Error Rate	CETEN-R	Mean bps	Std Dev	Std Err Mean	90% Confidence Interval (bps)
50	10^{-7}	Off/All	359536	554	248	(359007, 360064)
50	10^{-7}	Off/Mixed	363217	1022	457	(362242, 364192)
50	10^{-7}	On/All	359507	541	242	(358990, 360023)
50	10^{-7}	On/Mixed	348480	2813	1258	(345798, 351162)
50	10^{-6}	Off/All	328435	1281	573	(327213, 329656)
50	10^{-6}	Off/Mixed	327789	817	365	(327010, 328567)
50	10^{-6}	On/All	335767	918	411	(334892, 336643)
50	10^{-6}	On/Mixed	337600	6797	3040	(331119, 344081)
50	10^{-5}	Off/All	44034	895	400	(43181, 44888)
50	10^{-5}	Off/Mixed	44389	915	409	(43516, 45261)
50	10^{-5}	On/All	78580	588	263	(78019, 79140)
50	10^{-5}	On/Mixed	78999	1435	642	(77631, 80367)
320	10^{-7}	Off/All	138410	729	326	(137505, 139315)
320	10^{-7}	Off/Mixed	138824	544	243	(138306, 139343)
320	10^{-7}	On/All	132052	856	383	(130990, 133115)
320	10^{-7}	On/Mixed	132614	1573	704	(131114, 134114)
320	10^{-6}	Off/All	73392	865	387	(72567, 74216)
320	10^{-6}	Off/Mixed	71462	1139	509	(70376, 72548)
320	10^{-6}	On/All	71470	601	269	(70897, 72043)
320	10^{-6}	On/Mixed	71158	1223	547	(69992, 72325)
320	10^{-5}	Off/All	15695	436	195	(15279, 16111)
320	10^{-5}	Off/Mixed	15679	313	140	(15381, 15978)
320	10^{-5}	On/All	21344	91	41	(21257, 21429)
320	10^{-5}	On/Mixed	21381	444	199	(20958, 21805)

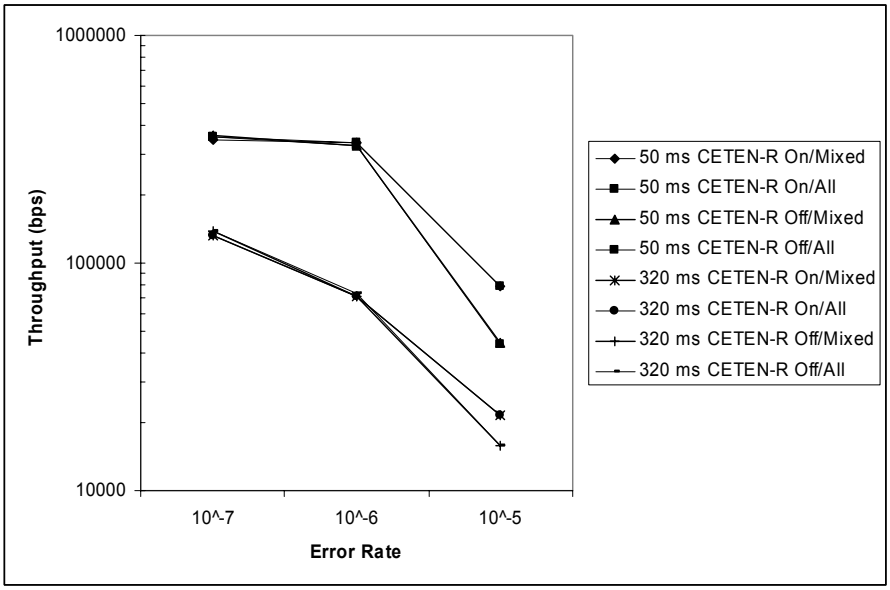

Figure 18. TCP Reno Throughput Comparisons

A line by line comparison of these tables and a visual comparison of the figures reveal that in most cases CETEN-R is not "unfair" TCP Reno or TCP New Reno. Table 19 summarizes the four cases where the mean throughput differs in a mixed CETEN-R/standard TCP version environment as compared to a homogenous environment, where all clients are either using standard TCP or all are CETEN-R enabled.

For TCP New Reno, at a delay of 50 ms and an error rate of 10^{-6}, comparing CETEN-R flows, in the mixed environment case, the flow with CETEN-R enabled had a mean goodput of 399656 bps compared to 348976 bps per client when all are CETEN-R enabled, a 14.5% higher goodput. Conversely, for TCP New Reno at a delay of 50 ms and an error rate of 10^{-6}, comparing standard TCP New Reno flows, when all flows were using TCP New Reno, mean goodput is 328936 bps compared to 316227 bps in a mixed

environment, a 3.8% decrease. In this case, CETEN-R is behaving slightly aggressive and receives more bandwidth that it otherwise would.

Table 19. Comparison of Differences in Mixed Versus Homogeneous Environments

Flavor	Delay (ms)	Error Rate	CETEN-R	Mean Tput (bps)	90% Confidence Interval (bps)
New Reno	50	10^{-6}	On/Mixed	399656	(394780, 404532)
New Reno	50	10^{-6}	On/All	348976	(348308, 349643)
New Reno	50	10^{-6}	Off/Mixed	316227	(314131, 318324)
New Reno	50	10^{-6}	Off/All	328936	(327933, 329939)
New Reno	50	10^{-5}	On/Mixed	112272	(109734, 114811)
New Reno	50	10^{-5}	On/All	108477	(106682, 110272)
New Reno	50	10^{-5}	Off/Mixed	45789	(45248, 46330)
New Reno	50	10^{-5}	Off/All	45352	(44896, 45809)
Reno	50	10^{-7}	On/Mixed	348480	(345798, 351162)
Reno	50	10^{-7}	On/All	359507	(358990, 360023)
Reno	50	10^{-7}	Off/Mixed	363217	(362242, 364192)
Reno	50	10^{-7}	Off/All	359536	(359007, 360064)
Reno	320	10^{-6}	On/Mixed	71158	(69992, 72325)
Reno	320	10^{-6}	On/All	71470	(70897, 72043)
Reno	320	10^{-6}	Off/Mixed	71462	(70376, 72548)
Reno	320	10^{-6}	Off/All	73392	(72597, 74216)

For TCP New Reno, a delay of 50 ms and an error rate of 10^{-5}, comparing CETEN-R flows, in a mixed environment, the CETEN-R mean goodput was 112272 bps compared to 108477 bps when all flows are CETEN-R enabled, a 3.5% increase, but there was no statistical difference between the goodput of the standard TCP New Reno flows in the mixed and homogeneous environments. In this case, the small advantage

CETEN-R appears to have in a mixed environment may be due to the relatively small number of replications of the experiment.

For TCP Reno, with 50 ms delay and an error rate of 10^{-7}, for the CETEN-R flows, in a mixed environment, the CETEN-R goodput is 348480 bps compared to 359507 bps when all flows are CETEN-R enabled, a 3% decrease. For the TCP Reno flows, in a mixed environment, the goodput is 363217 bps compared to 359536 bps when all flows are using TCP Reno, an increase of 1%. These differences are very small and not very significant.

The final case is TCP Reno, an error rate of 10^{-6} and delay of 320 ms. For the CETEN-R enabled flows, the goodput confidence intervals overlap and the mean goodput is contained within the confidence interval so there is no statistical difference. For the TCP Reno flows, in a mixed environment the TCP Reno flow's goodput is 71462 bps compared to 73392 bps when all flows are using TCP Reno, a 2.6% decrease. This difference is small and not very significant.

It appears that for most cases CETEN-R is not unfair to TCP Reno and TCP New Reno in a mixed client system; however, the results are curious and further experimentation is required to determine the exact causes of the anomalies seen.

Summary

In this chapter, the results of this research and analysis of this research have been presented. First, a comparison of goodput between standard versions of TCP and CETEN-R enabled TCP was done with a single client on a link. CETEN-R enabled TCP performed better than TCP Reno and TCP New Reno at higher error rates. As the error

rate decreased, little or no improvement in goodput was observed. Then goodput was compared with four clients sharing the bandwidth. Again improvements were more significant at higher error rates than at lower error rates, where little or no improvement in performance was observed. ANOVAs were done delay and error rate and the interaction between delay and error rate accounted for most of the variation in goodput, as expected. However, when experiments are examined side by side, holding latency, error rate, number of flows and TCP version constant, CETEN-R demonstrates an increase in goodput in some cases, especially at low delay and high error rates; it also showed a significant increase in goodput when delay and error rates are high. Finally, an analysis of experiments in which some clients were CETEN-R enabled and some were using standard TCP versions was conducted. This analysis showed that in general CETEN-R was not "unfair" to standard TCP, with a couple of minor exceptions.

VI. Conclusions and Recommendations

Chapter Overview

This chapter discusses the conclusions of this research, noting areas where CETEN-R provides the greatest improvements in TCP performance over standard versions of TCP. Next, an overview of the significance of this research is discussed. Finally, recommendations for future research are outlined.

Conclusions of Research

CETEN-R is a technique which shows some promise in improving TCP throughput over satellite and wireless links. The results of a previous CETEN study [KAP02] noted gains in goodput over TCP Reno except at high error rates. In contrast, this research has shown CETEN-R to provide significant improvements in goodput over TCP Reno and TCP New Reno at high error rates and both low and high latencies. CETEN-R's most pronounced effect was when combined with TCP New Reno at high error rates. For TCP New Reno combined with CETEN-R, 320 ms delay, goodput increased by 77% at an error rate of 10^{-5}, by 18.9% at an error rate of 10^{-6} and was not statistically different at an error rate of 10^{-7}. For TCP New Reno combined with CETEN-R, 50 ms delay, goodput increased by 146% at an error rate of 10^{-5}, by 15.6% at an error rate of 10^{-6}, and by 2.3% at an error rate of 10^{-7}. For TCP Reno combined with CETEN-R, 320 ms delay, goodput increased by 33.8% at an error rate of 10^{-5}, was not statistically different at an error rate of 10^{-6}, and goodput was reduced by 4% at an error rate of 10^{-7}. For TCP Reno combined with CETEN-R, 50 ms delay, goodput increased by 77% at an

error rate of 10^{-5} and by 3.5% at an error rate of 10^{-6}; goodput was reduced by 5% at an error rate of 10^{-7}.

CETEN-R also performed well in a four flow scenario. Improvements in goodput were similar to those noted for the single flow experiments. For TCP New Reno combined with CETEN-R, 320 ms delay, goodput increased by 74% at an error rate of 10^{-5}, by 20% at an error rate of 10^{-6} and was not statistically different at an error rate of 10^{-7}. For TCP New Reno combined with CETEN-R and 50 ms delay, goodput increased by 139% at an error rate of 10^{-5}, by 6.1% at an error rate of 10^{-6}, and was not statistically different at an error rate of 10^{-7}. For TCP Reno combined with CETEN-R, 320 ms delay, goodput increased by 36% at an error rate of 10^{-5}, was reduced by 2.7% at an error rate of 10^{-6}, and was reduced by 4.8% at an error rate of 10^{-7}. For TCP Reno combined with CETEN-R, 50 ms delay, goodput increased by 78 at an error rate of 10^{-5} and by 2.2% at an error rate of 10^{-6} and was not statistically different at an error rate of 10^{-7}.

Finally, when a single link is shared between CETEN-R flows and TCP flows, CETEN-R does not perform overly aggressively; in only one case was the goodput in the mixed environment significantly different from the goodput in the experiments where all flows were using standard TCP. The one anomaly was TCP New Reno, latency of 50 ms, and an error rate of 10^{-6}. In this in instance, in a mixed environment, the CETEN-R enabled flow had a 14.5% increase in goodput, while the TCP New Reno flows had a 3.5% reduction in goodput.

This research shows that it is always to better to opt for a low delay, low environment when possible. CETEN-R provides an improvement in goodput when error rates are high; when error rates are low, TCP Reno or TCP New Reno is a better choice.

Significance of Research

TCP performs quite well in a traditional fixed network environment. This research focused on a technique to improve TCP goodput over nontraditional networks, such as satellite communications and wireless networks. These networks typically have longer latency and higher error rates than fixed wired networks. Satellite and wireless networks are becoming more ubiquitous and methods of improving TCP performance, thereby improving customer satisfaction, are vitally important. This research focused on developing and testing an algorithm, CETEN-R, which uses information it receives from IP about the state of the network and determines whether a lost packet was dropped due to congestion or corruption. If TCP can accurately determine a packet was lost due to corruption, it can retransmit the unacknowledged packet and dispense with the traditional congestion control mechanisms.

Recommendations for Future Research

There are several areas of further investigation that are warranted. First, the current CETEN-R algorithm needs to be implemented in conjunction with TCP SACK and its performance characterized. TCP SACK is recommended over satellite links [AGS99]. A study of CETEN performance should be conducted over a more realistic network, with multiple links and both wireless and wired hops. CETEN should be studied in an environment where not all routers are CETEN capable, since any deployment of CETEN would most likely be incremental. This research used simulated file transfers as the workload. A study of CETEN performance with more typical traffic,

such as http traffic is warranted. Finally, CETEN should be studied over a "real-world" network. Both this study and the original [KAS02] study were simulation studies.

Summary

CETEN-R is a technique developed to enhance TCP performance; this approach shows promise for improving TCP goodput over any network, but in particular over networks which are experiencing high rates of packet corruption, as would be typical over satellite and wireless networks. The main goal of this research was achieved. CETEN-R was shown to improve TCP goodput over long latency, high error rate networks. However, additional investigation is necessary to fine-tune the algorithm and explore its use with TCP SACK.

Appendix A

Algorithm Code

tcp_conn_v3 process model

tcp_ack_check function

```
static int
tcp_ack_check (void)
        {
        TcpT_Seg_Fields*            fd_ptr;
        char                                str0 [128];
        TcpT_Seq                            old_snd_una;
        TcpT_Size                           acked_bytes;
        double                              current_time;
        double                              next_timeout_time;
        char                                stra [256];

        /** Check the ACK bit and ACK sequence number of the received segment.      **/
        /** Use the segment information to update congestion window, remote         **/
        /** receive window and to flush the acknowledged data from the retrans      **/
        /** buffer. This check is used in the following states:                                 **/
        /**         ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING,
        **/
        /**                 LAST-ACK, and TIME-WAIT.
                                **/
        /** Returns 1 if ACK is acceptable, 0 otherwise.
        **/
        FIN (tcp_ack_check ());

        …

        /* Check for a duplicate acknowledgment. Duplicate ACKs are those which            */
        /* repeat an ACK sequence number already seen in a previous ACK. Hence,    */
        /* snd_una has already been advanced up to or past the seg_ack in the pkt  */
        /* just received. There are several situations which can cause dup-ACKs:   */
        /*  1) the ACK-sender could be in error or the packet might have been      */
        /*     delayed and thus received out of order.
        */
        /*  2) a TCP might repeat an ACK sequence when transmitting new data or a  */
        /*     new send window if no new data had been received between the time    */
        /*     the new packet was sent and the time the previous ACK was sent.     */
        /*  3) another possibility is that TCP is duplicating ACKs because it is   */
        /*     receiving packets but it is missing a packet prior to those being   */
        /*     received. Thus, it must still send ACKs because new data has arrived */
        /*     but the cumulative ACK cannot be advanced.  This might indicate     */
        /*     packet loss, or it might simply indicate packet reordering          */
```

```
/*     somewhere in the network.
                        */
if (tcp_seq_lt (seg_ack, snd_una))
            {
            /* This segment duplicates an ACK older than the most recently received          */
            /* ACK. Count only consecutive receptions of the most recent ACK reset           */
            /* counter, as long as Fast Retransmit has not occurred.
*/
            if (dup_ack_cnt < tcp_parameter_ptr->fr_dup_ack_thresh)
                        dup_ack_cnt = 0;

            if (tcp_trace_active || tcp_extns_trace_active)
                        op_prg_odb_print_major ("TCP received an old duplicate ACK; ignoring.",
                        OPC_NIL);

            /* Check if the incoming segment contains data.         */
            if (seg_len > 0)
                        {
                        /* Even though this segment is not in order, accept its */
                        /* data; however, dont process the other details.                */
                        FRET (1);
                        }

            else
                        {
                        FRET (0);
                        }
   }

/* Check if this segment duplicates the most recently received ACK.                          */
else if (seg_ack == snd_una)
            {
            if ((seg_len != 0) && (conn_supports_ts == TCPC_OPTION_STATUS_ENABLED))
                        {
                        /* Time stamp is supported and this is not a duplicate ACK.     */

                        /* Process timestamp information carried in the packet.          */
                        tcp_ts_info_process (ev_ptr->pk_ptr);
                        }

            /* Does this duplicate ACK contain any new data or a window update?              */
            if ((seg_len != 0) || (fd_ptr->rcv_win << snd_scale != snd_wnd))
                        {
                        if ((tcp_trace_active || tcp_extns_trace_active) && dup_ack_cnt != 0)
                                    {
                                    op_prg_odb_print_major ("TCP received a duplicate ACK containing
                                    new data or a window update.", OPC_NIL);
                                    }

                        /* Process SACK-data contained in this packet, if any.           */
                        tcp_sack_processing (ev_ptr->pk_ptr);

                        /* Reset the duplicate count, as long as Fast Retransmit has not          */
```

```
        /* already occured.
                                        */
        if (dup_ack_cnt < tcp_parameter_ptr->fr_dup_ack_thresh)
                {
                dup_ack_cnt = 0;
                }
        }
else
        {
   /* This segment is a true duplicate, i.e., no new data/window                */
        /* update. Thus, it must indicate packet drop. Now there is              */
        /* outstanding unacknowledged data which was may have been lost.         */
        if (tcp_seq_gt (snd_max, snd_una))
                {
                /* Increment the count of "pure" duplicate ACK segment.          */
                dup_ack_cnt++;
                if (tcp_trace_active || tcp_extns_trace_active)
                        {
                        sprintf (str0, "TCP received consecutive duplicate ACK
                        number %d.", dup_ack_cnt);
                        op_prg_odb_print_major (str0, OPC_NIL);
                        }
                /* Process SACK-data contained in this packet, if any.           */
                tcp_sack_processing (ev_ptr->pk_ptr);

                if ((ceten_support == 1) && tcp_ceten_packet_loss())
                        {
                        tcp_eten_retransmit();
                        }
                else
                        {

                        /* Perform fast-retransmission, if applicable. */
                        tcp_frfr_processing ();
                        }

                /* Additional packets from snd/una buffers will be sent          */
                /* if allowed by the congestion control/send window. */
                FRET (1);
                }
        else
                {
                /* Completely duplicate ACK, but there is no outstanding data so
                simply   */
                /* discard the packet.        */

                …
```

tcp_eten_retransmit function

```
static void
tcp_eten_retransmit (void)
	{
	char                                    msg [128];
	char                                    msg1 [256];
	TcpT_Seq                                onxt, cwnd_old;

	FIN (tcp_eten_retransmit (void));

	/* If RTT measurements are currently being taken, reset the timer.          */
	rtt_active = 0;

	/* Retransmit the segment lost due to error.  This will be donw by calling  */
	/* tcp_una_buf_process ().  Temporarily set the value of snd_nxt, so that   */
	/* the next sent packet is indeed the lost packet.  Then reset snd_nxt back  */
	/* to its original value.  To send only one segment, temporarily set the    */
	/* cwnd value to 1 MSS.
	                                        */

	/* Store current snd_nxt value.  This is being done as when we call         */
	/* una_buf_process.  We need to start sending from the dropped segment,     */
	/* rather than snd_nxt.  After the function call, values will be restored.  */


	if ((SACK_PERMITTED && (pipe < cwnd)) || !SACK_PERMITTED)
		{
		/* Retransmit the missing packet.  Only one will be transmitted due to cwnd. */
		onxt      = snd_nxt;
		snd_nxt = snd_una;

		/* Store current congestion window value.  This is done to send just one
		segment.*/
		cwnd_old          = cwnd;
		cwnd              = snd_mss;
		tcp_una_buf_process (OPC_FALSE);

		/* Restore the value of send_nxt.                                   */
		snd_nxt = MAX(snd_nxt, onxt);

		/* Restore the congestion window value                              */
		cwnd = cwnd_old;

		/* Collect statistics related to delays in sending segments.        */
		tcp_seg_send_delay_stat_record ();
		}
	FOUT;
```

tcp_ceten_packet_loss function

```
static Boolean
tcp_ceten_packet_loss()
	{

	double			ceten_test;
	double			scale_forward_error;
	double			scale_forward_congestion;
	Boolean		bypass_congestion = OPC_FALSE;

	FIN (tcp_ceten_packet_loss());

	if (ceten_support)
		{
		ceten_test = op_dist_uniform (1.0);
		scale_forward_error = 1 - ceten_forward_error_ratio;
		scale_forward_congestion = 1 - ceten_forward_congestion_ratio;
		if (scale_forward_error + scale_forward_congestion <= 0)
			{
			bypass_congestion = OPC_FALSE;
			}
		if (ceten_test < (scale_forward_error / (scale_forward_error +
			scale_forward_congestion)))
			{
			bypass_congestion = OPC_TRUE;
			}
		}
	FRET (bypass_congestion);
	}
```

tcp_manager_v3 process model

tcp_mgr_tcp-params_parse function

```
tcp_mgr_tcp_params_parse( )

…

/* Read in the values for CETEN attributes                                    */
if (op_ima_obj_attr_get (tcp_parameter_objid, "Ceten Status", &tcp_parameter_ptr->ceten_options_flag)
        ==  OPC_COMPCODE_FAILURE)
        {
        tcp_mgr_error ("Unabel to get CETEN Status attribute");
        }
if (op_ima_obj_attr_get (tcp_parameter_objid, "Ceten Alpha", &tcp_parameter_ptr->ceten_alpha_ratio) ==
        OPC_COMPCODE_FAILURE)
        {
        tcp_mgr_error ("Unable to get CETEN Alpha attribute.");
        }
…
```

ip_encap_v4 process model

encap state – enter executives

```
...

if (op_ici_attr_get (ul_iciptr, "ceten_options_flag", &ceten_stat) ==
        OPC_COMPCODE_FAILURE)
        {
        ip_encap_error ("Unable to get CETEN Options Flag from transport ICI.");
        }

if (op_ici_attr_get (ul_iciptr, "ceten_alpha_ratio", &ceten_alpha_value) ==
        OPC_COMPCODE_FAILURE)
        {
        ip_encap_error ("Unable to get CETEN Alpha Ratio from transport ICI.");
        }

if (op_ici_attr_get (ul_iciptr, "ceten_cumulative_probability", &ceten_cum_prob) ==
        OPC_COMPCODE_FAILURE)
        {
        ip_encap_error ("Unable to get CETEN cumulative probability from transport ICI.");
        }

/* If the destination address is multicast, then we need to retrieve          */
/* major and minor ports, which the higher layer specifies.                   */
if (inet_address_is_multicast (dest_addr) &&  (protocol_type != IpC_Protocol_Rsvp))
        {
        if (op_ici_attr_get (ul_iciptr, "multicast_major_port", &mcast_major_port) ==
                OPC_COMPCODE_FAILURE)
                {
                mcast_major_port = 0;
```

```
            ipnl_protwarn_mcast_no_major_port_specified (pkptr, dest_addr);

            /* inet_address_print (dest_addr_str, dest_addr); */
            /* sprintf (error_string, "Unable to retrieve multicast major port for multicast
                        address (%s)", */
            /*          dest_addr_str);
*/
            /* ip_encap_error (error_string);                                    */
            }

if (op_ici_attr_get (ul_iciptr, "multicast_minor_port", &mcast_minor_port) ==
            OPC_COMPCODE_FAILURE)
            {
            inet_address_print (dest_addr_str, dest_addr);
            sprintf (error_string, "Unable to retrieve multicast minor port for multicast
                        address (%s)",
                        dest_addr_str);
            ip_encap_error (error_string);
            }

/* Prepare an ICI that is to be sent to ip_dispatch,indicating the major*/
/* and minor ports on which to send the multicast packet.                                    */
ip_iciptr = op_ici_create ("ip_rte_req_v4");
op_ici_attr_set (ip_iciptr, "multicast_major_port", mcast_major_port);
op_ici_attr_set (ip_iciptr, "multicast_minor_port", mcast_minor_port);

op_ici_attr_set (ip_iciptr, "ceten_status", ceten_stat);
op_ici_attr_set (ip_iciptr, "ceten_alpha", ceten_alpha_value);
op_ici_attr_set (ip_iciptr, "ceten_cum_prob", ceten_cum_prob);
op_ici_install (ip_iciptr);
}
else if (protocol_type == IpC_Protocol_Isis)
{
/* Get the output index from the incoming ICI
*/
if (op_ici_attr_get (ul_iciptr, "out_intf_index", &isis_out_intf_index) ==
            OPC_COMPCODE_FAILURE)
            {
            sprintf (error_string, "Unable to retrieve the ISIS packet's output index");
            ip_encap_error (error_string);
            }

/* Prepare an ICI that is to be sent to ip_dispatch,indicating the             */
/* output index in the major port
*/
ip_iciptr = op_ici_create ("ip_rte_req_v4");
op_ici_attr_set (ip_iciptr, "out_intf_index", isis_out_intf_index);

op_ici_attr_set (ip_iciptr, "ceten_status", ceten_stat);
op_ici_attr_set (ip_iciptr, "ceten_alpha", ceten_alpha_value);
op_ici_attr_set (ip_iciptr, "ceten_cum_prob", ceten_cum_prob);

/* Install this ICI
                        */
```

```
            op_ici_install (ip_iciptr);


            /* The ISIS packets don't use the IP header, so reduce the IP          */
            /* header length from the packet bulk length                          */
            data_len -= IPC_DGRAM_HEADER_LEN_BYTES;
            }
else if (protocol_type != IpC_Protocol_Rsvp)
            {
            ip_iciptr = op_ici_create ("ip_rte_ind_v4");

            intf_ici_fdstruct_ptr = ip_rte_ind_ici_fdstruct_create ();

            intf_ici_fdstruct_ptr->ceten_status = ceten_stat;
            intf_ici_fdstruct_ptr->ceten_alpha = ceten_alpha_value;
            intf_ici_fdstruct_ptr->ceten_cumulative_probability = ceten_cum_prob;

            op_ici_attr_set (ip_iciptr, "rte_info_fields", &intf_ici_fdstruct_ptr);
            op_ici_install (ip_iciptr);

            }

/* If this is an RSVP packet, also get Next hop Address and Interface index   */
if (protocol_type == IpC_Protocol_Rsvp)
            {
            if (op_ici_attr_get (ul_iciptr, "RSVP Packet Route Info", &pkt_route_info_ptr) ==
                        OPC_COMPCODE_FAILURE)
                        {
                        ip_encap_error ("Unable to get routing information from transport ICI.");
                        }

            /* Prepare an ICI that is to be sent to ip_dispatch, indicating interface on      */
            /* which to send the RSVP packet so as IP does not do route query                 */
            ip_iciptr = op_ici_create ("ip_rte_req_v4");

            op_ici_attr_set (ip_iciptr, "RSVP Packet Route Info", pkt_route_info_ptr);

            op_ici_attr_set (ip_iciptr, "ceten_status", ceten_stat);
            op_ici_attr_set (ip_iciptr, "ceten_alpha", ceten_alpha_value);
            op_ici_attr_set (ip_iciptr, "ceten_cum_prob", ceten_cum_prob);
            op_ici_install (ip_iciptr);

            /* Destroy the ICI only for RSVP packets.    */
            op_ici_destroy (ul_iciptr);
            }

…
```

Decap state – enter executives

```
…
/* Get data from lower level ici */
ceten_stat = intf_ici_fdstruct_ptr->ceten_status;
ceten_alpha_value = intf_ici_fdstruct_ptr->ceten_alpha;
ceten_cum_prob = intf_ici_fdstruct_ptr->ceten_cumulative_probability;
…

/* Set info in higher layer ici */

if (op_ici_attr_set (transp_iciptr, "ceten_status", ceten_stat) == OPC_COMPCODE_FAILURE)
        ip_encap_error ("Unable to set ceten status field in transport layer ICI.");
if (op_ici_attr_set (transp_iciptr, "ceten_alpha", ceten_alpha_value) == OPC_COMPCODE_FAILURE)
        ip_encap_error ("Unable to set ceten alpha field in transport layer ICI.");
if (op_ici_attr_set (transp_iciptr, "ceten_cumulative_probability", ceten_cum_prob) ==
        OPC_COMPCODE_FAILURE)
        ip_encap_error ("Unable to set ceten cumulative probability field in transport layer ICI.");

…
```

ip_rte_central_cpu process model

ip_rte_central_cpu_packet_arrival function

```
static void
ip_rte_central_cpu_packet_arrival (void)
        {
        Packet *                        pkptr = OPC_NIL;
        int                             instrm;
        Ici *                           iciptr;
        IpT_Rte_Ind_Ici_Fields *        intf_ici_fdstruct_ptr = OPC_NIL;
        IpT_Interface_Info *            rcvd_iface_info_ptr = OPC_NIL;
        int                             result;
        char                            stra [256];
        char                            strb [256];
        char                            strc [256];
        char                            strd [256];
        char                            stre [256];
        char                            strf [256];
        char                            format_name[128];
        char                            target_format[128] = "tcp_seg_v2\0";
        IpT_Dgram_Fields *              packet_fields_ptr;
        Packet *                        ul_pkptr = OPC_NIL;
        Packet *                        ul_pkptr_copy = OPC_NIL;
        OpT_Packet_Size                 packet_size;

        /** An incoming packet has arrived.  It might       **/
        /** be from an "upper layer", a "lower layer",      **/
        /** or generated from within ip.                            **/
        FIN (ip_rte_central_cpu_packet_arrival ());
```

```
if (invoke_mode == OPC_PROINV_INDIRECT)
	{
	/* Packet generated from withing IP and forwarded by our	*/
	/* parent process.
	*/
	pkptr = (Packet *)op_pro_argmem_access ();
	instrm = IpC_Pk_Instrm_Child;

	/* if (op_sim_debug () == OPC_TRUE)
		{
		op_prg_odb_print_major ("packet generated from within IP and forwarded by
			parent process");
		}
	}
else
	{
	/* Packet coming from some stream */
	instrm = op_intrpt_strm ();
	pkptr = op_pk_get (instrm);
	if (pkptr == OPC_NIL)
		ip_rte_cpu_error ("Unable to get packet from input stream.");
	else
		{
		/* Check if GTP encapsulation is enable on this node */
		/* then the GTP module must process this packet.		*/
		if (module_data_ptr->gtp_status == OPC_TRUE)
			{
			/* Prepare shared memory with the arriving packet.	*/
			module_data_ptr->ip_ptc_mem.child_pkptr = pkptr;
			module_data_ptr->ip_ptc_mem.pk_processed_by_gtp = OPC_FALSE;

			/* Invoke GTP process model, which processes the	*/
			/* GTP packet contained in the IP datagram.			*/
			op_pro_invoke (module_data_ptr->gtp_process_handle, OPC_NIL);

			/* The packet will be processed by IP if GTP didn't	*/
			/* take control of it.					*/
			if (module_data_ptr->ip_ptc_mem.pk_processed_by_gtp ==
				OPC_TRUE)
				FOUT;

			/* Get the packet that the child process sent.		*/
			pkptr = module_data_ptr->ip_ptc_mem.child_pkptr;
			}
		}
	}

/* Make sure we care about this packet */
if (ip_rte_packet_format_valid (module_data_ptr, pkptr) == OPC_FALSE)
	{
	FOUT;
	}
```

```
/* Perform standard IP processing of incoming packet            */
/*          1. Perform forwarding decision and populate the     ICI      */
/*          2. Populate rcvd_iface_info_ptr. It is set to NIL   */
/*             if packet arrives from higher layer.                                  */
/* op_pk_print (pkptr);     */

result = ip_rte_packet_arrival (module_data_ptr,
          &pkptr, instrm, &intf_ici_fdstruct_ptr, &rcvd_iface_info_ptr);

/*op_pk_print (pkptr);*/

op_pk_nfd_access (pkptr, "fields", &packet_fields_ptr);
if ((packet_fields_ptr->protocol == 6) && (op_pk_nfd_is_set (pkptr, "data")))
          {
          op_pk_fd_get_pkt (pkptr, 3, &ul_pkptr);
          ul_pkptr_copy = op_pk_copy (ul_pkptr);
          op_pk_nfd_set (pkptr, "data", ul_pkptr, op_prg_mem_copy_create, op_prg_mem_free,
                    packet_size);
          packet_size = op_pk_total_size_get (ul_pkptr_copy);
          if (ul_pkptr == OPC_NIL)
                    {
                    }
          else
                    {
                    op_pk_format (ul_pkptr_copy, format_name);
                    if (strcmp(format_name, target_format) == 0)
                    {
                    intf_ici_fdstruct_ptr->ceten_status = get_CETEN_status (ul_pkptr_copy);
                    intf_ici_fdstruct_ptr->ceten_alpha = get_CETEN_alpha (ul_pkptr_copy);
                    if (intf_ici_fdstruct_ptr->ceten_status == 1)
                    {
                    if (op_td_is_set (pkptr, OPC_TDA_PT_NUM_ERRORS))
                    {
                    if (op_td_get_int (pkptr, OPC_TDA_PT_NUM_ERRORS) > 0)
                    {
                    intf_ici_fdstruct_ptr->ceten_cumulative_probability = (1.0 -
                              intf_ici_fdstruct_ptr->ceten_alpha) *
                              intf_ici_fdstruct_ptr->ceten_cumulative_probability;
                    }
                    else
                    {
                    intf_ici_fdstruct_ptr->ceten_cumulative_probability =
                              intf_ici_fdstruct_ptr->ceten_alpha + ((1.0 –
                              intf_ici_fdstruct_ptr->ceten_alpha) *
                              intf_ici_fdstruct_ptr->ceten_cumulative_probability);
                    }
                    }
                    else
                    {
                    intf_ici_fdstruct_ptr->ceten_cumulative_probability =
                              intf_ici_fdstruct_ptr->ceten_alpha + ((1.0 –
```

```
                                        intf_ici_fdstruct_ptr->ceten_alpha) *
                                        intf_ici_fdstruct_ptr->ceten_cumulative_probability);
                                }
                                adjust_forward_CETEN_survival_ratio (ul_pkptr_copy,
                                        intf_ici_fdstruct_ptr->ceten_cumulative_probability);
                                }
                                }
                                }
                        op_pk_destroy (ul_pkptr_copy);
                        }

                if (result == OPC_FALSE)
                        {
                        /* Packet was dropped in call */
                        FOUT;
                        }

                /* Attempt to place new packet in pending queue */
                if (oms_buffer_bgutil_enqueue (routing_buffer, pkptr)
                                != OmsC_Buffer_Enqueue_Success)
                        {
                        /*      The insertion failed (due to a full buffer).    */
                        char intf_addr_str [IPC_ADDR_STR_LEN];

                        /*  Thesis modification                                 */

                        if (intf_ici_fdstruct_ptr->ceten_status == 1)
                        {
                        intf_ici_fdstruct_ptr->ceten_cumulative_probability = (1.0 –
                                intf_ici_fdstruct_ptr->ceten_alpha) *
                                intf_ici_fdstruct_ptr->ceten_cumulative_probability;
                        adjust_forward_CETEN_congestion_ratio (ul_pkptr, intf_ici_fdstruct_ptr->ceten_status);
                        }


                                /*  Get a printable version of the interface addr.      */
                        if (rcvd_iface_info_ptr == OPC_NIL)
                                sprintf (intf_addr_str, "Higher Layer");
                        else
                                ip_address_print (intf_addr_str,
                                        rcvd_iface_info_ptr->addr_range_ptr->address);

                        /*      Issue a warning message to the sim. log.            */
                        ipnl_reswarn_pktinsert (op_pk_id (pkptr),
                                op_pk_tree_id (pkptr), intf_addr_str);

                        /* Update packets dropped statistics and destroy        */
                        /* the IP datagram.
                */
                        ip_rte_dgram_discard (module_data_ptr, pkptr, op_pk_ici_get (pkptr), "Buffer
                                overflow");
                        }
                FOUT;
}
```

tcp_seg_support – header file

```
/** tcp_seg_sup.h                                                    **/

/***************************************/
/*      Copyright (c) 1987 - 2002      */
/*                  by OPNET Technologies, Inc.      */
/*      (A Delaware Corporation)       */
/*    7255 Woodmont Av., Suite 250     */
/*     Bethesda, MD 20814, U.S.A.      */
/*       All Rights Reserved.          */
/***************************************/

/* Protect against multiple includes.           */
#ifndef   _TCP_SEG_SUP_H_INCLUDED_
#define _TCP_SEG_SUP_H_INCLUDED_

/** Include directives.       **/
#include <opnet.h>
#include "oms_dt.h"
#include "tcp_v3.h"

#if defined (__cplusplus)
extern "C" {
#endif

/* Size of TCP Timestamp option in bytes.   */
#define TCPC_SEG_TIMESTAMP_SIZE              12

/* Size (in bytes) of kind-length block in the option fields of the TCP header.     */
#define TCPC_KIND_LENGTH_BLOCK_SIZE                          2

/* Size of CETEN option in bytes                */
#define TCPC_SEG_CETEN_SIZE                                  32

/* Data structure for fields in the     */
/* tcp segment.                                 */
typedef struct
   {
        int                             src_port;
        int                             dest_port;
        unsigned int        seq_num;
        unsigned int        ack_num;
        unsigned int        rcv_win;
        int                             urgent_pointer;
        int                             data_len;

        /* The following represents bytes 13 and 14 of the TCP header.          */
        /*
                                                        */
        /*   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15          */
        /*  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+      */
        /*  |           |           | C | E | U | A | P | R | S | F |      */
        /*  | Header Length |   Reserved   | W | C | R | C | S | S | Y | I |   */
```

```
        /*  |           |           | R | E | G | K | H | T | N | N |    */
        /*  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+       */
        /*
                                                                        */
        TcpT_Flag               flags;          /* Information from bits 8 through 15.
        */

        /* The following two fields are not part of    */
        /* the standard TCP header, but are used in    */
        /* the OPNET TCP model for enhancing the */
        /* simulation performance. Each TCP process    */
        /* assigns itself a unique "key" which can     */
        /* be used to perform fast lookup when ever */
        /* is a need to match a connection ID.         */
        OmsT_Dt_Key             local_key;
        OmsT_Dt_Key             remote_key;


    } TcpT_Seg_Fields;

/* Data structure for segment field containing TCP timestamp option.    */
typedef struct
        {
        unsigned int    timestamp_value;/* Timestamp value                          */
        unsigned int    timestamp_echo;         /* Timestamp echo reply.  */
        } TcpT_Seg_Option_TS;

/* The following fields are not part of        */
/* the standard TCP header, but instead of     */
/* added for thesis research   */

typedef struct
        {
        int                                     ceten_value;
        double                                  ceten_alpha;
        double                                  forward_ceten_survival_ratio;
        double                                  backward_ceten_survival_ratio;
        double                                  forward_ceten_congestion_ratio;
        double                                  backward_ceten_congestion_ratio;
        } TcpT_Seg_CETEN_Option;


/* Function Prototypes.     */
TcpT_Seg_Fields*        tcp_seg_fdstruct_create (void);
TcpT_Seg_Fields*        tcp_seg_fdstruct_copy (TcpT_Seg_Fields* pk_fd_ptr);
void    tcp_seg_fdstruct_destroy (TcpT_Seg_Fields* pk_fdstruct_ptr);
void    tcp_seg_fdstruct_print (TcpT_Seg_Fields* pk_fdstruct_ptr);
void    tcp_seg_fields_pkprint (void* arg_field_ptr, Prg_List* list);
void    tcp_seg_msg_print (const char* dir_str, TcpT_Seq seq, TcpT_Seq ack_num, TcpT_Size len,
                TcpT_Flag flags);
void    tcp_seg_timestamp_set (Packet* seg_ptr, unsigned int echoed_timestamp, unsigned int
        my_timestamp);
void    tcp_seg_timestamp_pkprint (void* arg_field_ptr, Prg_List* output_list);
```

```
/* Function Prototypes for CETEN. */
void                tcp_seg_CETEN_set (Packet* seg_ptr, int ceten_val, double ceten_alpha_value);
void                tcp_seg_CETEN_pkprint (void* arg_field_ptr, Prg_List* output_list);
void                adjust_forward_CETEN_survival_ratio (Packet* seg_ptr, double adjustment);
void                adjust_backward_CETEN_survival_ratio (Packet* seg_ptr, double adjustment);
void                adjust_forward_CETEN_congestion_ratio (Packet* seg_ptr, double adjustment);
void                adjust_backward_CETEN_congestion_ratio (Packet* seg_ptr, double adjustment);
unsigned int                        get_CETEN_status (Packet *seg_ptr);
double                              get_CETEN_alpha (Packet *seg_ptr);
double                              get_forward_CETEN_congestion_ratio ();
double                              get_backward_CETEN_congestion_ratio ();
double                              get_forward_CETEN_survival_ratio ();
double                              get_backward_CETEN_survival_ratio ();

/* End function prototypes for CETEN.        */

#if defined (__cplusplus)
} /* end of 'extern "C" {' */
#endif

/* End if for protection against multiple includes. */
#endif /* _TCP_SEG_SUP_H_INCLUDED_ */
```

tcp_seg_support – C Code

…

```
void
tcp_seg_CETEN_set (Packet* seg_ptr, int ceten_val, double ceten_alpha_value)
        {
        TcpT_Seg_CETEN_Option*          ceten_field_ptr;

        /** Create a CETEN field in the segment and set it     **/
        FIN (tcp_seg_CETEN_set (seg_ptr, ceten_val, ceten_alpha_value));
        /*  If the pooled memory object has not yet been defined, do    */
        /* so now, prior to allocation.
        */
        if (pk_ceten_pmo_defined == OPC_FALSE)
                {
                /* Prevent redundant definition.      */
                pk_ceten_pmo_defined = OPC_TRUE;

                tcp_seg_ceten_pmh = op_prg_pmo_define ("TCP Seg CETEN Option", sizeof
                        (TcpT_Seg_CETEN_Option), 1000);
                /* Set the packet print procedure for "ETEN Capability" field print.       */
                op_pk_format_print_proc_set ("tcp_seg_v2", "Ceten Option", tcp_seg_CETEN_pkprint);
                }
        /* Allocate memory for the field.                */
        ceten_field_ptr = (TcpT_Seg_CETEN_Option *) op_prg_pmo_alloc (tcp_seg_ceten_pmh);
        if (ceten_field_ptr == OPC_NIL)
                {
                op_sim_end ("Error in TCP segment support code:",
```

```
                        "Unable to allocate memory for TCP segment CETEN option.",
                        OPC_NIL, OPC_NIL);
                }
        /*Set survival and congestion values.                                   */
        ceten_field_ptr->ceten_value = ceten_val;
        ceten_field_ptr->ceten_alpha = ceten_alpha_value;
        ceten_field_ptr->forward_ceten_survival_ratio = 1.0;
        ceten_field_ptr->backward_ceten_survival_ratio = 1.0;
        ceten_field_ptr->forward_ceten_congestion_ratio = 0.0;
        ceten_field_ptr->backward_ceten_congestion_ratio = 0.0;
        /* Set the field in the packet.                 */
        op_pk_nfd_set (seg_ptr, "Ceten Option", ceten_field_ptr, op_prg_mem_copy_create,
                op_prg_mem_free, sizeof (TcpT_Seg_CETEN_Option));
        FOUT;
        }

void
adjust_forward_CETEN_survival_ratio (Packet* seg_ptr, double adjustment)
        {
        TcpT_Seg_CETEN_Option*          ceten_field_ptr;

        FIN (adjust_forward_CETEN_survival_ratio (seg_ptr, adjustment));
        op_prg_odb_bkpt ("adjust.forward.survival.ratio");

        op_pk_nfd_get (seg_ptr, "Ceten Option", &ceten_field_ptr);
        ceten_field_ptr->forward_ceten_survival_ratio = adjustment *
                ceten_field_ptr->forward_ceten_survival_ratio;
        op_pk_nfd_set (seg_ptr, "Ceten Option", ceten_field_ptr, op_prg_mem_copy_create,
                op_prg_mem_free, sizeof (TcpT_Seg_CETEN_Option));

        FOUT;
        }

void
adjust_backward_CETEN_survival_ratio (Packet* seg_ptr, double adjustment)
        {
        TcpT_Seg_CETEN_Option*          ceten_field_ptr;

        FIN (adjust_backward_CETEN_survival_ratio (seg_ptr, adjustment));

        ceten_field_ptr->backward_ceten_survival_ratio = adjustment *
                ceten_field_ptr->backward_ceten_survival_ratio;
        op_pk_nfd_set (seg_ptr, "Ceten Option", ceten_field_ptr, op_prg_mem_copy_create,
        op_prg_mem_free, sizeof (TcpT_Seg_CETEN_Option));

        FOUT;
        }

void
adjust_forward_CETEN_congestion_ratio (Packet *seg_ptr, double adjustment)
        {
        TcpT_Seg_CETEN_Option*                  ceten_field_ptr;

        FIN (adjust_forward_CETEN_congestion_ratio (seg_ptr, adjustment));
```

```
	ceten_field_ptr->forward_ceten_congestion_ratio = adjustment *
		ceten_field_ptr->forward_ceten_congestion_ratio;
	op_pk_nfd_set (seg_ptr, "Ceten Option", ceten_field_ptr, op_prg_mem_copy_create,
	op_prg_mem_free, sizeof (TcpT_Seg_CETEN_Option));

	FOUT;
	}

void
adjust_backward_CETEN_congestion_ratio (Packet *seg_ptr, double adjustment)
	{
	TcpT_Seg_CETEN_Option*		ceten_field_ptr;

	FIN (adjust_backward_CETEN_congestion_ratio (seg_ptr, adjustment));

	ceten_field_ptr->backward_ceten_congestion_ratio = adjustment *
		ceten_field_ptr->backward_ceten_congestion_ratio;
	op_pk_nfd_set (seg_ptr, "Ceten Option", ceten_field_ptr, op_prg_mem_copy_create,
	op_prg_mem_free, sizeof (TcpT_Seg_CETEN_Option));

	FOUT;
	}

unsigned int
get_CETEN_status (Packet *seg_ptr)
	{
	TcpT_Seg_CETEN_Option*		ceten_field_ptr;
	int				ceten_val;

	FIN (get_CETEN_Status (seg_ptr));

	op_pk_nfd_get (seg_ptr, "Ceten Option", &ceten_field_ptr);
	ceten_val = ceten_field_ptr->ceten_value;
	op_pk_nfd_set (seg_ptr, "Ceten Option", ceten_field_ptr, op_prg_mem_copy_create,
		op_prg_mem_free, sizeof (TcpT_Seg_CETEN_Option));

	FRET (ceten_val);
	}

double
get_CETEN_alpha (Packet *seg_ptr)
	{
	TcpT_Seg_CETEN_Option*		ceten_field_ptr;
	double				ceten_alpha_value;

	FIN (get_CETEN_alpha (seg_ptr));

	op_pk_nfd_get (seg_ptr, "Ceten Option", &ceten_field_ptr);
	ceten_alpha_value = ceten_field_ptr->ceten_alpha;
	op_pk_nfd_set (seg_ptr, "Ceten Option", ceten_field_ptr, op_prg_mem_copy_create,
		op_prg_mem_free, sizeof (TcpT_Seg_CETEN_Option));
	FRET (ceten_alpha_value);
	}
```

```
void
tcp_seg_CETEN_pkprint (void* arg_field_ptr, Prg_List* output_list)
	{
	char                                    temp_str[128];
	char*                                   alloc_str;
	TcpT_Seg_CETEN_Option*
			ceten_option_ptr = (TcpT_Seg_CETEN_Option*) arg_field_ptr;

	/* Print the CETEN options as specified in the packet.                  */
	FIN (tcp_seg_CETEN_pkprint (ceten_option_ptr, output_list));

	sprintf(temp_str, "                     ceten_value                                     unint
				%-16u(32)", ceten_option_ptr->ceten_value);
	PKPRINT_STRING_INSERT (alloc_str, temp_str, output_list)
	sprintf(temp_str, "                     ceten_alpha                                     double
				%-16f(32)", ceten_option_ptr->ceten_alpha);
	PKPRINT_STRING_INSERT (alloc_str, temp_str, output_list)

	sprintf(temp_str, "                     forward_survival_ratio          double
			%-16f(32)", ceten_option_ptr->forward_ceten_survival_ratio);
	PKPRINT_STRING_INSERT (alloc_str, temp_str, output_list)
	sprintf(temp_str, "                     backward_survival_ratio         double
			%-16f(32)", ceten_option_ptr->backward_ceten_survival_ratio);
	PKPRINT_STRING_INSERT (alloc_str, temp_str, output_list)
	sprintf(temp_str, "                     forward_congestion_ratio  double
			%-16f(32)", ceten_option_ptr->forward_ceten_congestion_ratio);
	PKPRINT_STRING_INSERT (alloc_str, temp_str, output_list)
	sprintf(temp_str, "                     backward_congestion_ratio       double
			%-16f(32)", ceten_option_ptr->backward_ceten_congestion_ratio);
	PKPRINT_STRING_INSERT (alloc_str, temp_str, output_list)

	FOUT;
	}
```

Bibliography

[ADG00] Allman, Mark, Dawkins, Spencer, Glover, Dan, Griner, Jim, Tran, Diepchi, Henderson, Tom, Heidemann, John, Touch, Joe, Kruse, Hans, Osterman, Shawn, Scott, Keith and Semke, Jeffrey. "Ongoing TCP Research Related to Satellites", RFC 2760, February 2000.

[AGS99] Allman, Mark, Glover, Daniel L. and Sanchez, Luis A. "Enhancing TCP over Satellite Channels using Standard Mechanisms", RFC2488, January 1999.

[BCC98] Braden, Bob, Clark, David D., Crowcroft, Jon, Davie, Bruce, Deering, Steve, Estrin, Deborah, Floyd, Sally, Jacobson, Van, Minshall, Greg, Partridge, Craig, Peterson, Larry, Ramakrishnan, K. K., Shenker, Scott, Wroclawski, John, Zhang, Lixia. "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[FaF96] Fall, Kevin and Floyd, Sally. "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", ACM Computer Communication Review, V. 26, N. 3, July 1996, p. 5-21.

[FlJ93] Floyd, Sally and Jacobson, Van. "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413.

[Flo94] Floyd, Sally. "TCP and Explicit Congestion Notification", ACM Computer Communication Review, V. 24, N. 5, October 1994, p. 8-23.

[FlH99] Floyd, Sally and Henderson, Tom. "The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 2582, April 1999.

[FMM00] Floyd, Sally, Mahdavi, Jamshid, Mathis, Matt and Podolsky. "An Extension to the Selective Acknowledgement (SACK) Option for TCP, RFC 2883, July 2000.

[Jac88] Jacobson, Van. "Congestion Avoidance and Control", SIGCOMM Symposium on Communications Architectures and Protocols, p. 314-329, 1988.

[Jac90] Jacobson, Van. "Modified TCP Congestion Avoidance Algorithm", Technical report, 30 April 1990.

[Jai91] Jain, Raj. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* New York: John Wiley & Sons, 1991.

[JBB92] Jacobson, Van, Braden, Robert and Borman, David. "TCP Extension for High Performance", RFC 1323, May 1992.

[KAP02] Krishnan, Rajesh, Allman, Mark, Partridge, Craig and Sterbenz, James P.G. "Explicit Transport Error Notification (ETEN) for Error-Prone Wireless and Satellite Networks", Technical Report TR-8333, BBN Technologies, March 2002.

[MMF96] Mathis, Matt, Mahdavi, Jamshid, Floyd, Sally, and Romanow, Allyn. "TCP Selective Acknowledgement Options, RFC 2018, October 1996.

[PFT98] Padhye, Jitendra, Firoiu, Victor, Towsley, Don and Kurose, Jim. "Modeling TCP Throughput: A Simple Model and its Empirical Validation", ACM Computer Communication Review, V. 28, N. 4, October 1998, p. 303-314.

[PaS97] Patridge, Craig and Shepard, Timothy J. "TCP/IP Performance over Satellite Links", IEEE Network, September/October 1997, p. 44-49.

[RFB01] Ramakrishan, K., Floyd, Sally, Black. "The Addition of Explicit Congestion Notification for IP", RFC3168, September 2001.

[SaA00] Salim, Jamal Hadi, and Ahmed, Uvaiz. "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks, RFC 2884, July 2000.

[Ste94] Stevens, W. Richard. *TCP/IP Illustrated, Volume I: The Protocols.* Addison Wesley, 1994.

[Ste97] Stevens, W. "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001, January 2001.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
14-09-2004	Master's Thesis	March 2003 – July 2004

4. TITLE AND SUBTITLE

Improving TCP Performance by Estimating Errors in a Long Delay, High Error Rate Environment

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Carroll, Stephanie E., SMSgt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/04-04

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Emil Tejkowski DSN 576-5167
Air Force Communications Agency
203 West Losey Street Room 1065
Scott AFB, IL 62225

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Interest in finding methods of improving TCP performance over satellite and wireless networks is high. This has been an active area of research within the networking community. This research develops an algorithm, CETEN-R for TCP to determine if a particular packet is lost due to congestion or corruption and react accordingly. An analysis of the performance of CETEN-R under a variety of conditions is studied and then compared to TCP Reno and TCP New Reno. When delay is high and the error rate is high CETEN-R showed a 77.5% increase in goodput over TCP New Reno and a 33.8% increase in goodput over TCP Reno. When delay is low and the error rate is high, CETEN-R showed a 146% increase in goodput over TCP New Reno and a 77% increase in goodput over TCP Reno. At low error rates, CETEN-R provides no advantage over TCP Reno or TCP New Reno.

15. SUBJECT TERMS

Networks, Satellite Communications, Satellite Networks, Internet, Communications Protocols, Wireless Communications

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Rusty O. Baldwin, AFIT/ENG
U	U	U	UU	103	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, ext 4445 (rusty.baldwin@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18